Iterative solutions of large sparse matrices arising from groundwater flow problems

Hagger, Mark Julian

Award date:
1995

Awarding institution:
University of Bath

Iterative solutions of large sparse matrices arising from groundwater flow problems

submitted by

Mark Julian Hagger

for the degree of Ph.D

of the

University of Bath

1995

Signature of Author ..... M J Hagger .....

Mark Julian Hagger

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Summary

In this thesis we consider iterative solutions of the large sparse symmetric positive definite linear systems arising from finite element discretisations of a groundwater flow model. Realistic rock formations give rise to models which feature highly discontinuous coefficients caused by variations in rock type. In addition highly unstructured grids are used to produce accurate models, typically with large numbers of freedoms. These features combine to produce very large, and poorly conditioned, linear systems, hence standard iterative methods can perform very poorly. Two main problems are considered, one in 2D and one in 3D, each arising from an actual physical problem.

We first investigate the use of conjugate gradients as an iterative solver with simple diagonal preconditioning. A two grid method is discussed with a number of "matrix-free" smoothers, including conjugate gradients. Additionally the two grid method is extended to a three grid method, and we also test the effectiveness of the two grid method as a preconditioner to conjugate gradients. A possible extension to the two grid method for non-symmetric problems is briefly considered, using conjugate gradients on the normal equations as the smoother.

The use of polynomial preconditioners is examined in some detail, we discuss both the least squares and Chebyshev polynomials. Finally the incomplete factorisation preconditioner for the conjugate gradient algorithm is examined, and an "off the shelf" implementation is tested for effectiveness.

Dedicated to the memory of my sister, Joanne Francis Oliver.

I'd like to thank loads of people who have given me much help and support in the last few years. In no particular order:

- My supervisor, Alastair Spence, for many useful comments and suggestions and for patiently correcting my continual use of "ing".
- My industrial supervisor, Andrew Cliffe, for his helpful hints on how to make NAMMU perform in a way that I would like. Plus of course his tireless help in getting me set up each time I arrived at Harwell to find one machine or other had "changed". Thanks also go to both the Theoretical Physics division at Harwell and the EPSRC for providing funding for me for the last three years.
- My parents and family, for not looking too bemused when I explained (no doubt very badly) what I have been doing in the last few years. Thanks for all their support and encouragement.
- My office mates of 1W3.5: Rob Slade for introducing me to C and accompanying me for many coffee breaks. Simon Juden for numerous things, including convincing me to try cricket. Rob Collins, for introducing me to Murphys. Jon Ashman for his amusing stories. Thanks to the others on the BBB development team for many hours of enjoyment.
- Thanks to all those who have shared a house with me over the last few years, Claire and Philippa in particular, one way or another you've all kept me sane.
- The support staff for their many comments and help over the last few years.
- Anyone else that I've missed.

"A mathematician is a device for turning coffee into theorems" – P. Erdos

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The aim of the work

Finite elements are used in a great many applications to discretise partial differential equations (pdes). The work involved in setting up the problem in a finite element setting varies considerably, depending on

- the form of the pde
- the accuracy of the solution required
- the type of basis element used
- the nature of the grid over the domain
- the smoothness of the (unknown) solution.

Irrespective of these points, it will generally become necessary, at some core level, to solve linear systems of the form

$$A\mathbf{x} = \mathbf{b}. \tag{1.1.1}$$

Here, $A \in \mathbb{R}^{N\times N}$ is an $N \times N$ matrix, and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^N$.

Modern demands involve solving ever more challenging problems. These often feature more complicated pdes or systems of pdes and require a greater refinement of grid, perhaps to give more accurate solutions, or model larger domains. The net result is that the size of the linear system, N, is made increasingly larger.

Traditional numerical techniques for solving (1.1.1) involve performing Gaussian elimination, albeit in a sophisticated form. These methods can be generally made very robust, as long as the system is not too badly conditioned, and give an exact solution. The major disadvantage of these approaches is in the scaling of the work as the problem becomes larger. In particular for 3D problems, even modest discretisations produce large systems, into the millions and beyond, so that direct solution techniques are completely unfeasible, since they would require huge amounts of storage and computer processing time. With the relatively recent advent of vector processing, distributed processing, massively parallel computer architectures and many other computing hardware advances a further difficulty with direct methods is coming to light. It is generally very difficult to write direct solution software that takes full, or even a reasonable, advantage of these computer hardware technologies.

As a result of these points, attention, over the last decades, has increasingly focussed on iterative methods, in order to find a good approximation to the solution of (1.1.1). Typically we start with a guess to the solution, update it in some fashion and continue until we decide that we have a "good enough" solution. The great advantage of the majority of iterative methods is that the computational work involved scales at a much slower rate than that for direct solvers. A second, and very important advantage, is that iterative processes generally can be adapted to utilise well the features of more modern computer architectures.

In the majority of the academic world iterative processes for solving large linear systems are the *de facto* standard, it is very rare to find modern research codes using direct solving techniques for large systems of the form (1.1.1). In the industrial world there often exists a very different story. The typical industrial code is sold to a number of users and it is necessary to be able to guarantee that it will work in most cases. It is here that the main drawback of the iterative process lies. Research codes typically mention words like "effective preconditioning" or "poor convergence rates". These terms are very

worrying from the point of view of the industrialist, who requires a product that works all the time with little, or no, fine tuning effort from users of the package. Nonetheless, the cost savings of using iterative techniques make them a very attractive approach.

Clearly the answer is to produce an iterative method that is guaranteed to work well for any problem. Unfortunately this has yet to be achieved, indeed different methods are known to work well on certain classes of problems, but fail on others.

This project is motivated by the need for such an implementation for solving large linear systems arising from discretisations of groundwater flow problems (see §1.2.2) in the Harwell AEA Technology finite element code called NAMMU. The basic aim is to investigate a number of iterative schemes, to determine which approach gives the best performance. The iterative scheme will be required to significantly outperform the direct method before it can be considered as a replacement technique. This is more likely to happen for the much larger three dimensional problems. For the two dimensional case merely producing a method that requires the same amount of computational time as the direct method will be a fair indication of success. The hope is to produce an iterative solver that could replace the direct solver as a solution technique for a general linear system. Essentially we are then looking for a "black box" approach that could be applied to any problem. It is therefore not feasible to make any assumptions about the underlying problem, eliminating, for example, any possibility of using some form of a domain decomposition technique (see [44, §11] for a introduction to domain decomposition methods).

Chapters 2 and 3 are concerned with an iterative method that meets these criteria more than adequately, for a certain problem. The iterative method discussed there outperforms the direct method in both two and three dimensions.

1.2 NAMMU - A groundwater flow package

NAMMU is an acronym for Numerical Assessment Method for Migration Underground. It was developed over a number of years by the Theoretical Physics Division at Harwell AEA Technology. Specific details on the NAMMU package are available in [48].

Principally NAMMU models (a) groundwater flow in a saturated medium, (b) coupled groundwater flow and heat transport, (c) radionuclide transport, (d) groundwater flow in the undersaturated zone, (e) radionuclide transport in the undersaturated zone, (f) coupled groundwater flow and heat transport in the undersaturated zone, and (g) coupled flow and solute transport with the fluid density strongly dependent upon concentration. The basic conceptual models underlying all these processes are flow and transport in a porous medium, which is primarily governed by Darcy's law, an empirical law which relates fluid pressure to fluid velocity, see (1.2.2). This project is concerned with solutions of the first of these, namely groundwater flow in a saturated medium.

The NAMMU package uses the finite element method to discretise for all the processes mentioned above. There are a number of reasons for this choice, see Johnson[56, p11] and also comments in [48]. In particular, compared to finite differences, the finite element methods allow easier modelling of complicated geometry, such as the complex geological structures seen in real applications. In addition general boundary conditions can be handled relatively easily. These are clearly important aspects when dealing with a package designed to handle a number of different pdes and user specified geometries.

Input to the NAMMU package is given by a sequence of simple commands, specifying the geometry and grid to use, the problem to be solved and the types of output that are desired.

1.2.1 The NAMMU code

It is important to realise that NAMMU is a very large package, incorporating many hundreds of subroutines with well over a million lines of fortran code. Built up over a large number of years, no one person is exactly sure what every bit of the code does. In the past, in order to minimise the size of the source code, most of the comment lines have been removed! Documentation is also not available at this time, although work on this is in progress. Many restrictions exist on data types, storage and the overall structure of any additional code. As in any large package a number of undocumented "features" still exist, these can cause unwanted side affects to certain test problems and test code.

In total these features mean that even the most trivial modification of the original code can be very challenging. This is particularly so when attempting to make the code do something that it was never intended to do, as was the case in the two and three grid methods of chapters 2 and 3.

1.2.2 A groundwater flow model

Steady state groundwater flow in a saturated porous medium is modelled in terms of pressure, p, and Darcy velocity $\mathbf{q}$, using the following equations and assuming constant fluid density,

$$\nabla.\mathbf{q} = 0 \quad \text{(Continuity equation)} \tag{1.2.1}$$

and

$$\mathbf{q} = -\frac{k}{\mu}(\nabla p) \quad \text{(Darcy's law)}, \tag{1.2.2}$$

together with a mixture of Dirichlet and Neumann boundary conditions. The coefficient k represents the rock permeability, and μ the fluid viscosity, which is assumed constant over the entire region. The two equations (1.2.1) and (1.2.2) may be combined into a single second-order differential equation for p,

$$-\nabla.(k\nabla p) = 0, \tag{1.2.3}$$

which is called the pressure equation.

Clearly if k were constant the problem would reduce to the classical Laplace's equation, for which there is considerable work on solution techniques. However, for realistic geological situations the permeability k, is position dependent, and can vary considerably over the entire region. This is one of the features that provides the interest and difficulties in this thesis. The variation in k will be discussed in more detail in §1.3.

1.2.3 A finite element model

A standard Galerkin formulation of the finite-element method is used to solve equation (1.2.3). The NAMMU package allows use of many different types of basis function, but principally, for the problems we consider, piecewise biquadratic basis functions on 9 node isoparametric quadrilateral elements (see figure 1-7) are used for the 2D discretisation. In the 3D case quadratic basis functions on 27 node (see figure 1-8) isoparametric cuboid elements are used.

We first formulate the problem as: find p such that

$$-\nabla.(k\nabla p) = 0 \text{ on } \Omega \tag{1.2.4}$$

with

$$p = h \text{ on } \partial\Omega_0 \tag{1.2.5}$$

$$\nabla p.\mathbf{n} = g \text{ on } \partial\Omega_1. \tag{1.2.6}$$

Here $\partial\Omega_0 + \partial\Omega_1 = \partial\Omega$, and $\mathbf{n}$ denotes the outward unit normal. Equations (1.2.5) and (1.2.6) represent the Dirichlet and Neumann boundary conditions respectively.

Now let the space U be the Hilbert space $H^1(\Omega)$, and let V be the subspace of U such that $V = \{v \in H^1 : v = 0 \text{ on } \Omega_0\}$. Then a variational formulation of the problem (1.2.4 - 1.2.6) is given by: find $p \in U$ such that

$$(p - h) \in V \tag{1.2.7}$$

and

$$a(p, v) = f(v) \ \forall v \in V, \tag{1.2.8}$$

where

$$a(u, v) = \int_\Omega k\nabla u.\nabla v \tag{1.2.9}$$

$$f(v) = \int_{\partial\Omega_1} kvg. \tag{1.2.10}$$

A solution p of (1.2.4 - 1.2.6) is a solution of (1.2.8 - 1.2.10) since by multiplying (1.2.4)

by $v \in V$, integrating and using Green's formula[28, p14] we obtain

$$\int_{\Omega} k\nabla p.\nabla v - \int_{\partial\Omega_1} vk\nabla p.\mathbf{n} = 0.$$

A solution of the variational formulation (1.2.8 - 1.2.10) is then a weak solution of (1.2.4 - 1.2.6). It is not immediately clear that a weak solution p of (1.2.8 - 1.2.10) is also a classical solution of (1.2.4 - 1.2.6) since this requires p to be sufficiently regular that $\nabla.(k\nabla p)$ is defined in a classical sense, which in turn requires k to be sufficiently regular.

Existence and uniqueness of a solution of (1.2.8 - 1.2.10) follow from an application of the Lax-Milgram lemma given by [28, Theorem 1.1.3]

Theorem 1.2.1 (Lax-Milgram Lemma) *Let V be a Hilbert space, let $a(\cdot,\cdot) : V \times V \rightarrow \mathbb{R}$ be a continuous V-elliptic bilinear form, in the sense that*

$$\exists\ \alpha > 0,\ \forall v \in V,\ \text{such that}\ \alpha \parallel v \parallel^2 \leq a(v,v).$$

Further let $f : V \rightarrow \mathbb{R}$ be a continuous linear form.

Then the abstract variational problem: Find an element u such that

$$u \in V \ \text{and}\ \forall v \in V,\ a(u,v) = f(v),$$

has one and only one solution.

Proof See [28, pp8-9]. □

A more complete discussion of the existence and uniqueness of the solution to this problem can be found by examining the example problem in Ciarlet [28, p20], which discusses a more general case of the problem we consider here.

From the variational formulation (1.2.8 - 1.2.10) the finite element method proceeds by choosing a set of element basis functions Φ_j in some finite dimensional finite element

space, and seeking a solution of the form

$$p = \sum_{j=1}^{N} x_j \Phi_j, \tag{1.2.11}$$

where the x_j represent the unknown nodal values of p. Using (1.2.8 - 1.2.10) we then obtain the linear system

$$A\mathbf{x} = \mathbf{b}, \tag{1.2.12}$$

where A is the stiffness matrix and $\mathbf{b}$ depends on the boundary conditions. For more details on the finite element method see Johnson[56] and Ciarlet[28].

From the weak form (1.2.8 - 1.2.10) it is clear that the stiffness matrix, A, is symmetric, since the bilinear form is symmetric. Since $k > 0$ then the bilinear form is also positive definite and hence the resulting stiffness matrix is also positive definite. It is well known that a positive definite matrix is non-singular[44, §2.10], and so the linear system (1.2.12) has a unique solution.

We also note that A is *sparse*, by which we mean that only comparatively few entries in A are non-zero. This follows from the fact that the basis functions are chosen to be only non-zero on a small number of intervals in the domain, and hence many of the integrals (1.2.9) are zero,

Conditioning of the stiffness matrix

We conclude this section by briefly looking at the condition number of the finite element stiffness matrix arising from discretisations of the (1.2.3). As will be seen later this quantity is very important in the discussion of the convergence rate of iterative methods. Typically large condition numbers imply poor convergence properties of iterative methods.

Recall that the condition number, κ, of a symmetric positive definite matrix A is given by the ratio of maximum and minimum eigenvalues of A, i.e.

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}. \tag{1.2.13}$$

In Johnson[56] the condition number of the problem of the form (1.2.3) with constant k is analysed, for a two dimensional problem. It is a relatively simple matter to adapt this approach to include the varying coefficient k. We note that the bilinear form (1.2.9) is V - elliptic (this being one of the conditions in the Lax-Milgram lemma), ie

$$a(v,v) \geq \alpha \parallel v \parallel_V^2 \qquad \forall v \in V, \ \alpha > 0. \tag{1.2.14}$$

For varying k it is then sufficient to assume that $\alpha = C_1 k_{\min}$, where C_1 is a positive constant.

Under certain regularity assumptions on the grid size, the following result allows us to estimate the maximum and minimum eigenvalues. Here the grid is assumed quasi-uniform in the sense that all the elements are roughly the same size, a more precise definition of this is given in Johnson [56, pp141-142]. The grid size parameter h relates to the average size (area or volume) of each element. For example in 1D a truly uniform grid, over $[0,1]$ with Dirichlet boundary conditions, would have $h = 1/N$, where N is the number of nodes.

Lemma 1.2.2 *(cf. Lemma 7.3 of [56]) There are constants C_2 and C_3 independent of the grid size h, such that for all $v = \sum_i x_i \Phi_i$*

$$C_2 h^2 |\mathbf{x}|^2 \ \leq \ \parallel v \parallel^2 \leq C_3 h^2 |\mathbf{x}|^2 \tag{1.2.15}$$

and

$$a(v,v) \ \equiv \ \int_\Omega k |\nabla v|^2 dx \leq k_{\max} C_3 h^{-2} \parallel v \parallel^2 . \tag{1.2.16}$$

where $\parallel v \parallel = \parallel v \parallel_{L^2(\Omega)}$.

Hence from (1.2.15) and (1.2.16)

$$\frac{\mathbf{x}^T A \mathbf{x}}{|\mathbf{x}|^2} = \frac{a(v,v)}{|\mathbf{x}|^2} \leq k_{\max} C_4 h^{-2} \frac{\parallel v \parallel^2}{|\mathbf{x}|^2} \leq C_5 k_{max} \quad \forall \mathbf{x} \in \mathbb{R}^N. \tag{1.2.17}$$

From (1.2.14) and (1.2.15) we have, since trivially $\parallel v \parallel_V \geq \parallel v \parallel$,

$$\frac{\mathbf{x}^T A \mathbf{x}}{|\mathbf{x}|^2} = \frac{a(v,v)}{|\mathbf{x}|^2} \geq \ \alpha \ k_{\min} \frac{\parallel v \parallel^2}{|\mathbf{x}|} \geq C_6 \alpha k_{\min} h^2 \quad \forall \mathbf{x} \in \mathbb{R}^N. \tag{1.2.18}$$

Hence equations (1.2.17) and (1.2.18) show that $\kappa(A) = \frac{\lambda_{max}}{\lambda_{min}} \leq C\frac{k_{max}}{k_{min}}h^{-2}$, so that the condition number increases as k_{max}/k_{min} increases or as h decreases. Note that an analogous result holds in three dimensions.

For problems considered here, the ratio $k_{\max}/k_{\min}$ will be often large, however, as will be seen in chapter 2, by a diagonal scaling of the stiffness matrix A the effect of this ratio can be almost removed.

In problems that we will consider the grid is rarely uniform, however, by setting $h = \min_i(h_i)$, where h_i relates the size of each element, the above analysis still holds. The result of this is that for grids of the type seen in figure 1-1 the condition number of the stiffness matrix can be very high.

Finally we briefly comment on the discretisation error in the finite element method when using quadratic basis functions. From Johnson[56, §4.3] we have an estimate of the following form

$$\| u - u_h \|_{L^2(\Omega)} \leq Ch^{r+1}|u|_{H^{r+1}(\Omega)}, \tag{1.2.19}$$

where u is the exact solution, u_h is the approximate solution and $r \geq 1$ is the degree of the polynomial used in the basis function. Hence for the quadratic basis function case, i.e. $r = 2$, it is clear that in the L^2-norm the discretisation error is $O(h^3)$.

1.3 Discretisations of example problems

Apart from some calculations on two simple model problems, the results presented in this thesis arise from a physically realistic problem in 2D and 3D. The 2D problem models a geological region from a location in the UK and the 3D problem is taken from the dataset used to model the groundwater flow and saline transport at the Gorleben site in Germany[22]. In the 3D problem $N \approx 172,000$, and in the 2D case $N \approx 60,000$. In the 2D problem piecewise biquadratic basis functions on 9 node isoparametric quadrilateral elements are used for the discretisation, see figure 1-7. In the 3D case quadratic basic functions on 27 node cuboid elements are used, see figure 1-8.

1.3.1 Grids and rock types

As mentioned in §1.2, we shall assume that the rock permeability coefficient k in the pde (1.2.3) is always piecewise constant, with large variations in values in realistic rock formations. In addition, the creation of an accurate model of a typical underground rock structure often leads to highly unstructured grids with a range of element sizes. Figure 1-1 demonstrates a typical grid for the 2D problem, with approximately $N = 3500$. Figure 1-2 shows the regions in this grid with different rock permeabilities. Table 1.1 gives the actual values of rock permeabilities for this problem, in the x and y directions.

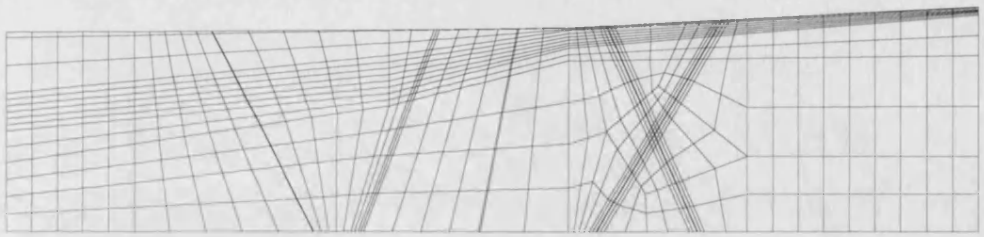

Figure 1-1: *Sample grid for realistic 2D problems.*

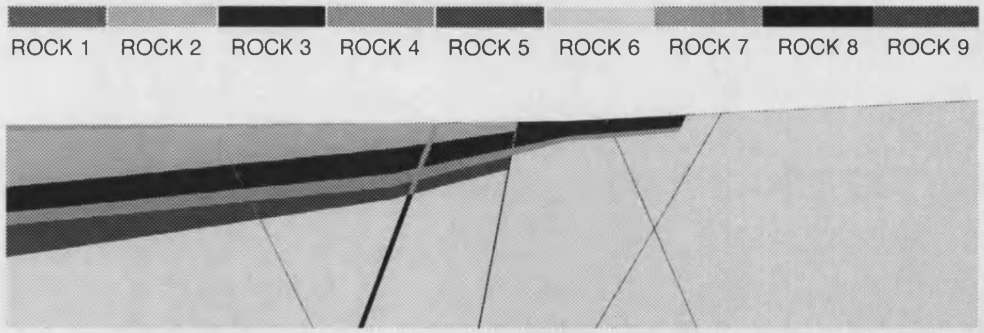

Figure 1-2: *Permeability regions corresponding to the 2D grid in figure 1-1, actual values are given in table 1.1.*

For the 3D problem typical grids are shown in figures 1-3 - 1-5, these figures also show the different rock regions. For this problem the k values are in the range 10^{-5} to 10^{-10}.

Region Type	k_x	k_y
1	2.5E-12	1.0E-14
2	3.0E-13	3.5E-14
3	2.5E-15	2.25E-15
4	1.75E-15	1.75E-15
5	4.0E-15	4.0E-15
6	1.0E-17	1.0E-17
7	1.5E-12	1.5E-12
8	1.0E-14	1.0E-14
9	5.0E-15	4.0E-18

Table 1.1: *Permeability coefficients corresponding to the rock regions in figure 1-2*

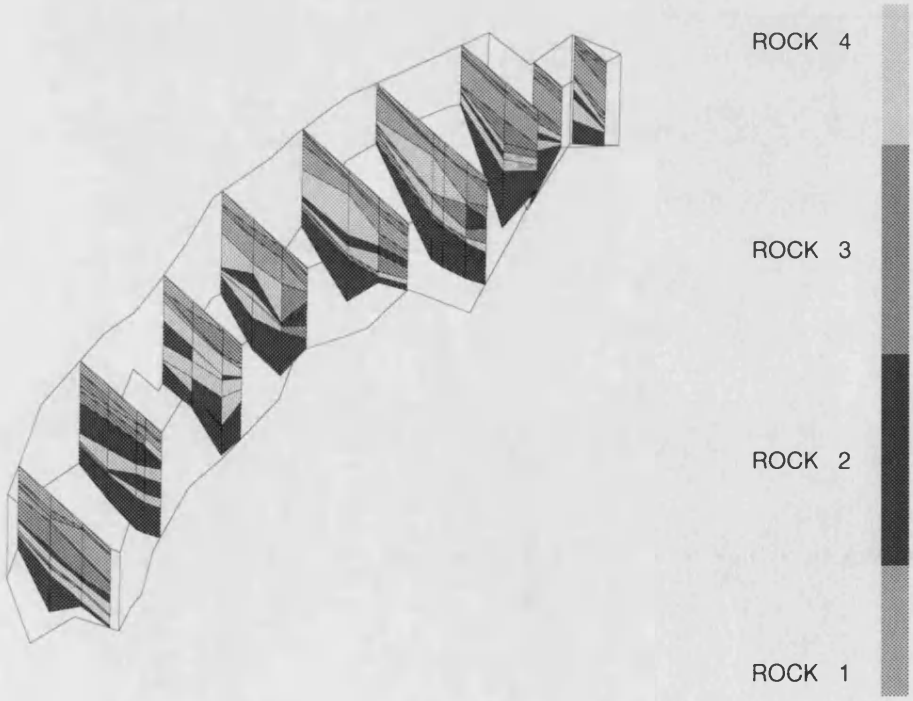

Figure 1-3: *3D grid cross section.*

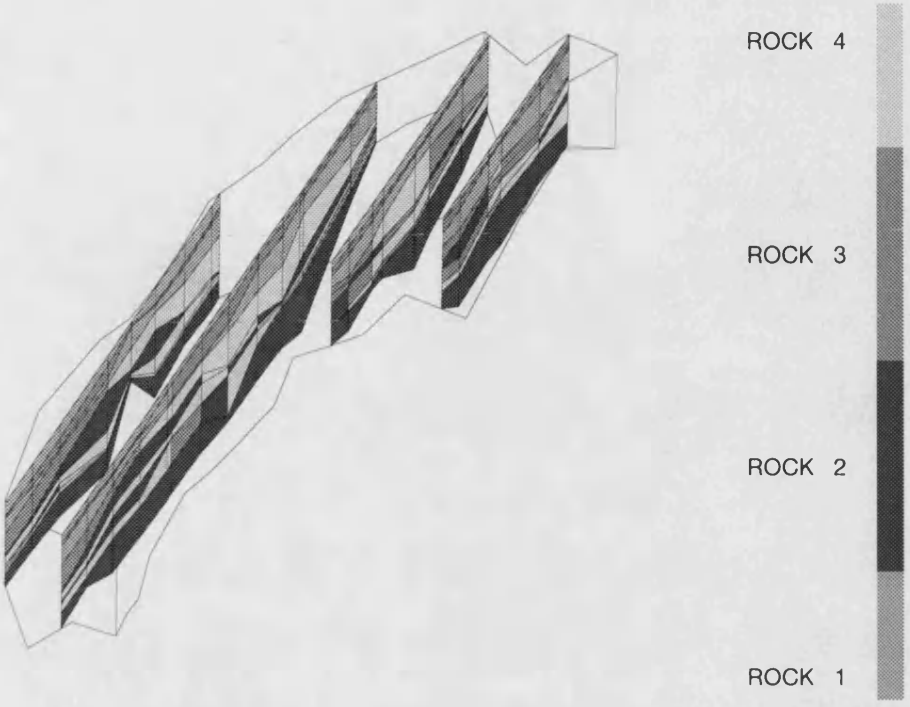

Figure 1-4: *3D grid cross section.*

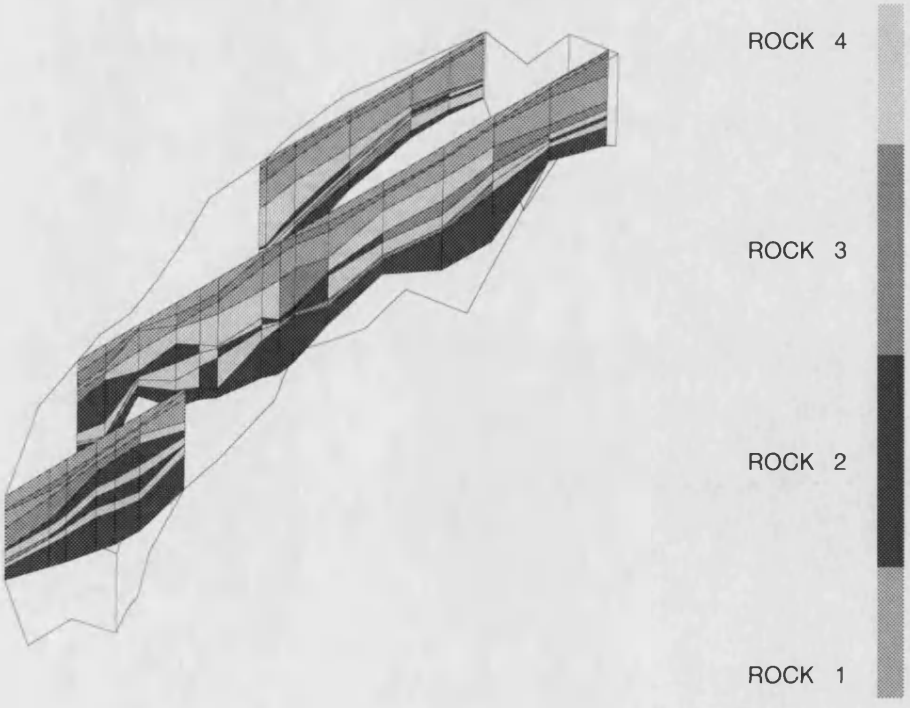

Figure 1-5: *3D grid cross section.*

However, we shall see in chapter 2 that diagonal preconditioning can mitigate the effect of discontinuities in k, as has been stated before, see, for example [74] and [40]. Nevertheless, even after diagonal scaling the resulting system is still very badly conditioned, due mainly to the highly irregular grid. Hence methods which perform well on more straightforward problems, like conjugate gradients or polynomial preconditioned conjugate gradients, perform extremely poorly for this problem. This poor performance will be discussed in more detail in §1.6.

1.3.2 Finite elements

Use of quadratic basis functions does have some drawbacks from the point of view of iterative methods. In particular, unlike the linear basis functions case, it is no longer guaranteed that the resulting stiffness matrix is "diagonally dominant" or an "M-matrix". Generally this means that it is hard to produce quantatative convergence statements for iterative methods. For example the standard results relating convergence of Jacobi, Gauss-Seidel and SOR, which depend on the M-matrix property, see [44, §6.4-§6.6], cannot be applied. In addition incomplete factorisations (see chapter 5) cannot be guaranteed to exist and be numerically stable for problems that have neither of these properties.

We define diagonal dominance by

Definition 1.1 *An $N \times N$ matrix A is diagonally dominant if*

$$|a_{ii}| \geq \sum_{j=1}^{N} |a_{ij}|, \quad i = 1, ..., N.$$

Whilst an M-matrix is given by

Definition 1.2 *A matrix A is an M-matrix if*

(1) A is non-singular.

(2) $a_{ij} \leq 0, \quad i \neq j.$

(3) $a_{ij}^{-1} \geq 0, \quad \forall i, j.$

To see that neither of these properties hold in our case we merely need to examine an element stiffness matrix for a square 2D 9 node region, (of the type shown later in figure 1-7). From the finite element formulation this involves calculating the 9 biquadratic basis functions, Φ_i, and computing

$$a_{ij} = \int\int \nabla\Phi_i . \nabla\Phi_j, \quad \text{for } i, j = 1, 9.$$

This results in the 9×9 matrix,

$$\begin{bmatrix}
\frac{28}{45} & -\frac{1}{30} & -\frac{1}{45} & -\frac{1}{30} & -\frac{1}{5} & \frac{1}{9} & \frac{1}{9} & -\frac{1}{5} & -\frac{16}{45} \\
-\frac{1}{30} & \frac{28}{45} & -\frac{1}{30} & -\frac{1}{45} & -\frac{1}{5} & -\frac{1}{5} & \frac{1}{9} & \frac{1}{9} & -\frac{16}{45} \\
-\frac{1}{45} & -\frac{1}{30} & \frac{28}{45} & -\frac{1}{30} & \frac{1}{9} & -\frac{1}{5} & -\frac{1}{5} & \frac{1}{9} & -\frac{16}{45} \\
-\frac{1}{30} & -\frac{1}{45} & -\frac{1}{30} & \frac{28}{45} & \frac{1}{9} & \frac{1}{9} & -\frac{1}{5} & -\frac{1}{5} & -\frac{16}{45} \\
-\frac{1}{5} & -\frac{1}{5} & \frac{1}{9} & \frac{1}{9} & \frac{88}{45} & -\frac{16}{45} & 0 & -\frac{16}{45} & -\frac{16}{15} \\
\frac{1}{9} & -\frac{1}{5} & -\frac{1}{5} & \frac{1}{9} & -\frac{16}{45} & \frac{88}{45} & -\frac{16}{45} & 0 & -\frac{16}{15} \\
\frac{1}{9} & \frac{1}{9} & -\frac{1}{5} & -\frac{1}{5} & 0 & -\frac{16}{45} & \frac{88}{45} & -\frac{16}{45} & -\frac{16}{15} \\
-\frac{1}{5} & \frac{1}{9} & \frac{1}{9} & -\frac{1}{5} & -\frac{16}{45} & 0 & -\frac{16}{45} & \frac{88}{45} & -\frac{16}{15} \\
-\frac{16}{45} & -\frac{16}{45} & -\frac{16}{45} & -\frac{16}{45} & -\frac{16}{15} & -\frac{16}{15} & -\frac{16}{15} & -\frac{16}{15} & \frac{256}{45}
\end{bmatrix}. \tag{1.3.1}$$

Clearly, this matrix is not diagonally dominant, and the presence of positive off-diagonal entries implies that it is not an M-matrix. These properties are extremely likely to be carried forward to the full stiffness matrix, since it is merely a summation of matrices of the form in (1.3.1).

1.4 A direct solution technique

For problems involving finite elements, one of the standard direct solution techniques is the *frontal method*. This method is essentially Gaussian elimination, but is performed in such a way as to avoid assembling the stiffness matrix. In this section we give a brief

overview of the frontal method. For more detailed accounts on the frontal method and its variants see Johnson[56, §6.5] and Duff et al[32, §10.5].

1.4.1 The Frontal method

For finite elements, the stiffness matrix can be written in terms of its element stiffness matrices as the sum

$$A = \sum_l A^{[l]}, \tag{1.4.1}$$

where each $A^{[l]}$ has entries only in the principal submatrix corresponding to the freedoms in element l and represents the contributions from this element. In a real application these submatrices would be stored in a packed form as small matrices, see (1.3.1), with an additional vector to specify the freedoms present. The formation of the sum in (1.4.1) is called *assembly* and involves operations of the form

$$a_{ij} = a_{ij} + a_{ij}^{[l]}. \tag{1.4.2}$$

An entry in the matrix A is denoted *fully summed* when all contributions of the form (1.4.2) have been summed.

The basic operation of Gaussian elimination performed on the matrix A, is given by

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) a_{kj}^{(k)}. \tag{1.4.3}$$

The key point, first noted by Irons[55], is that this operation can be performed before all the assemblies in (1.4.2) are finished, provided that the three terms a_{ik}, a_{kk}, a_{kj} in (1.4.3) are fully summed. In essence this means that the kth freedom can be eliminated as soon as all the entries in the kth row and kth column are fully summed, which will happen as soon as the last element matrix $A^{[l]}$ referring to this freedom has been processed. Hence the elimination process can be confined purely to the submatrix of rows and columns corresponding to freedoms that have not yet been eliminated, but are currently involved in an elimination process, i.e. one or more of their elements has been assembled. These freedoms in this submatrix are referred to as the *active* freedoms. At any one time the working is then done on a full matrix, the size of which increases as new freedoms are

introduced or decreases as freedoms are eliminated. If the elements are ordered in some systematic fashion from one end of the domain to another, the active freedoms form a *front* that moves along systematically through the ordering. Hence the full matrix where all the working is done is called the *frontal matrix* and the resulting technique is called the *frontal method*. In figure 1-6 this idea of the frontal matrix, denoted B_k, is demonstrated. Note that this frontal matrix will generally be far smaller than the original matrix, and so can be reasonably stored in a full form.

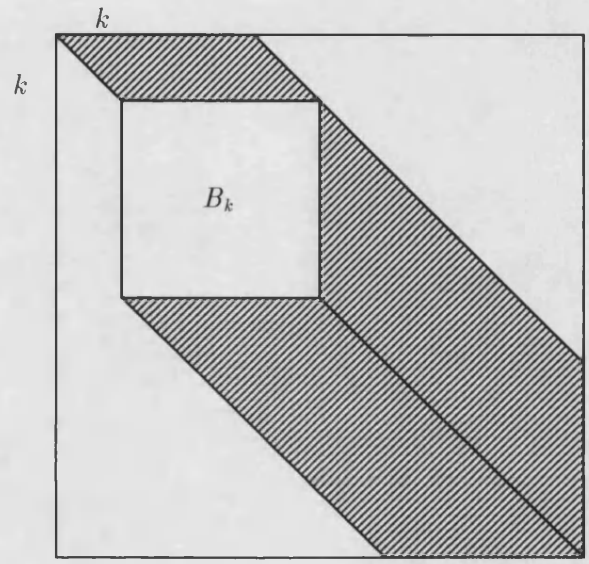

Figure 1-6: *Partially eliminated banded matrix, showing the frontal matrix, B_k, and the $k \times k$ eliminated region.*

Clearly ordering of the nodes plays a major role in determining the frontwidth, ordering algorithms such as the "reverse Cuthill-Mckee" algorithm are often used to minimise the frontwidth, see Duff et al[32] for more discussion on this.

1.4.2 Work and storage considerations

The amount of work, or arithmetic operations, and storage required for this method is closely related to the size of the frontal matrix required, this maximum size is called the *frontwidth*. The size of the front required is controlled by the bandwidth of the matrix. For finite elements using meshes with approximately n elements in each spatial direction, this is generally $O(n^{d-1})$, where d is the dimension of the problem. Table 1.2 shows the approximate work and storage requirements for the frontal method for

2D and 3D finite element discretisations. Note that for a 3D problem full Gaussian elimination would require $O(n^9)$ operations to factorise the stiffness matrix.

	2D	3D
Frontwidth	n	n^2
Storage	n^2	n^4
Work	n^4	n^7

Table 1.2: *Work and storage counts for the frontal method on finite element discretisations. Here n is the number of elements in each spatial direction.*

It is clear that for large problems the cost of computing the solution rapidly becomes prohibitively large, particularly in 3D. Indeed to put this into perspective consider the following "thought experiment". Working in 3D, we start with a discretisation that requires 30 seconds of computer time to solve, for some computer. Doubling the mesh size in each direction gives a problem requiring 1 hour to solve, doubling again requires over 5 days of computer time.

Now compare this with a typical iterative method, say conjugate gradients (see §1.6). This method requires one matrix-vector multiplication per iteration, the cost of which is $O(N)$, where N is the size of the matrix (i.e. n^2 in 2D and n^3 in 3D). In addition we can expect the method to converge in $O(n)$ steps, hence the method has work counts of approximately $O(n^3)$ in 2D and $O(n^4)$ in 3D. This means that if conjugate gradients required the same CPU time of 30 seconds on the small problem then the "5 day problem" could be solved iteratively in less than 1 hour, a very significant saving on a machine where you might be billed for each minute of CPU time. A further issue is that for 3D problems the storage requirement of the frontal method is generally larger than that of an iterative method, by a factor of n, this follows from the fact that the frontal method requires full storage of the front matrix, of size $n^2 \times n^2$, whereas an iterative method like conjugate gradients requires only storage of the stiffness matrix, which, for finite elements, requires $O(n^3)$.

1.4.3 Harwell frontal code

The frontal code used in the NAMMU package, up until June 1994, is called MA32. An updated version called MA42 has been recently implemented, although this version is based on exactly the same method, the code now implements a number of improved features, including wide use of BLAS routines.

Both of these frontal codes use the methodology developed by Duff et al, the MA32 version is the one discussed in [32]. Developed over a number of years these frontal codes form a package that has been highly optimised and runs very fast on the Cray YMP. The MA32 code, and the rest of the routines from the Harwell subroutine library are widely believed to be the best of their kind anywhere in the world. Indeed timings and performance from these routines are often used as benchmarks.

As discussed in the previous section, iterative methods generally have a significant advantage for large problems, especially in 3D. Nonetheless, it was not expected at the outset of the project that the frontal solver could be beaten for 2D problems, without considerable optimising of any iterative code. Results in chapter 2 are therefore very encouraging.

1.5 Storage of the stiffness matrix

Any iterative method for the solution of (1.2.12) will require matrix-vector multiplication operations. Hence before discussing iterative methods it will first be necessary to decide on the storage format of the finite element stiffness matrix, A. Clearly the storage method will directly affect the implementation of the matrix-vector multiplication routine.

In the existing NAMMU code this matrix is never fully assembled from its element stiffness matrices (see 1.3.1) as this is not necessary for the frontal solver technique discussed in §1.4.

For the iterative schemes requiring the matrix-vector operation there are two alternatives:

- Fully assemble the element matrices into a sparse matrix storage scheme
- Retain the element matrix form.

Each option has its own merits: clearly assembly and representation by a sparse scheme will require extra work, but manipulations of the matrix, or parts of the matrix, are possible. By retaining the element format only the matrix-vector multiplication operation is possible, so iterative methods are limited to the so called "matrix-free" methods.

Note that, in the element scheme, computing the full matrix-vector multiplication is achieved by computing a series of small matrix-vector multiplications and summing the result. With only a careful ordering of the loops it is possible to, in theory, achieve near maximum machine performance on the Cray YMP. By this we mean that the full capability of the computer architecture can be realised, and computational work can be performed at nearly the maximum rate that the machine can achieve. However, in practice this was never actually managed since the lowest level loop needed to be implemented in Cray assembler, as the fortran compiler was not advanced enough to recognise the full optimisation potential of the code.

1.5.1 A storage example

In order to compare and contrast the two storage schemes discussed above we consider a simple example, using the most common type of 2D and 3D element used in NAMMU models. For the sake of simplicity the effects of Dirichlet boundary conditions on the problems will be neglected. The notation n will denote the number of elements in any one spatial direction. The system considered will be on a grid with $n \times n$ elements, or in a 3D case $n \times n \times n$ elements.

For the sparse matrix storage scheme it is generally required that each stored real value requires an associated index pointer, which is usually an integer value. Although in practice slightly more storage than this is required, nonetheless this gives a good indication of the best sparse storage method.

Also note that on Cray architecture all reals are double precision and integers take

the *same* amount of storage space as the reals. Clearly this will be an important consideration when discussing the sparse storage schemes.

It is important to realise that the sparsity of the fully assembled stiffness matrix is highly dependent on the type of element used and the degree of basis function. In the following discussion we will consider quadrilateral elements and quadratic basis functions. A basis function is associated with an element and is defined to be zero on any nodes lying outside the element. Hence non-zero entries in the stiffness matrix only arise in the case where a node is connected, via the basis function, to another node on elements where the basis function is not defined as zero. For example a node on the edge of an element is potentially connected to all the nodes in that element, plus all the nodes in the adjacent elements that share this node. The use of quadratic basis functions mean that nodes up to "two nodes away" are potentially connected in this sense.

We also remark that whilst this discussion centers on strictly square and cuboid elements, the NAMMU code uses isoparametric transformations to map these into other rectangular or cuboidal shapes. More details on these transformations are given in Johnson [56, §12].

The 2D case

The 2D element we consider with its associated node points is given in figure 1-7.

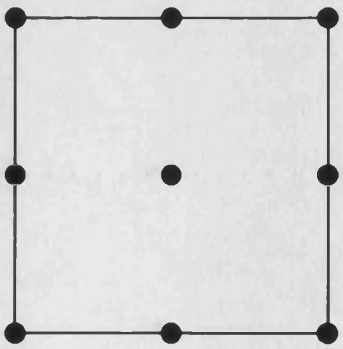

Figure 1-7: *Node points of the 2D 9 node quadrilateral.*

Consider first the number of non-zero entries in the fully assembled sparse storage scheme associated with each element. For each element we need to consider (a) the corner nodes, (b) the edge nodes, and (c) the centre nodes. Clearly each of these types of nodes appears in a different number of elements: 4 for the corners, 2 for the edges, and 1 for the centres. Hence a corner node is "connected" to 5×5 other nodes, but this contribution is "shared" by 4 elements. A similar consideration can be constructed for the edge and centre nodes.

In total we have the following contributions from each element:

$$\begin{array}{lll} \text{corners} & 4 \times 25 \times \frac{1}{4} & = 25 \\ \text{cell edge} & 4 \times 15 \times \frac{1}{2} & = 30 \\ \text{cell centre} & 1 \times 9 & = 9 \\ & & \overline{64} \end{array}$$

Each entry requires one real and one integer and there will be n^2 elements. Hence the total required storage is $128n^2$.

For the element storage case we require a 9×9 matrix for each element plus 9 integer pointers, hence the total storage will be $90n^2$.

We can see that in the 2D quadrilateral case we would need approximately 42% more storage to assemble the full matrix in a sparse form. Note that this analysis is only for a machine like the Cray where integers/reals take the same storage space. Nonetheless, even on machines where the integers take half the storage than the reals (or double precisions) the element form is still likely to win, or at least be comparable. In the above case we would then be comparing $90n^2$ for the element case with $92n^2$ for the sparse case.

The 3D case

The 3D element we are considering together with its associated node points in given in figure 1-8.

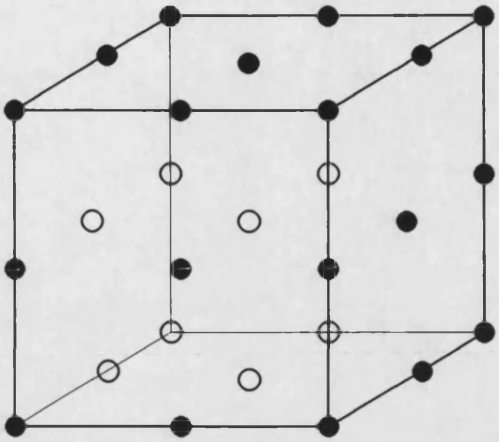

Figure 1-8: *Node points of the 3D 27 nodes cuboid.*

As for the 2D case, we consider corner nodes, edge nodes, centre nodes, and also for this case, face centre nodes. Here a corner node is "connected" to $5 \times 5 \times 5$ other nodes, but this contribution is shared by 8 other elements.

In total for the sparse storage case per element we have the following contributions:

corners	$8 \times 125 \times \frac{1}{8}$	$= 125$
cell edge	$12 \times 75 \times \frac{1}{4}$	$= 225$
cell face centre	$6 \times 45 \times \frac{1}{2}$	$= 135$
cell middle	1×27	$= 27$
		$\overline{512}$

As before each entry requires one real and one integer and there will be n^3 elements. Hence the total required storage is $1024n^3$.

The corresponding element storage case requires a 27×27 matrix plus 27 integer pointers per element, hence the total storage is $756n^3$.

In a similar fashion to the two dimensional case we again need more storage for the sparse approach than in the element storage approach, approximately 35% in this case.

1.5.2 Preferred storage method

There would appear to be no performance loss when using element matrices rather than a sparse matrix format. Although results in §5.3 indicate that a sparse scheme is potentially detrimental to the performance of the matrix-vector multiplications, this is mainly due to the difficulty in produced effective code when working with a sparse scheme. More effective implementations of sparse techniques have been shown to give performance that is comparable with the element storage scheme. Finally note that the speed of the matrix-vector multiplications will have a marked effect on the overall solution time of the linear solver.

1.6 A first iterative method

Given the restriction of using a "matrix-free" method (see §1.5), even a relatively quick review of iterative methods immediately points to the conjugate gradient method of Hestenes and Stiefel[49] as an "obvious choice". Although originally devised in 1952, this method was not originally widely considered for use as an iterative solver until around 1971, when Reid[75] re-introduced the method and advocated its use for sparse matrices. Prior to this paper the numerical analysis community was not satisfied with either the understanding of this method or its speed (preconditioners were not widely known at this time). Since then it has become a very popular method in a wide range of applications and many variants and modifications on the original idea have been produced, see Ashby et al[10, 6].

More complete discussions on the conjugate gradient method are given in Golub and van Loan [38, §10], Hackbusch[44, §9], and Johnson[56, §7.3]. A detailed history of the development of the method is given in Golub and O'Leary[37].

In this section we shall introduce some background to the method, discuss its convergence properties and briefly give some results for a test problem. Throughout the section the matrix will be assumed to be symmetric positive definite.

1.6.1 General theory of iterative methods

A general iterative method for the solution of (1.2.12), with an initial starting guess $\mathbf{x}_0$, produces a sequence of iterates $\mathbf{x}_k$ given by

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \sum_{j=0}^{k-1} \eta_{kj} \mathbf{r}_j, \tag{1.6.1}$$

where $\eta_{kj} \in \mathbb{R}$ are suitably chosen for convergence and we define the residual, $\mathbf{r}_j$, by

$$\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j. \tag{1.6.2}$$

The error in solution at the k^{th} step, $\mathbf{e}_k$, is defined by

$$\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k. \tag{1.6.3}$$

The iterative process (1.6.1) may be called a *polynomial method*, by which it is meant that the error (1.6.3) can be written in the form

$$\mathbf{e}_k = p_k(A)\mathbf{e}_0, \tag{1.6.4}$$

where $p_k(A)$ is a polynomial of degree k, and hence from (1.6.4) we may also write

$$\mathbf{r}_k = p_k(A)\mathbf{r}_0,$$

since $A\mathbf{e}_k = A\mathbf{x} - A\mathbf{x}_k = \mathbf{b} - A\mathbf{x}_k = \mathbf{r}_k$. This polynomial is called the residual or error polynomial. Note that this polynomial must satisfy the condition $p_k(0) = 1$ since if $A = 0$ then clearly $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k = \mathbf{b}$ $\forall k$. Hence if $\mathbf{r}_k = p_k(0)\mathbf{r}_0$ then $p_k(0)$=1.

For a method that converges well, $\mathbf{e}_k$ should be as "small" as possible, in some sense. The most likely choice being that $\| \mathbf{e}_k \|$ is small for some chosen norm. However, it is clear that for large problems the recursion (1.6.1) would be impractical, as it would require storage of all previous residuals. A more practical version could be given by

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k,$$

where $\mathbf{d}_k$, which we call a direction vector, is a linear combination of a few previous

residuals $\{\mathbf{r}_{k-1},...,\mathbf{r}_{k-s}\}$ and a few previous direction vectors $\{\mathbf{d}_{k-1},...,\mathbf{d}_{k-t}\}$, for some integers s and t. The value $\alpha_k \in \mathbb{R}$ is suitably chosen to satisfy some minimisation property. Since the method is required to converge monotonically, then clearly we also require

$$\| \mathbf{e}_k \| \leq \| \mathbf{e}_{k-1} \|$$

i.e.

$$\| \mathbf{x} - \mathbf{x}_{k-1} - \alpha_k \mathbf{d}_k \| \leq \| \mathbf{x} - \mathbf{x}_{k-1} \| .$$

1.6.2 Conjugate Gradients (CG)

From the definition of $\mathbf{r}_k$ we can re-arrange (1.6.1) to show that it is equivalent to writing

$$\mathbf{x}_k = \mathbf{x}_0 + \mathbf{v}, \quad \mathbf{v} \in \text{span } \{\mathbf{r}_0, A\mathbf{r}_0, .., A^{k-1}\mathbf{r}_0\}. \tag{1.6.5}$$

We call span$\{\mathbf{r}_0, A\mathbf{r}_0, .., A^{k-1}\mathbf{r}_0\}$ a *Krylov subspace* of dimension at most k, denoted by $\mathcal{K}_k(\mathbf{r}_0, A)$.

Consider an iterative method given by choosing $\mathbf{d}_k \in \mathcal{K}_k(\mathbf{r}_0, A)$ such that $\| \mathbf{e}_k \|$ is minimised at each step, for some norm. The CG method is then given by measuring the error in the following norm

$$\| \cdot \|_A = < A\cdot, \cdot >^{\frac{1}{2}} . \tag{1.6.6}$$

Since $\mathcal{K}_{k-1} \subset \mathcal{K}_k$ it follows that $\| \mathbf{e}_k \|_A \leq \| \mathbf{e}_{k-1} \|_A$, i.e. the error is monotonically non-increasing.

The minimisation of the error in this A-norm is achieved by requiring that $\mathbf{e}_k \perp_A \mathcal{K}_k$, i.e. all components of the error lying in $\mathcal{K}_k$ have been removed (cf. Johnson [56, pp133-134]) and thus

$$< A\mathbf{e}_k, \mathbf{y} >= 0 \ \ \forall \mathbf{y} \in \mathcal{K}_k(\mathbf{r}_0, A). \tag{1.6.7}$$

This is turn means $\mathbf{d}_k \in \mathcal{K}_k$ and $\mathbf{d}_k \perp_A \mathcal{K}_{k-1}$. So an A-orthogonal basis $\{\mathbf{p}_k\}_{j=1}^k$, for $\mathcal{K}_k$ is found. By construction $\mathbf{p}_k \in \mathcal{K}_k$ and $\mathbf{p}_k \perp_A \mathcal{K}_{k-1}$. Since $\mathbf{e}_k \perp_A \mathcal{K}_{k-1}$ it follows that

$\mathbf{d}_k = \alpha_k \mathbf{p}_k$, $\alpha_k \in \mathbb{R}$.

From the requirement that $\mathbf{e}_k \perp_A \mathcal{K}_k$ we determine α_k thus

$$\begin{aligned} 0 &= < A\mathbf{e}_k, \mathbf{p}_k > = < A(\mathbf{e}_{k-1} - \alpha_k \mathbf{p}_k), \mathbf{p}_k > \\ &= < A\mathbf{e}_{k-1}, \mathbf{p}_k > - \alpha_k < A\mathbf{p}_k, \mathbf{p}_k > \\ \text{and hence } \alpha_k &= \frac{< A\mathbf{e}_{k-1}, \mathbf{p}_k >}{< A\mathbf{p}_k, \mathbf{p}_k >} \\ &= \frac{< \mathbf{r}_{k-1}, \mathbf{p}_k >}{< A\mathbf{p}_k, \mathbf{p}_k >}. \end{aligned}$$

We next quote a result about the CG method from Golub and van Loan[38],

Theorem 1.6.1 *(Theorem 10.2.2) After k iterations of the CG algorithm we have*

$$\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k A\mathbf{p}_k \tag{1.6.8}$$

$$< \mathbf{p}_j, \mathbf{r}_k > = 0, \quad 1 \leq j \leq k \tag{1.6.9}$$

$$\begin{aligned} \text{span}\{\mathbf{p}_1, ..., \mathbf{p}_k\} &= \text{span}\{\mathbf{r}_0, ..., \mathbf{r}_{k-1}\} \\ &= \text{span}\{\mathbf{b}, A\mathbf{b}, ..., A^{k-1}\mathbf{b}\}. \end{aligned} \tag{1.6.10}$$

From (1.6.10) it is apparent that we can write

$$\begin{aligned} \mathbf{p}_{k+1} &= \mathbf{r}_k - \sum_{i=1}^{k} \beta_{k,i} \mathbf{p}_i \\ \text{with } \beta_{k,i} &= \frac{< A\mathbf{r}_k, \mathbf{p}_i >}{< A\mathbf{p}_i, \mathbf{p}_i >}. \end{aligned} \tag{1.6.11}$$

However, from (1.6.8) and (1.6.10) together with the fact that $\mathbf{p}_i \perp_A \mathbf{p}_j$, $i \neq j$, ie

$$< A\mathbf{p}_j, \mathbf{p}_i > = 0 \quad \forall i \neq j,$$

it is clear that $\beta_{k,i} = 0$, $\forall i < k$. Hence

$$\mathbf{p}_{k+1} = \mathbf{r}_k - \beta_{k,k} \mathbf{p}_k. \tag{1.6.12}$$

Writing β_k for $\beta_{k,k}$ and, for computational convenience, re-arranging the expressions for

α_k and β_k, see [38, pp522-523], gives the usual algorithm for CG.

Algorithm 1.1 [$\mathbf{x}$ =CG($A, \mathbf{b}, \mathbf{x}_0$)] From some initial guess $\mathbf{x}_0$, this algorithm applies CG until a specified convergence criterion is met.

$$
\begin{aligned}
\mathbf{r}_0 &= \mathbf{b} - A\mathbf{x}_0 \\
\mathbf{p}_1 &= \mathbf{r}_0 \\
\text{for } k &= 1, 2, \ldots \text{until converged} \\
\alpha_k &= \frac{< \mathbf{r}_{k-1}, \mathbf{r}_{k-1} >}{< A\mathbf{p}_k, \mathbf{p}_k >} \\
\mathbf{x}_k &= \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k \\
\mathbf{r}_k &= \mathbf{r}_{k-1} - \alpha_k A\mathbf{p}_k \\
\beta_k &= \frac{< \mathbf{r}_{k-1}, \mathbf{r}_{k-1} >}{< \mathbf{r}_k, \mathbf{r}_k >} \\
\mathbf{p}_{k+1} &= \mathbf{r}_k - \beta_k \mathbf{p}_k \\
\text{end}
\end{aligned}
$$

Note that from this implementation it is clear that only one matrix-vector multiplication and two inner products are needed per step of the algorithm.

1.6.3 Convergence of conjugate gradients

Firstly we remark that since $\mathbf{e}_N \perp_A \mathcal{K}_N = \mathbb{R}^N$ space, then, in absence of round-off error, $\mathbf{e}_N = 0$, i.e. CG converges in at most N steps. More precisely define

$$
\begin{aligned}
d(\mathbf{r}_0, A) &= \text{dimension of largest Krylov space defined by } (\mathbf{r}_0, A) \\
&\leq \text{number of distinct eigenvalues of } A.
\end{aligned}
$$

Then in exact arithmetic CG converges to the exact solution in precisely $d(\mathbf{r}_0, A)$ steps.

The convergence of the CG method is strongly dependent on the condition number of the matrix A, defined as (see also definition 1.2.13) $\kappa = \kappa(A) = \lambda_{max}/\lambda_{min}$. The following theorem gives a bound on the convergence of CG.

Theorem 1.6.2 (cf. Theorem 9.4.12 in Hackbusch[44]) *The error* $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k$ *after* k *steps of the C.G. method satisfies:*

$$\| \mathbf{e}_k \|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \| \mathbf{e}_0 \|_A$$

Proof We follow the treatment in Hackbusch[44]. From (1.6.4) and using the fact that the CG method is an optimal polynomial method, then clearly: $\| \mathbf{e}_k \|_A \leq \| \chi_k(A) \mathbf{e}_0 \|_A$, where $\chi_k(t)$ is any polynomial of degree k satisfying $\chi_k(0) = 1$. Now, write $\mathbf{e}_0 = \sum_{j=1}^n \gamma_j \mathbf{v}_j$, where $\mathbf{v}_j$ are eigenvectors, and also write $a = \lambda_{min}(A)$, $b = \lambda_{max}(A)$. Then the following holds

$$\begin{aligned} \| \chi_k(A) \mathbf{e}_0 \|_A &= \left(\sum_{j=1}^n \gamma_j^2 \lambda_j \chi_k(\lambda_j)^2 \right)^{\frac{1}{2}} \\ &\leq \max_j | \chi_k(\lambda_j) | \| \mathbf{e}_0 \|_A \\ &\leq \max_{t \in (a,b)} | \chi_k(t) | \| \mathbf{e}_0 \|_A . \end{aligned} \tag{1.6.13}$$

For a general polynomial $\chi_k(t)$ the best bound for (1.6.13) is well known to be given by the scaled and translated Chebyshev polynomial[76]

$$\chi_k(t) = \left[T_k \left(\frac{b+a}{b-a} - \frac{2t}{b-a} \right) \right] / \left[T_k \left(\frac{b+a}{b-a} \right) \right] . \tag{1.6.14}$$

For $t \in [a, b]$ the numerator in (1.6.14) lies in $[-1, 1]$. The kth Chebyshev polynomial is given by

$$T_k(t) = \frac{1}{2} \left[(t + \sqrt{t^2 - 1})^k + (t - \sqrt{t^2 - 1})^k \right] ,$$

so that

$$T_k \left(\frac{b+a}{b-a} \right) \geq \frac{1}{2} \left(\frac{1 + 1/\sqrt{\kappa}}{1 - 1/\sqrt{\kappa}} \right)^k$$

and the result follows. □

Note that if it was desired that the relative error satisfy the following

$$\frac{\| \mathbf{e}_k \|_A}{\| \mathbf{e}_0 \|_A} \leq \epsilon,$$

then the bound in the previous theorem can be re-arranged to obtain

$$k \geq \frac{1}{2} \mid ln(\epsilon/2) \mid \sqrt{\kappa}, \tag{1.6.15}$$

which gives the required number of iterations to achieve this tolerance. From this result it can be seen that the CG algorithm converges at a rate determined by $\sqrt{\kappa}$.

It is well known that the CG algorithm often converges faster than (1.6.15) would indicate. This is primarily due to the convergence rate depending on the entire eigenspectrum of the matrix A, not just the extreme values. Clustering of eigenvalues plays a very important role in the convergence rate, see van der Sluis and van der Vorst[87]. Considerable work has been done on the conjugate gradient method and its convergence for certain classes of problems, see Axelsson et al[16, 15, 12, 13], van der Sluis and van der Vorst[88], Ramage[74], Greenbaum et al[40, 39, 41] and many others.

1.6.4 Lanczos connection to CG algorithm

The Lanczos algorithm, for computing eigenvalues and eigenvectors of symmetric positive definite matrices is closely connected to the CG algorithm as is now outlined. Note that this connection is the basis for a slightly modified CG algorithm that can solve symmetric indefinite systems. It is also sometimes used in order to calculate the eigenvalues of the system whilst a CG solve is being performed, perhaps to determine when to halt the algorithm, by estimating the condition number[7], or adaptively determine a preconditioner[3, 8].

The Lanczos method generates orthonormal vectors in the following way: Let $\mathbf{v}_0$ be a vector such that $\| \mathbf{v}_0 \|_2 = 1$ and let $\mathbf{v}_{-1} = 0$. An orthogonal basis for $\mathcal{K}_k(\mathbf{v}_0, A)$ can be constructed by the recurrence:

$$\gamma_{j+1}\mathbf{v}_{j+1} = A\mathbf{v}_j - \delta_j\mathbf{v}_j - \gamma_j\mathbf{v}_{j-1}, \quad 0 \leq j \leq k-1 \tag{1.6.16}$$

where $\delta_j = (\mathbf{v}_j, A\mathbf{v}_j)$ and γ_{j+1} is chosen such that $\| \mathbf{v}_{j+1} \|_2 = 1$.

Write $V_k = [\mathbf{v}_0, .., \mathbf{v}_{k-1}]$ and $T_k = tri[\gamma_j, \delta_j, \gamma_{j+1}]$, $0 \leq j \leq k-1$, then (1.6.16) is

equivalent to

$$AV_k = V_k T_k + \gamma_k [0, .., 0, \mathbf{v}_k] \tag{1.6.17}$$

and $V_k^T A V_k = T_k$. The Lanczos algorithm constructs the orthonormal set $\{\mathbf{v}_j\}$ and uses the eigenvalues of the T_k as estimates of the the eigenvalues of A.

Now, consider the case for the CG algorithm, the residual and direction vectors satisfy a relation of the form:

$$\begin{aligned} \mathbf{r}_{j+1} &= \mathbf{r}_j - \alpha_j A(\mathbf{r}_j + \beta_{j-1}\mathbf{p}_{j-1}) \\ &= -\alpha_j A\mathbf{r}_j + \left(1 + \frac{\alpha_j \beta_{j-1}}{\alpha_{j-1}}\right)\mathbf{r}_j - \left(\frac{\alpha_j \beta_{j-1}}{\alpha_{j-1}}\right)\mathbf{r}_{j-1}, \\ \text{i.e. } AR_k &= R_k S_k - \frac{1}{\alpha_k}[0, .., 0, \mathbf{r}_k]. \end{aligned} \tag{1.6.18}$$

Here we have written

$$\begin{aligned} R_k &= [\mathbf{r}_0, .., \mathbf{r}_{k-1}] \\ S_k &= tri\left[-\frac{1}{\alpha_{j-1}}, -\frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}}, -\frac{\beta_j}{\alpha_j}\right]. \end{aligned}$$

Now write $\Delta_k = diag(\| \mathbf{r}_0 \|_2, .., \| \mathbf{r}_k \|_2)$ and postmultiply (1.6.18) by Δ_k^{-1} whilst setting $\tilde{V}_k = R_k \Delta_k^{-1}$ to obtain

$$A\tilde{V}_k = \tilde{V}_k \tilde{T}_k - \frac{\sqrt{\beta_{k-1}}}{\alpha_{k-1}}[0, .., 0, \tilde{\mathbf{v}}_k].$$

This is of an identical form to (1.6.17) so that we have $\tilde{V}_k = V_k$ and $\tilde{T}_k = T_k$. This means that the normalised residuals generated by CG are the Lanczos vectors. In fact the CG iterates $\mathbf{x}_k$ can be recovered directly from the Lanczos iterates.

1.6.5 Preconditioners

We can define a preconditioning of the linear system by the following

Definition 1.3 *Given a linear system*

$$A\mathbf{x} = \mathbf{b}, \tag{1.6.19}$$

where A is symmetric positive definite, we define a symmetric positive definite matrix M as the preconditioner such that we solve the equivalent problem

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{1.6.20}$$

Clearly, the system (1.6.20) is not symmetric, but it can be shown that, in the CG algorithm, solving the symmetric positive definite preconditioned system

$$\left(M^{-1/2}AM^{-1/2}\right) M^{1/2}\mathbf{x} = M^{-1/2}\mathbf{b},$$

is equivalent to solving a system of the form (1.6.20).

Implementation of the preconditioning (1.6.20) in the CG algorithm involves one extra computational step per iterate, more details are given in Golub and van Loan[38, §10]. This extra step involves solutions of the linear system

$$M\mathbf{z}_k = \mathbf{r}_k. \tag{1.6.21}$$

Since $\mathbf{r}_k$ is the residual at the k^{th} step, the vector $\mathbf{z}_k$ is often referred to as the *preconditioned residual.* We remark that $\mathbf{z}_k$ is often used in the stopping criterion (rather than $\mathbf{r}_k$) since, on the assumption that the preconditioner approximates A, then $\mathbf{z}_k = M^{-1}\mathbf{r}_k \approx A^{-1}\mathbf{r}_k = \mathbf{e}_k$.

For the matrix M to be an effective preconditioner we generally require that

- linear systems of the form $M\mathbf{z} = \mathbf{r}$ are easy to solve,
- convergence of the preconditioned problem (1.6.20) is faster, in terms of computational time, than the original problem (1.6.19).

The preconditioned problem has convergence properties dependent on the eigenspectrum of the preconditioned matrix $M^{-1}A$. Hence the condition number of the symmetrically preconditioned matrix $M^{-1/2}AM^{-1/2}$ is important, (cf. (1.6.15)). Clearly a good preconditioner M would be such that $M^{-1}A \approx I$.

Diagonal scaling

One of the most commonly used preconditioners is diagonal scaling. Here the preconditioner M is taken to be the diagonal of the matrix A. Solutions of the linear system involving M are clearly very straightforward.

Although very simple, this preconditioning is often highly effective. One of the reasons why this is so is because a diagonal preconditioner is equivalent to *scaling* the problem by the diagonal. The following simple example illustrates the potential effect of this.

Example 1.1 *Consider CG applied to a problem with*

$$A = \begin{bmatrix} 1 & -10^{-5} \\ -10^{-5} & 10^{-8} \end{bmatrix}.$$

The standard "condition number" theory of theorem 1.6.2 would indicate that a problem that featured this form of entries would converge in $O(10^4)$ steps, since the condition number is $O(10^8)$. However, by symmetrically diagonally scaling this matrix we are essentially applying CG to the problem

$$M^{-1/2}AM^{-1/2} = \begin{bmatrix} 1 & -0.1 \\ -0.1 & 1 \end{bmatrix}.$$

This has condition number $O(1)$, and hence would be expected to converge in just a single CG step.

Where possible, this diagonal scaling is often implemented in an explicit fashion, i.e. the matrix is actually symmetrically scaled and CG applied to the problem

$$M^{-\frac{1}{2}}AM^{-\frac{1}{2}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}},$$

or

$$\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}.$$

In some cases this then allows a small reduction in the work per matrix-vector multpli-

cation, since the diagonal is then known to be unity.

A number of authors have commented on the effect of diagonal scaling, see for example Ramage and Wathen[74] and Greenbaum et al[41, 40]. In particular the diagonal scaling is known to greatly mitigate the effect of the discontinuous coefficients for discretisations of problems such as (1.2.3). In particular this effect is illustrated in figure 2-12, where we compare the convergence of the two grid method in the cases with and without for diagonal scaling for cases with and without discontinuous coefficients.

1.6.6 Poor performance of CG on a realistic problem

In spite of the nice theoretical properties of the convergence of the conjugate gradient method, in practice CG with no preconditioning often performs very poorly indeed. Convergence is either extremely poor or sometimes there is no noticable convergence at all. It is generally believed that the main reason for this is due to computational round-off in the CG algorithm, and so in practice some form of preconditioning is often *required* to give convergence.

In this section we very briefly give some results showing the convergence of the conjugate gradient algorithm: First a model problem on a regular mesh, with two different levels of refinement, resulting in discretisations of sizes $N = 32896$ and $N = 56616$, and second on a problem arising from a physical realisation of (1.2.3) in 2D with three different sizes of discretisation, $N = 3684, 14736, 58944$. The first problem is based on a regular mesh of the type given in figure 1-9 and has a constant permeability over the entire domain. Quadratic basis elements are again used for this discretisation. The second problem uses meshes that are all based on a grid of the form in figure 1-1 with permeability coefficients in table 1.1. For the reasons discussed above a diagonal preconditioner is used for all these problems.

In figure 1-10 we plot the log of the 2-norm of the relative error in solution against CG iterations for the first problem. It is clear that CG converges reasonably well in these cases, although we note that the direct solver requires less CPU time than the iterative method for both $N = 32896$ and $N = 56616$.

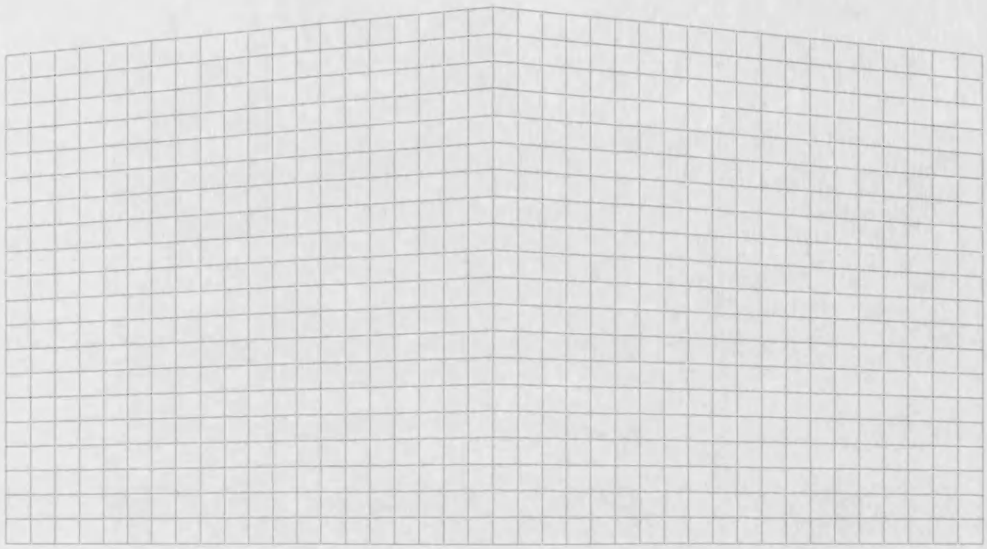

Figure 1-9: *Simple grid for a model 2D problem.*

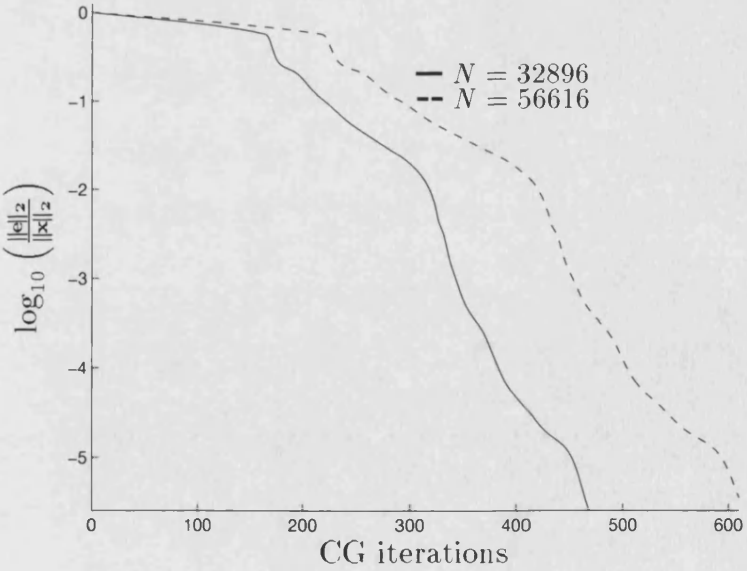

Figure 1-10: *Convergence of diagonally preconditioned CG, on a model 2D problem with a regular mesh, for 2 sizes of FEM discretisations of (1.2.3).*

In figure 1-11 we plot the log of the 2-norm of the relative error in solution against CG iterations for the realistic problem. It is immediately apparent that diagonally preconditioned CG converges very poorly in this case. Even for the small problem convergence is very slow indeed, for the larger problems there is very little convergence at all. The highly irregular mesh would appear to be the main cause of the difficulties for this problem. It is important to note that the frontal solver gives a solution in significantly less time than that taken by any of the iterative results shown in figure 1-11.

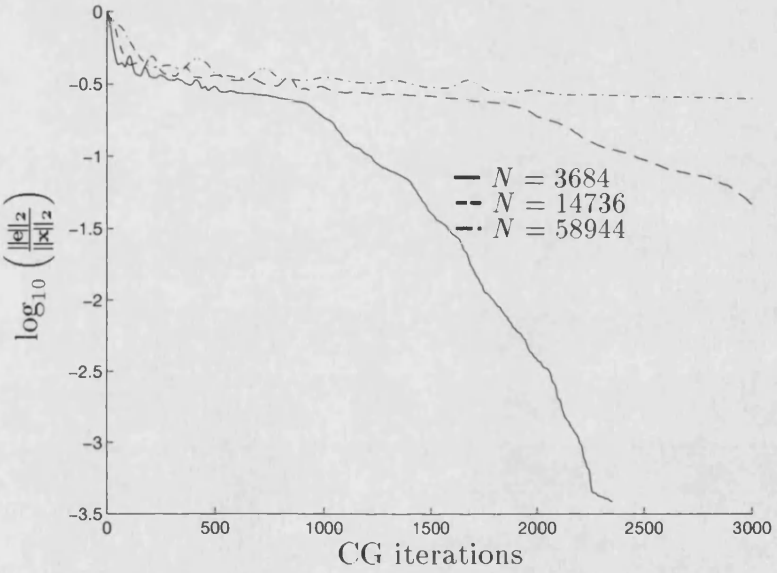

Figure 1-11: *Convergence of diagonally preconditioned CG, on a 2D realistic problem, for 3 sizes of FEM discretisations of (1.2.3).*

To solve this problem effectively in an iterative fashion it will be necessary to find a much better preconditioner than diagonal scaling. Choices such as a two grid preconditioner (§2.4), a polynomial preconditioner (chapter 4) or an incomplete factorisation, (chapter 5) may be the answer to this. An alternative approach may be to try a different iterative method, such as the two grid method in chapter 2.

1.7 Overview of thesis and results

In chapter 1 we have introduced the groundwater flow model (§1.2), and the finite element discretisation being used (§1.3). The direct method currently being used in the

Harwell code NAMMU has been discussed (§1.4). Issues relating to the implementation of a solution method, particularly the storage of the finite element stiffness matrix (§1.5) have been addressed and a discussion of conjugate gradients has been given in (§1.6). It is clear that conjugate gradients alone is not sufficient to provide a robust solution technique for the physical problems we are considering. The remainder of the thesis is therefore concerned with discussing other iterative methods or variants on the original conjugate gradient method.

Chapter 2 introduces the two grid method. For finite difference methods on uniform meshes this method is very widely used, and has been shown to be highly effective, especially in the more general multigrid. The use of the two grid method for solving linear systems arising from finite element methods does not appear to be so common, particularly with non-uniform meshes. However, although almost all the theory for these methods has been developed with finite difference methods in mind, this is not a requirement for two grid methods. Implementation for finite elements therefore follows the same general approach, the main difference being that for finite elements there is a very natural way of performing the grid transfer operations required for the two grid method, this is discussed in some detail in §2.2. For two grid methods the effectiveness of the smoothing operation is very important. We discuss a number of possible choices of "matrix-free" smoothers (§2.3) and give results for their effectiveness on both a simple model problem and a more complicated physical example (§2.6). These results indicate that use of conjugate gradients as a smoother in the two grid method is highly effective for the problems we are concerned with. For a reasonably large 2D problem (60,000 unknowns) the two grid method is able to produce a solution in nearly a third of the time required by the direct solver. A three grid method is also briefly discussed in this chapter, although for the problems we discuss it is found to be less effective than the two grid method. In §2.7 the use of the two grid method for solving non-symmetric linear systems arising from finite element discretisations of a radionuclide transport problem is briefly discussed. This method differs from the symmetric case only in that a non-symmetric smoother is now neeeded. For this we use conjugate gradients applied to the normal equations. For a small simple model problem on a uniform mesh this approach seems to work well.

Chapter 3 is primarily concerned with the use of the two grid method, discussed in the

previous chapter, applied to a large 3D problem. Here we are only concerned with using conjugate gradients as a smoother. In this case the two grid method is considerably faster than the original direct method, mostly as a result of the n^7 work scaling for the direct solver on 3D problems, where n corresponds to the number of elements in each spatial direction. The three grid method is also tested on this problem, and once again it is not as effective as the two grid method, although does show slightly more uniform convergence properties.

Chapter 4 is concerned with the use of polynomial preconditioners for the conjugate gradient algorithm. The idea here is produce a low order polynomial in the problem matrix A such that it can be used as a preconditioner. We discuss two approaches to finding such polynomials, resulting in least squares polynomials and Chebyshev polynomials. We test these polynomials on both a large problem with a uniform mesh and a large 2D problem from a physical application. This polynomial approach is found to be ineffective at producing a preconditioner that allows fast solutions of the problems we are concerned with.

Chapter 5 discusses the use of the incomplete factorisation methods as a preconditioner for conjugate gradients. We discuss the various issues relating to the use of incomplete factorisations, in particular the problems with the method for the matrices from our application. An "off the shelf" implementation is briefly discussed and we present results for a large 2D problem. This implementation is shown to be very ineffective for our problem.

Chapter 6 gives some brief indication of the areas that still require future work.

Chapters 2 and 3 have been written as a paper and submitted to the Journal of Computational Physics.

Chapter 2

Two Grid Method

2.1 Introduction

In this chapter we shall demonstrate the effectiveness of the two grid method as a solution technique for discretisations of (1.2.3), for groundwater flow problems of the type introduced in chapter 1.

A number of different choices of iterative methods for use as a smoother in the two grid method are considered. In particular conjugate gradients is shown to be a very effective choice.

Extensive literature is available on the subject of two grid methods or the more general multigrid methods. Considerable work in this area has been done by Hackbusch et al, in particular see [43, 44, 45, 46]. The use of conjugate gradients as a smoother in multigrid methods is discussed by Bank and Douglas[19], Kettler[63], Deconinck and Hirsch[31], and Braess[23]. We will discuss more on the available theory for this combination in §2.3.4.

In all cases where it is appropriate, the CPU time taken by the two grid method will be compared with the CPU time taken by the direct solver MA32 from the Harwell subroutine library.

As discussed in chapter 1, the use of biquadratic finite elements on quadrilaterals results in stiffness matrices that are unlikely to be either diagonally dominant or M-matrices, and hence much of the standard convergence theory for iterative methods (e.g. as discussed in [43, §6]) is not applicable. Also there are restrictions in the choice of smoothers, since the NAMMU package provides the stiffness matrix in an element form. In this form matrix-vector multiplications can be carried out very efficiently, but more complicated operations, involving the whole matrix A, or parts of the matrix, are not easily performed. For this reason when discussing the smoothers in this chapter we will restrict attention to iterative methods that only require matrix-vector multiplications. This eliminates Gauss-Seidel and SOR smoothers.

The plan of the chapter is as follows. The standard formulation of a two grid method is stated in §2.2, and follows the description in [43]. In §2.3 the following choices of smoother are considered:

- Richardson iteration
- Gradient method
- Conjugate gradients (CG)
- Conjugate residuals (CR).

Each of these methods is a "matrix-free" method, where the linear system matrix is only known by its action on a vector, i.e. only matrix-vector multiplications are required. In each case the linear system is either symmetrically diagonally scaled or a diagonal preconditioner is used. Note that Richardson's method with a diagonal preconditioner is identical to Jacobi's method. Also, in order to analyse the effect of the smoothers discussed above we carry out a "computational modal analysis" on a 2D model problem, with $N = 1000$, to compare their effectiveness.

In §2.4 the use of the two grid method as a preconditioner for the conjugate gradient method is explained.

In §2.5 an extension to the two grid approach, now using three grids, is briefly discussed. The impressive results from much of the literature on multigrid methods, for model

problems, would indicate that more levels of grids than just two are needed to take full advantage of the general approach. Using three grids is then seen as a first step towards a full multigrid approach.

In §2.6 we first present results for a 2D problem arising from an actual physical realisation of an underground rock structure. Both the two grid method (§2.2), with the various smoothers, and the preconditioned conjugate gradient method (§2.4) are considered. The conjugate gradient and conjugate residual methods are shown to be effective smoothers, whereas use of Richardson iteration or the gradient method as smoother gives poor overall rates of convergence. To illustrate that diagonal scaling can mitigate, almost completely, the ill-conditioning due to the variations in k, we also compare results for two related problems. Both use the same grid, but one has $k = 1$ uniformly over the entire region, instead of the variations given in table 1.1. Using the two grid method as a preconditioner for the conjugate gradient method is shown to give satisfactory convergence, although inferior to the standard two grid method with either conjugate gradients or conjugate residuals as a smoother. Some results from the three grid method (§2.5) are also presented in §2.6.2.

In §2.7 we briefly discuss an extension of the two grid method to a non-symmetric problem arising from a radionuclide transport problem. The conjugate gradient smoother is now replaced by conjugate gradients applied to the normal equations. On a problem involving a uniform mesh this approach seems to give a good rate of convergence. Note that merely using conjugate gradients on the normal equations does not give an adequate rate of convergence.

In §2.8 we summarise our experiences with the two grid method on the 2D problems. The two grid method is seen to be reasonably effective for the problem discussed here, converging to a solution in less computational time than required by the direct solver.

2.1.1 Notation and coarsening strategy

For the rest of this chapter we shall denote a fine grid linear system problem as

$$A_f \mathbf{x} = \mathbf{b},$$

this being a problem with N_f degrees of freedom. The corresponding coarse grid problem will have N_c degrees of freedom and the associated stiffness matrix will be denoted A_c, and we shall assume that each coarse grid point is also a point on the fine grid.

The strategy employed in order to obtain the coarser grids is to halve the number of elements in each spatial direction on the fine grid to produce the coarse grid. Since, for the finite element method, the number of degrees of freedom in the resulting stiffness matrix is proportional to the number of elements this means that $N_c \approx \frac{1}{4}N_f$ in the 2D case. However, it is worth noting at this stage that we will never consider a coarsening that removes any of the permeability regions completely. As a result of this there is a finite limit to the number of coarser grids that can be considered. This is particularly true in the 3D case, discussed in chapter 3 where there will already be only a few elements in each spatial direction for each permeability region. This is principally the reason why a full multigrid method, with a large number of grids, is not considered at this point. In addition, by using this method of coarsening, problem dependent methods for the prolongation, such as in [30], are not required.

2.2 Two Grid Method

The two grid method employed, which will henceforth be referred to as TGM, is taken from [43]. We denote prolongation and restriction matrices that transfer vectors between the grids by P and R respectively. Also we assume that an LU factorisation of A_c has been obtained before starting any TGM steps and so solves on the coarse grid require only forward and back substitutions.

The steps in the TGM are as follows: from an initial vector $\mathbf{x}_0$ we perform a coarse grid correction followed by $\mu(\geq 1)$ smoothing steps to produce $\mathbf{x}_1$:

$$\mathbf{r} = \mathbf{b} - A_f\mathbf{x}_0 \tag{2.2.1}$$

$$\mathbf{r}_c = R\mathbf{r} \tag{2.2.2}$$

$$\mathbf{d}_c = A_c^{-1}\mathbf{r}_c \tag{2.2.3}$$

$$\mathbf{x}_{\frac{1}{2}} = \mathbf{x}_0 + P\mathbf{d}_c \tag{2.2.4}$$

$$\mathbf{x}_1 = S^{\mu}(\mathbf{x}_{\frac{1}{2}}) \tag{2.2.5}$$

Note that steps (2.2.1-2.2.4) can be together written as the *coarse grid correction* step

$$\mathbf{x}_{\frac{1}{2}} = \mathbf{x}_0 + PA_c^{-1}R(\mathbf{b} - A_f\mathbf{x}_0). \tag{2.2.6}$$

Step (2.2.5) represents the application of μ steps of the *smoothing iteration* and one step of the TGM can be written as

$$\mathbf{x}_1 = S^{\mu}[\mathbf{x}_0 + PA_c^{-1}R(\mathbf{b} - A_f\mathbf{x}_0)]. \tag{2.2.7}$$

In order to solve a given problem, this two grid step is applied a number of times until some preset convergence criterion is met.

It should be noted that coarse grid correction steps by themselves are insufficient to guarantee a convergent algorithm as is illustrated in the following example taken from [43]. Note, however, that the addition of one smoothing step is, in general, sufficient to ensure convergence.

Example 2.1 *Consider applying the TGM steps above with $\mu = 0$, i.e. no smoothing steps. Since the restriction matrix, R, is rectangular it has a non-trivial kernel. So, choose $\mathbf{0} \neq \mathbf{w} \in \ker(R)$ and set $\mathbf{x}_0 = A_f^{-1}(\mathbf{b} - \mathbf{w})$. Thus $\mathbf{r} = \mathbf{w}$ and since $R\mathbf{w} = \mathbf{0}$, $\mathbf{x}_{\frac{1}{2}} = \mathbf{x}_0$. Thus there is no improvement in $\mathbf{x}_0$.*

Consider further the addition of one step of a Richardson iteration as a smoother, i.e. write

$$\tilde{\mathbf{x}}_0 = \mathbf{x}_0 + (\mathbf{b} - A_f\mathbf{x}_0) = \mathbf{x}_0 + \mathbf{w}$$

and now apply the coarse grid correction step on the residual $\tilde{\mathbf{r}} = \mathbf{w} - A_f\mathbf{w}$. Assuming that $A_f\mathbf{w}$ does not lie in the kernel of R then the coarse grid correction step now gives an improvement. Generically we would expect this to be the case.

The choices of smoothers will be examined in more detail in §2.3, together with some

discussion on convergence analysis. In the following two subsections we consider the choice of prolongation and restriction matrices for the finite element discretisation of (1.2.3).

2.2.1 Prolongation

First denote the i^{th} FEM nodal basis function for the fine grid as Φ_i^f, and the m^{th} coarse grid basis functions as Φ_m^c. By requiring that all coarse grid points are also fine grid points it is immediately apparent that the coarse grid basis functions can be written in terms of the fine grid basis functions, that is

$$\Phi_m^c = \sum_{i=1}^{N_f} a_i^{(m)} \Phi_i^f \tag{2.2.8}$$

for some coefficients $a_i^{(m)}$. Hence, if τ_j^f denotes the j^{th} fine grid node point then clearly

$$\Phi_m^c(\tau_j^f) = a_j^{(m)} \tag{2.2.9}$$

since the nodal basis functions satisfy $\Phi_i^f(\tau_j^f) = \delta_{ij}$.

In order to define the prolongation matrix from $\mathbb{R}^{N_c}$ to $\mathbb{R}^{N_f}$, consider any vector $\mathbf{q}^c \in \mathbb{R}^{N_c}$ with $(\mathbf{q}^c)_m = q_m^c$. There is a corresponding function in the coarse finite element space defined by

$$q^c(\tau) = \sum_{m=1}^{N_c} q_m^c \Phi_m^c(\tau). \tag{2.2.10}$$

Evaluation of (2.2.10) at the fine grid nodes τ_j^f, $j = 1, ..., N_f$, gives the corresponding fine grid vector, $\mathbf{q}^f$, with the j^{th} component given by

$$q_j^f = \sum_{m=1}^{N_c} q_m^c a_j^{(m)}. \tag{2.2.11}$$

Hence,

$$\mathbf{q}^f = P\mathbf{q}^c \tag{2.2.12}$$

where the rectangular matrix P is given by

$$P_{jm} = a_j^{(m)}, \tag{2.2.13}$$

these coefficients $a_j^{(m)}$ being given in (2.2.9) above. As a result of our imposed requirement on the choice of coarse grid, see §2.1.1, the coefficient k does not appear in the prolongation.

2.2.2 Restriction

First note that in the matrix-vector description of the TGM the restriction operation in (2.2.2) acts on the fine grid residual and produces a "coarse grid residual". To define an appropriate restriction for the FEM it is natural to consider the residuals of an approximate solution to the weak form of the pde with respect to both the fine and coarse meshes.

Given some approximate solution on the fine grid, p^f, the i^{th} residual corresponding to the i^{th} basis function is given by

$$r_i^f = \int_\Omega k\nabla p^f.\nabla\Phi_i^f - \int_{\partial\Omega} \Phi_i^f k\nabla p^f.\mathbf{n} \tag{2.2.14}$$

where Ω and $\partial\Omega$ denote the domain and the domain boundary respectively, $\mathbf{n}$ denotes the outward unit normal and Φ_i^f is the i^{th} fine grid basis function. Correspondingly, on the coarse grid the m^{th} residual corresponding to p^f is

$$r_m^c = \int_\Omega k\nabla p^f.\nabla\Phi_m^c - \int_{\partial\Omega} \Phi_m^c k\nabla p^f.\mathbf{n} \tag{2.2.15}$$

where Φ_m^c is the m^{th} coarse grid nodal basis function. Using (2.2.8) we have

$$r_m^c = \sum_{i=1}^{N_f} a_i^{(m)} \left(\int_\Omega k\nabla p^f.\nabla\Phi_i^f - \int_{\partial\Omega} \Phi_i^f k\nabla p^f.\mathbf{n} \right) \tag{2.2.16}$$

which by (2.2.14), gives

$$r_m^c = \sum_i a_i^{(m)} r_i^f. \tag{2.2.17}$$

In matrix form we can write

$$\mathbf{r}^c = R\mathbf{r}^f \tag{2.2.18}$$

where $\mathbf{r}^c$ and $\mathbf{r}^f$ denote the vectors with components r^c_m and r^f_i respectively. The rectangular matrix R is given by

$$R_{mj} = a^{(m)}_j \tag{2.2.19}$$

with the coefficients $a^{(m)}_j$ given by (2.2.9) above. Again there is no dependence on the coefficient k.

Finally, by comparing the definitions of the prolongation and restriction matrices, i.e. (2.2.13) and (2.2.19), we see that

$$R = P^T. \tag{2.2.20}$$

From a computational point of view this has obvious advantages; only the prolongation matrix needs to be calculated thus halving both the calculations required and the storage requirements for this one off calculation. Standard NAMMU routines exist to perform the majority of the above calculations, so in the NAMMU environment implementing this approach is relatively simple. In addition, when the two grid iteration is used as a preconditioner for the conjugate gradient method (§2.4), the choice (2.2.20) ensures that the preconditioning matrix is symmetric, which is important to ensure that the resulting method will converge, see [38, §10.3]. In Hackbusch[43, §3.6], this relation between R and P is discussed, in a more abstract form, and is called the "canonical" choice for a finite element method. The idea of using the weak form of the pde to define the restriction matrix, which links the residuals on the two grids, is well known, see for example [31].

2.3 Different smoothers on a model problem

The overall philosophy of the two grid method, and multigrid methods in general, is well known (see [19, 43, 24]) and so we merely sketch the main points which are relevant to the convergence of the two grid method. In order to provide an efficient solution procedure the smoother and coarse grid solver must work in tandem. For theoretical

purposes only, consider an expansion of the error in a fine grid solution, $\mathbf{x}_i$ say, of the form

$$\mathbf{e}_i = \mathbf{x} - \mathbf{x}_i = \sum_{j=1}^{N_f} \alpha_i^{(j)} \mathbf{v}_j \tag{2.3.1}$$

where $\mathbf{v}_j$ are eigenvectors of A_f, corresponding to eigenvalues $\lambda_j (\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{N_f})$. As a rough classification, the error components, $\alpha_i^{(j)}$, for $j = 1, ..., N_c$ are termed the "low frequency" components, and for $j = N_c + 1, ..., N_f$ they are the "high frequency" components. (Recall that for a typical 2D problem we will have $N_c \approx \frac{1}{4} N_f$). Now, the hope is that the smoothing iteration should damp out the high frequency errors, while the coarse grid correction should reduce the low frequency errors. However, as is seen by the simple example in [43, p27] and §2.3.1 given below, the coarse grid correction step also redistributes the error over all frequencies. We see this effect clearly in a computational modal analysis on a model problem later in §2.3.5.

Recall that for our smoothers we only consider iterative methods that are based on matrix-vector multiplications. This leads to the four choices: Richardson iteration, gradient method, conjugate gradient method and conjugate residual method. For each of these methods a diagonal preconditioning is also applied. For more details on these iterative methods see [38, 44].

In order to understand the convergence properties of stationary smoothers it is common to examine the reduction on each eigenmode, or frequency, of the error, see Hackbusch [43, §2.4]. For example, the following simple analysis gives the best choice of damping for the Richardson method on a 2D problem. Define the damped Richardson method as:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \omega(\mathbf{b} - A\mathbf{x}_i), \tag{2.3.2}$$

now, by writing the error as in (2.3.1) we can write the error component in the j^{th} eigendirection after one Richardson step as

$$\alpha_{i+1}^{(j)} = (1 - \omega\lambda_j)\alpha_i^{(j)}.$$

With the need to smooth high frequency components in mind, we require that $|1 - \omega\lambda_j|$ be minimised for all those eigenvalues in the fine grid component range, which, for the

2D problem, is taken to be approximately

$$\frac{\lambda_N}{4} \leq \lambda_j \leq \lambda_N$$

corresponding to the high frequency range of eigenvectors (when $N_c \approx \frac{1}{4}N_f$). We deduce that the best choice for ω is

$$\omega_{\text{best}} = \frac{8}{5\lambda_N}. \tag{2.3.3}$$

For the theoretical modal analysis to be useful for general matrices and other smoothers it is best if simple expressions for the eigenvectors are known. In §2.3.1 we discuss, for completeness, the standard approach to analysing the convergence of the two grid method on a model problem, where these expressions are known. However, for the matrices derived from the FEM with biquadratic basis functions over 9 node quadrilaterals simple expressions are not known. In general theoretical analysis of the non-stationary smoothers is not easy and it is difficult to interpret the results in a helpful way. Hence the "computational modal analysis" in §2.3.5 is used to compare the four smoothers discussed here.

2.3.1 Theoretical Modal Analysis

From the literature there appears to be two main approaches to the analysis of the convergence of the two grid method or TGM. However, the common start point is to consider the idea of an iteration matrix, that is, we look for a matrix M such that

$$\mathbf{e}_{m+1} = M\mathbf{e}_m \tag{2.3.4}$$

where $\mathbf{e}_m$ is the error at the m^{th} step.

By re-arranging (2.2.7) the TGM iteration can be equivalently written in the form of (2.3.4) by

$$M_{\text{TGM}} = M_{\text{TGM}}(\mu) = S^{\mu}(I - PA_c^{-1}RA_f) \tag{2.3.5}$$

which represents a coarse grid correction followed by the application of μ smoothing steps. Note that a different, but related method, can be produced by considering the

reverse of this, that is μ smoothing steps, followed by a coarse grid correction, this gives the alternative form

$$M_{\text{TGM}} = M_{\text{TGM}}(\mu) = (I - PA_c^{-1}RA_f)S^{\mu}. \tag{2.3.6}$$

For simple problems and stationary smoothers it is then possible to compute $\rho(M_{\text{TGM}})$, the spectral radius, giving an exact convergence rate, this will be done later for a simple model problem in §2.3.2. For harder problems and for non-stationary smoothers, this is less easy.

It is here that the literature differs slightly, Hackbusch[43, 44] analyses the convergence by writing (2.3.6) in the form

$$M_{\text{TGM}} = [A_f S^{\mu}][A_f^{-1} - PA_c^{-1}R] \tag{2.3.7}$$

and then

$$\| M_{\text{TGM}} \| \leq \| A_f S^{\mu} \| \, \| A_f^{-1} - PA_c^{-1}R \| . \tag{2.3.8}$$

The two expressions on the right hand side are then estimated separately, the idea being that it is easier to form these estimates individually than to analyse the whole expression. The factor $\| A_f^{-1} - PA_c^{-1}R \|$ is referred to as *the approximation property*, whilst $\| A_f S^{\mu} \|$ is referred to as *the smoothing property*. More details on these ideas, and some estimates of these properties for a number of model problems, are given in Hackbusch[43, §6.1-6.3].

In contrast Bank and Douglas[19] try to estimate convergence by analysing how the smoothing iteration complements the coarse grid correction step. If it is assumed that the coarse grid essentially removes all the low frequency errors then any work done by the smoother on these errors is extra work. The effectiveness of the smoother is then considered by examining how it performs on the error components not touched by the coarse grid. If this can be done it leads to a much sharper estimate of convergence rate; however, in general, it is hard to calculate these estimates, see §2.3.4.

2.3.2 Richardson iteration - Analysis for a model problem

As mentioned earlier in §2.3.1 the "ideal" approach to analysing convergence of the two grid method would involve estimating the spectral radius of the iteration matrix M_{TGM}, given in (2.3.5). For model problems on uniform grids this can often be performed in a quite straightforward fashion.

We shall now consider the use of the two grid method with Richardson iteration as a smoother on the following model problem

$$-u''(x) = f \tag{2.3.9}$$

with Dirichlet boundary conditions, $u(0) = u(1) = 0$. The fine grid is assumed to be uniform and has $h = 2^{-1-l}$, where l is some positive integer, hence N_f, the number of nodes on this grid, is given by $N_f = \frac{1}{h} - 1$ and is clearly an odd integer.

The coarse grid will be assumed to have twice the grid spacing of the fine grid, i.e. $h_c = 2h_f$, so that N_c is also odd. Using a FEM discretisation, with linear basis functions, gives the matrix problem

$$h_f^{-1}\begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & \end{bmatrix} \mathbf{x} = \mathbf{b} \tag{2.3.10}$$

for the fine grid.

For the following analysis we require the eigenvalues and eigenvectors of the fine and coarse matrices from (2.3.10), so put

$$\mathbf{v}_i = \sqrt{2h}\,(sin(ij\pi h))_{j=1}^{N}, \quad \text{for} \quad i = 1, .., N \tag{2.3.11}$$

with N as N_f and N_c for the fine and coarse grid respectively. These eigenvectors have

corresponding eigenvalues given by

$$\lambda_i = 4h^{-1}sin^2(i\pi h/2). \tag{2.3.12}$$

From this formulation it is now possible to give a convergence analysis for the TGM using Richardson iteration, with diagonal preconditioning, as a smoother.

In order to find an estimate for the convergence rate of the TGM the standard method proceeds in the following way: begin by writing the damped Richardson iteration matrix as

$$S = I - \omega D^{-1} A_f \tag{2.3.13}$$

and the usual damping coefficient for this problem is to take [43, §3.2]

$$\omega = \frac{1}{2}. \tag{2.3.14}$$

Note that an argument similar to that in §2.3 would suggest the best choice is given by $\omega = 2/3$, since in this case the maximum eigenvalue is bounded by 2. However, the choice (2.3.14) does give an error reduction in the high frequencies of at least 1/2 per Richardson step and also greatly simplifies the following analysis.

Now, by constructing unitary transformation matrices Q_f and Q_c by a suitable ordering of the eigenvectors (2.3.11), as follows

$$Q_f = [\mathbf{v}_1^f, \mathbf{v}_{N_f}^f, \mathbf{v}_2^f, \mathbf{v}_{N_f-1}^f, ..., \mathbf{v}_{N_c}^f, \mathbf{v}_{N_f+1-N_c}^f, \mathbf{v}_{N_c+1}^f] \tag{2.3.15}$$

and

$$Q_c = [\mathbf{v}_1^c, \mathbf{v}_2^c, ..., \mathbf{v}_{N_c-1}^c, \mathbf{v}_{N_c}^c]. \tag{2.3.16}$$

Now define the similarity transformed matrices,

$$\hat{M}_{\text{TGM}} = Q_f^{-1} M_{\text{TGM}} Q_f, \quad \hat{S} = Q_f^{-1} S Q_f, \quad \hat{A}_f = Q_f^{-1} A_f Q_f,$$
$$\hat{A}_c = Q_c^{-1} A_c Q_c, \quad \hat{P} = Q_f^{-1} P Q_c, \quad \hat{R} = Q_c^{-1} R Q_f$$

these matrices can be shown to have diagonal block structure, see [43, p26], and hence

the analysis of the iteration matrix (2.3.5) can be reduced to looking at the following set of sub-matrices:

$$A_f^{(t)} = 4h_f^{-1}\begin{bmatrix} s_t^2 & 0 \\ 0 & c_t^2 \end{bmatrix} \quad (1 \le t \le N_c), \quad A_f^{(N_c+1)} = 2h_f^{-1} \tag{2.3.17}$$

$$S^{(t)} = I - \frac{1}{4}h_f^{-1}A_f^{(t)} = \begin{bmatrix} c_t^2 & 0 \\ 0 & s_t^2 \end{bmatrix} \quad (1 \le t \le N_c), \quad S^{(N_c+1)} = \frac{1}{2} \tag{2.3.18}$$

$$R^{(t)} = \sqrt{2}\begin{bmatrix} c_t^2 & -s_t^2 \end{bmatrix}, \quad P^{(t)} = \sqrt{2}\begin{bmatrix} c_t^2 \\ -s_t^2 \end{bmatrix}, \quad (1 \le t \le N_c) \tag{2.3.19}$$

$$A_c^{(t)} = 8h_c^{-1}s_t^2c_t^2 \quad (1 \le t \le N_c + 1), \tag{2.3.20}$$

where we define

$$s_t^2 = \sin^2\left(\frac{t\pi h_f}{2}\right), \quad c_t^2 = \cos^2\left(\frac{t\pi h_f}{2}\right).$$

Hence we combine (2.3.17 – 2.3.20) with (2.3.5) to give the blocks for the TGM iteration matrix as

$$M_{\mathrm{TGM}}^{(t)} = \begin{bmatrix} s_t^2 & c_t^2 \\ s_t^2 & c_t^2 \end{bmatrix}\begin{bmatrix} c_t^2 & 0 \\ 0 & s_t^2 \end{bmatrix}^{\mu} \quad (1 \le t \le N_c), \quad M_{\mathrm{TGM}}^{(N_c+1)} = 2^{-\mu}. \tag{2.3.21}$$

The first matrix corresponds to the coarse grid correction and the second to μ steps of the smoothing process. Recall that these 2x2 matrices were constructed such that complementary high and low frequency errors were grouped together (see equations (2.3.15) and (2.3.16)). It is clear from this that the coarse grid correction step has the effect of averaging the error components corresponding to these complementary high and low frequency errors, whilst also reducing both in magnitude. In §2.3.5 this effect will be demonstrated more clearly.

The overall convergence rate of the TGM iteration can now be estimated by the largest spectral radius of the matrices given by (2.3.21). For varying μ we find, see [43, Thm 2.4.4], that $\rho(M_{\mathrm{TGM}})$ has an asymptotic behaviour of approximately c_1/μ for large μ, where $c_1 = 1/e \approx 0.3679$, hence the iteration will indeed converge.

2.3.3 Gradient method

The gradient method is based on the idea of minimising the error in a certain direction. These "directions" being often referred to as *search directions*. For the standard gradient method the residual vectors are taken as the search directions and the iteration proceeds as follows:

$$\mathbf{x}_{m+1} = \mathbf{x}_m + \alpha \mathbf{r}_m \tag{2.3.22}$$

where $\mathbf{r}_m = b - A\mathbf{x}_m$. The optimal coefficient α can be calculated exactly in terms of the residual vector, $\mathbf{r}_m$ and $A\mathbf{r}_m$, see [44, §9.2].

It is important to note that, applied as a solution technique to a linear system, the gradient method will converge at least as fast as Richardson's method with optimal damping. This is a consequence of the fact that the optimally damped Richardson's method has a convergence rate given by

$$\frac{\lambda_{max}(A) - \lambda_{min}(A)}{\lambda_{max}(A) + \lambda_{min}(A)} = \frac{\kappa(A) - 1}{\kappa(A) + 1}, \tag{2.3.23}$$

where $\kappa(A)$ is the condition number of the matrix A. The convergence rate given by (2.3.23) is precisely the same as that for the gradient method, see [44, pp252-253]. For this reason it is perhaps to be expected that Richardson's method (see §2.3.2) as a smoother is quantitatively the same as for the gradient method.

2.3.4 Conjugate Gradients and Conjugate Residuals

Analysis for these methods, even in the simple 1d problem case, is by no means straightforward, primarily due to the fact that they are non-stationary methods. An attempt to produce an analysis along the lines of that given for Richardson's method in §2.3.2 soon becomes very unwieldy and generally unhelpful if more than a handful of steps are considered.

Some attempt at producing a theoretical framework for the convergence of these methods has been done by Bank and Douglas[19]. Although the theory and approach considered there is of some theoretical interest, it is, however, difficult to see how this approach

applies in a useful practical way to any general problem. They consider the case where the initial error is written as $\mathbf{e}_0 = \sum_{i=1}^{N} c_i \mathbf{v}_i$, where $\mathbf{v}_i$ is an eigenvector corresponding to an eigenvalue λ_i. Then the error after m steps of the CG method can be written as $\mathbf{e}_m = \sum_{i=1}^{m} c_i \mathbf{v}_i p_m(\lambda_i)$ where

$$p_m(t) = \prod_{j=1}^{m} (1 - \tau_j^{-1} t), \tag{2.3.24}$$

and the τ_j lie in $(0, 1]$. Bank and Douglas then give the following theorem

Theorem 2.3.1 (cf. Theorem 11 of [19]) *Suppose there exists $\hat{\kappa} \geq 1$, independent of j, such that for $u \in \mathcal{M}_{j-1}^{\perp} \cap \mathcal{M}_j$,*

$$\| u \|_0 \leq \hat{\kappa}^{1/2} \| u \| .$$

Let $\mathcal{S}_m^{\mathrm{CG}}$ be generated by the conjugate gradient algorithm. Then

$$\| \mathcal{C}^{1/2}(\mathcal{S}_m^{\mathrm{CG}}(\mathcal{C}^{1/2}(v))) \| \leq \gamma_{\mathrm{CG}} \| v \|,$$

where

$$\gamma_{\mathrm{CG}} = \hat{\kappa}^{1/2} \inf_{p_m(t), p_m(0)=1} \hat{f}(m, 1/2),$$

and $\hat{f}(m, 1/2) = \sup_{t \in [0,1]} |t^{1/2} p_m(t)|$.

In this theorem $\hat{\kappa}$ represents the generalised condition number of the matrix A and the smoothing matrix, somewhat analogous to computing the condition number of the preconditioned matrix $M^{-1}A$ in a preconditioned iterative method setting (cf. equation (1.6.20) of chapter 1). In this case the condition number is computed in the subspace orthogonal to the coarse space. The notation $\mathcal{M}_j$ represents the finite dimensional subspace corresponding to the jth level of a sequence of discretisations, ie for the two grid approach $j = 1$ would represent the fine grid and $j = 0$ the coarse grid. Finally $\mathcal{S}^{\mathrm{CG}}$ and $\mathcal{C}$ represent the action of the CG smoother and the coarse grid solve respectively.

It is unclear how the coefficient γ_{CG} would be computed in a practical situation, since the polynomial p_m in (2.3.24) would depend on both the matrix A and the right-hand

side of the linear system problem. For this reason the usefulness of this theorem is limited.

We can, however, give a heuristic argument about the effectiveness, or applicability, of these methods as smoothers for the TGM. First write the error after m steps of CG as

$$\mathbf{e}_m = \mathbf{x}_m - \mathbf{x} = \sum_i c_i \mathbf{v}_i, \tag{2.3.25}$$

for some unknown c_i, which are assumed to have approximately the same order for each i.

Now, each successive step of CG minimises the error in the A-norm, given by

$$\| \mathbf{e}_m \|_A^2 = \mathbf{e}_m^T A \mathbf{e}_m = \sum_i c_i \mathbf{v}_i^T A c_i \mathbf{v}_i = \sum_i c_i^2 \lambda_i. \tag{2.3.26}$$

Clearly there is an inherent bias towards large eigenvalues in the minimisation of (2.3.26). So applying CG when the error is such that $c_i \approx c_j \ \ \forall i, j$, will correspond to damping the larger eigenvalue components more than the low eigenvalue components. This is precisely the effect we would desire from a smoother.

For the conjugate residual method the error is minimised with respect to the A^2-norm, so that (2.3.26) becomes

$$\| \mathbf{e}_m \|_{A^2}^2 = \mathbf{e}_m^T A A \mathbf{e}_m = \sum_i c_i \mathbf{v}_i^T A A c_i \mathbf{v}_i = \sum_i c_i^2 \lambda_i^2. \tag{2.3.27}$$

This potentially gives an even greater bias towards the high frequencies and so we might expect this method to be even more effective in the two grid setting.

In the simple model problem case examined in §2.3.5 we do indeed see that the conjugate residual method is marginally more effective. However, for the larger problems discussed in §2.6 the conjugate residual method is found to be no better, in terms of computational time, than the conjugate gradient method.

2.3.5 Computational Modal Analysis

To try to gain some understanding of the performance of the smoothers that we have introduced we shall now consider a discretisation of (1.2.3), over a rectangular 2D domain with constant k (hence (1.2.3) reverts to Laplace's equation, $\nabla^2 p = 0$). We construct a uniform mesh of quadrilaterals and use biquadratic basis functions such that the fine grid problem has 1056 degrees of freedom and the corresponding coarse grid problem has 289. The NAMMU code was used to generate the fine and coarse grid stiffness matrices. For a fine grid problem of this size we can compute both the exact solution, using the direct solver, and all the eigenvectors and eigenvalues of the fine grid stiffness matrix. Hence we can explicitly calculate the error components in the expansion (2.3.1). This is useful because, as was said earlier, it is not possible to carry out a modal analysis along the lines of [43, p25].

In order to see the distribution of error over the entire eigenspectrum we plot the modulus of the coefficients $\alpha_j^{(i)}$ against the eigenmode number j, though in order to improve readability and understand better the qualitative features of the results we group the error components $\alpha_j^{(i)}$ in sets of 5 and plot the maximum modulus in each set. More precisely we plot the sequence

$$\alpha_k^* = \max\{|\alpha_{5(k-1)+1}^{(i)}|, ..., |\alpha_{5k}^{(i)}|\}, \tag{2.3.28}$$

where k runs from 1 to $[N_f/5] + 1$.

In figure 2-1 we plot these maxima, on a log scale, after one coarse grid correction. The continuous line gives a qualitative idea of the distribution of the maximum error components over the entire frequency range.

To show how the iteration proceeds for each particular smoother, four TGM steps are performed, each with five smoothing steps. For the Richardson method the damping coefficient given by (2.3.3) is used. In figures 2-2 - 2-5 we compare the smoothers, each figure having the same structure, namely: writing ν as the TGM step number, on the first row the error distribution is shown after:

(a) A coarse grid solve

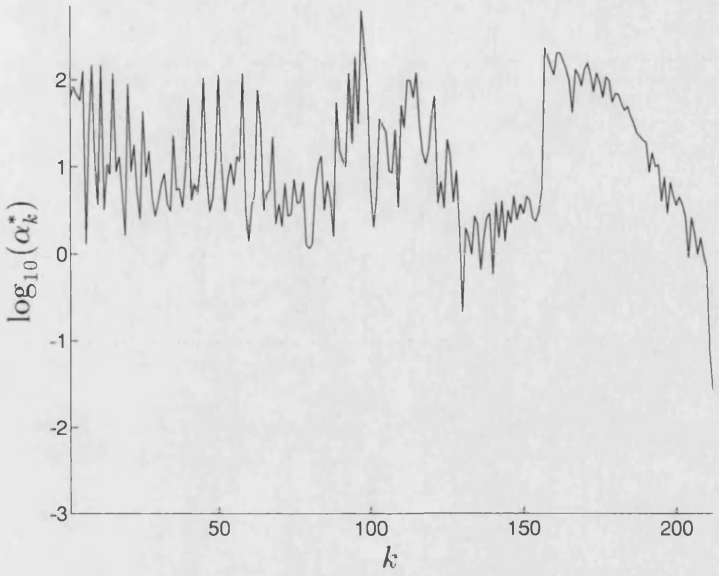

Figure 2-1: *Plot of maximum components, α_k^* (see (2.3.28)), in the error expansion (2.3.1). Note that $1 \leq k \leq 212$ corresponds to a fine grid with $N_f = 1056$.*

(b) one smoothing step ($\mu = 1$)

(c) three smoothing steps ($\mu = 3$)

(d) five smoothing steps ($\mu = 5$)

The second, third and fourth rows then show the maximum error components after two, three and four TGM steps.

As predicted, in figure 2-2 we see that the damped Richardson method is indeed successful at reducing the high frequency error components well. It is also clear from this figure that the coarse grid correction step reduces the error over the coarse grid frequencies but also redistributes the error over all the frequencies.

Figure 2-3 shows the effect of using the gradient method. Comparing this with the previous Richardson figure it can be seen that there is little difference. However, as already mentioned, this is not surprising since the gradient method has a convergence rate very similar to that of the optimally damped Richardson iteration.

From figures 2-4 and 2-5, it is clear that both CG and CR are greatly superior to either the Richardson or gradient smoother, for this simple problem at least. Also the

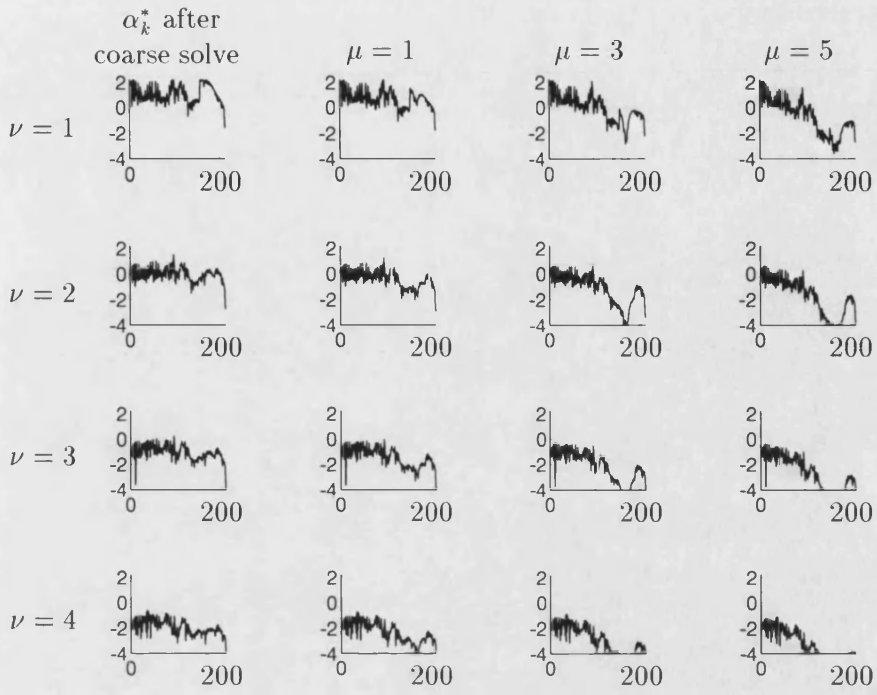

Figure 2-2: *Computational Modal Analysis with Richardson Smoothing, with axes as in figure 2-1 (so that the horizontal axis actually runs from* $k = 1, .., 212$*).*

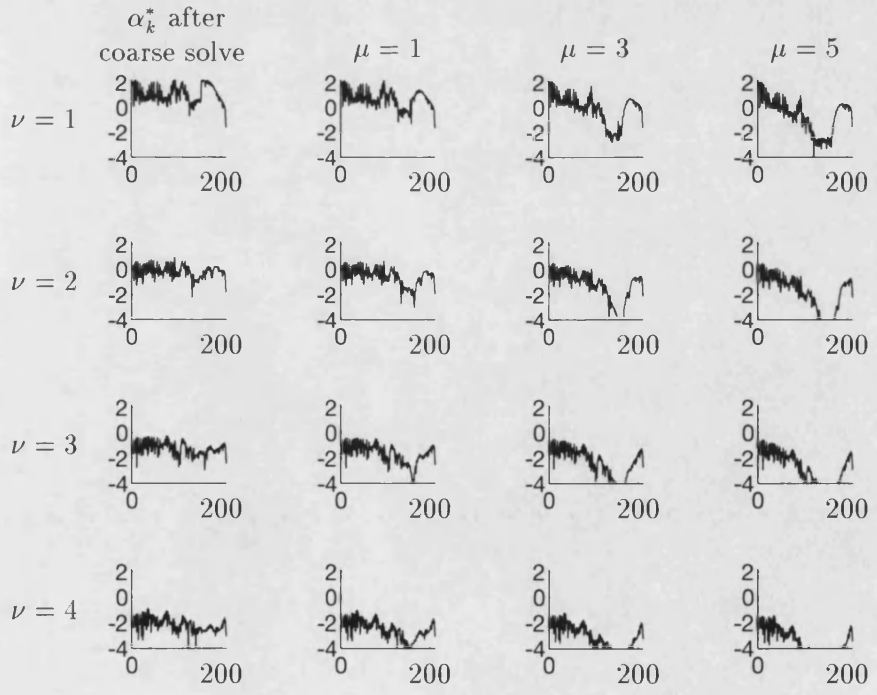

Figure 2-3: *Computational Modal Analysis with Gradient Smoothing, with axes as in figure 2-1 (so that the horizontal axis actually runs from* $k = 1, .., 212$*).*

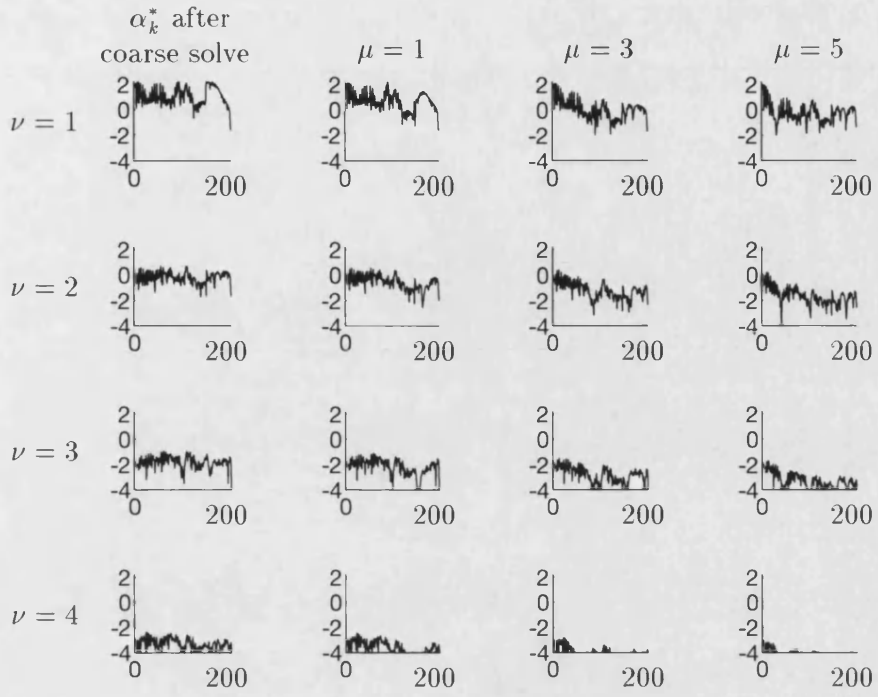

Figure 2-4: *Computational Modal Analysis with Conjugate Gradient Smoothing., with axes as in figure 2-1 (so that the horizontal axis actually runs from* $k = 1, .., 212$*).*

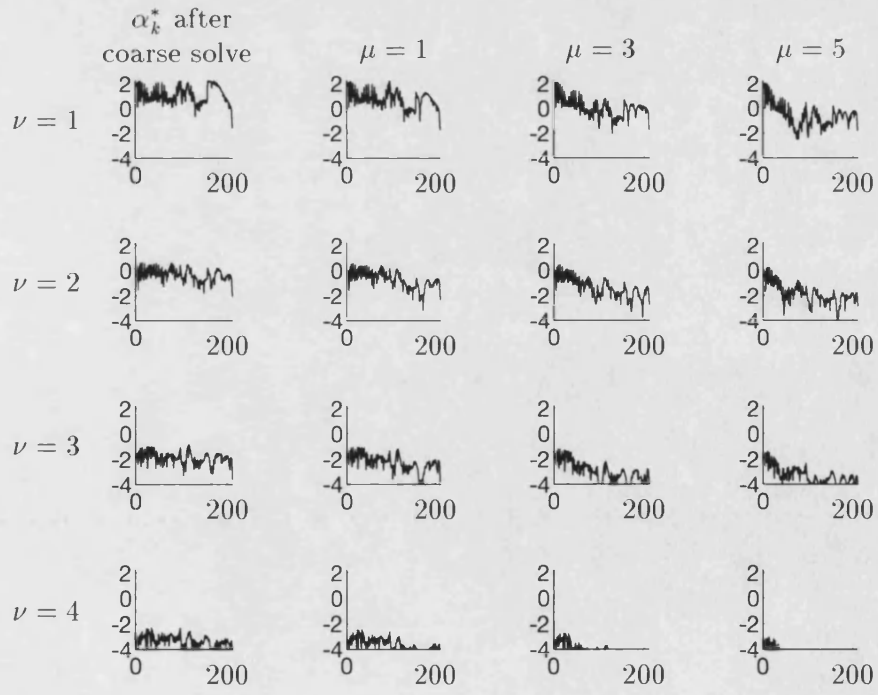

Figure 2-5: *Computational Modal Analysis with Conjugate Residual Smoothing, with axes as in figure 2-1 (so that the horizontal axis actually runs from* $k = 1, .., 212$*).*

conjugate gradient and conjugate residuals show little apparent difference. Comparison of the final few pictures indicate that the conjugate residual smoother has performed slightly better than the conjugate gradient smoother.

These results are presented purely in terms of iterations, which is potentially misleading since the CG and CR methods require more work per iteration than either the Richardson or gradient method. Later results in §2.6, on a problem from a realistic rock structure, will demonstrate that this extra work per step is well worth the effort.

For this model problem at least, whilst all the smoothers work, the CG and CR smoothers are easily the most effective, converging significantly faster than either the Richardson or gradient method. For problems with k not a constant we would reasonably expect the CG/CR methods to be even more effective than the Richardson/gradient methods, since the CG/CR methods are more adaptive in their suppression of the error components.

2.4 Preconditioned Conjugate Gradients

As discussed in chapter 1, simple diagonal preconditioners of CG fail to give an adequate rate of convergence for the physical problem discussed in this thesis. Hence a more natural idea is to use a preconditioner that encapsulates more of the features of the underlying pde. One way this can be achieved is to use the TGM as a preconditioner for the CG method. This method is quite widely used in the literature, see for example work by Kettler [63] and Braess [23].

Use of the TGM as a preconditioner can be examined by rewriting the TGM iteration from (2.3.5) as

$$W_{\mathrm{TGM}}(\mathbf{x}_{i+1} - \mathbf{x}_i) = \mathbf{b} - A_f \mathbf{x}_i, \tag{2.4.1}$$

where

$$M_{\mathrm{TGM}} = I - W_{\mathrm{TGM}}^{-1} A_f. \tag{2.4.2}$$

The matrix W_{TGM} is then the preconditioner used for the CG method. In order to ensure that the preconditioned CG method is guaranteed to converge we require that W_{TGM}

is both symmetric and positive definite. This symmetry condition can be achieved by the choice of P and R in §2.2 and by using pre- and post- smoothing steps, S_1^μ and S_2^μ constructed such that $S_1^\mu = (S_2^\mu)^T$. The TGM iteration matrix can be guaranteed to be positive definite by requiring that the two grid method converges in its own right. This follows from Hackbusch[44] where a result of the following form is given

Lemma 2.4.1 (cf. Lemma 10.7.1 of [44]) *Assume the the two grid iteration is a symmetric process. Then if the iteration converges, it converges monotonically with respect to the energy norm $\|.\|_{A_f}$, and the corresponding iteration matrix (cf. (2.4.1)) is positive definite.*

Proof See Hackbusch[44, p351]. □

The usual preconditioning step for the CG algorithm involves solving

$$M\mathbf{z}_k = \mathbf{r}_k$$

for $\mathbf{z}_k$, where M is some preconditioning matrix and $\mathbf{r}_k$ is the residual at the k^{th} step, see [38, §10.3]. For the TGM preconditioning this is replaced by applying one step of the TGM, i.e. as given in the steps (2.2.1 - 2.2.5), applied to the problem

$$A_f\mathbf{z}_k = \mathbf{r}_k$$

with an initial guess of $\mathbf{z}_k^{(0)} = 0$ (see [63]).

In our numerical experiments a single step of diagonally preconditioned Richardson's iteration is used for each of the pre- and post-smoother steps. By writing this Richardson step, for solutions of $A\mathbf{x} = \mathbf{b}$, as

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \omega D^{-1}(\mathbf{b} - A_f\mathbf{x}_i) \tag{2.4.3}$$

it is now possible to write down an exact iteration matrix for the preconditioning step such that we can later analyse the effect of this preconditioning on a model problem.

Start with

$$\mathbf{z}_k^{(0)} = \mathbf{0} \tag{2.4.4}$$

the pre-smooth step becomes

$$\mathbf{z}_k^{(1)} = \mathbf{z}_k^{(0)} + \omega D^{-1}(\mathbf{r}_k - A_f \mathbf{z}_k^{(0)}) = \omega D^{-1}\mathbf{r}_k \tag{2.4.5}$$

the coarse solve step is given by

$$\begin{aligned} \mathbf{z}_k^{(2)} &= \mathbf{z}_k^{(1)} + PA_c^{-1}R(\mathbf{r}_k - A_f \mathbf{z}_k^{(1)}) \\ &= (\omega D^{-1} + PA_c^{-1}R - \omega PA_c^{-1}RA_f D^{-1})\mathbf{r}_k \end{aligned} \tag{2.4.6}$$

and the post-smooth step is

$$\begin{aligned} \mathbf{z}_k^{(3)} &= \mathbf{z}_k^{(2)} + \omega D^{-1}(\mathbf{r}_k - A_f \mathbf{z}_k^{(2)}) \\ &= (\omega D^{-1} + PA_c^{-1}R - \omega PA_c^{-1}RA_f D^{-1} + \omega D^{-1} - \omega^2 D^{-1}A_f D^{-1} \\ &\quad -\omega D^{-1}A_f PA_c^{-1}R + \omega^2 D^{-1}A_f PA_c^{-1}RA_f D^{-1})\mathbf{r}_k. \end{aligned} \tag{2.4.7}$$

Note that $\mathbf{z}_k^{(3)} = \mathbf{z}_k$, now collecting terms in (2.4.7) gives

$$\begin{aligned} M^{-1} &= 2\omega D^{-1} + PA_c^{-1}R - \omega^2 D^{-1}A_f D^{-1} + \omega^2 D^{-1}A_f PA_c RA_f D^{-1} \\ &\quad -(\omega PA_c^{-1}RA_f D^{-1} + \omega D^{-1}A_f PA_c^{-1}R) \end{aligned} \tag{2.4.8}$$

and since $P = R^T$ and both A_f, A_c are symmetric this implies

$$(M^{-1})^T = M^{-1}$$

with positive definiteness being assured by the convergence of (2.4.3) and lemma 2.4.1. We next use the formulation (2.4.8) to analyse a simple model problem.

2.4.1 Convergence analysis

It is known that with no preconditioning conjugate gradients converges at a rate controlled by the condition number of the matrix (see §1.6.3). Using a symmetric positive definite preconditioner M then gives a method that has a convergence rate dependent on the condition number of the preconditioned system $M^{-1/2}AM^{-1/2}$, which is given

by the maximum and minimum eigenvalues of $M^{-1}A$.

For the simple 1D model problem, discussed in §2.3.1, and using the same unitary transformation ideas, it is possible to give some indication of the effectiveness of using the TGM as a preconditioner by examining the condition number, κ, of the preconditioned matrix, i.e. $\lambda_{max}(M^{-1}A)/\lambda_{min}(M^{-1}A)$. Our analysis will be based on estimating the condition number of the matrix preconditioned by a single TGM step.

Using precisely the same transformations as in §2.3.2 we obtain the same blocks for the matrices P, R, A_f and A_c, again with

$$s_t^2 = sin^2(t\pi h_f/2), \;\; c_t^2 = cos^2(t\pi h_f/2)$$

as before. Now substitute the blocks from §2.3.2 into (2.4.8), and collect terms, to give the blocks of $M^{-1}A$ as

$$(M^{-1}A)^{(t)} = \begin{bmatrix} c_t^2 + s_t^4 + s_t^4 c_t^2 & -c_t^4 s_t^2 \\ -s_t^4 c_t^2 & s_t^2 + c_t^4 + s_t^2 c_t^4 \end{bmatrix} \quad \text{for} \;\; 1 \leq t \leq N_c$$

and

$$(M^{-1}A)^{(N_c+1)} = \frac{3}{4}. \tag{2.4.9}$$

To estimate

$$\kappa(M^{-1}A) = \frac{\lambda_{max}(M^{-1}A)}{\lambda_{min}(M^{-1}A)}$$

we merely have to find the maximum and minimum eigenvalues of each of the block matrices given in (2.4.9).

By re-writing the 2x2 block matrices as

$$(M^{-1}A)^{(t)} = \begin{bmatrix} c_t^2 + (1-c_t^2)^2 + (1-c_t^2)^2 c_t^2 & -c_t^4(1-c_t^2) \\ -(1-c_t^2)^2 c_t^2 & (1-c_t^2) + c_t^4 + (1-c_t^2)c_t^4 \end{bmatrix} \tag{2.4.10}$$

it is easy to show that $(1 \;\; -1)^T$ is always an eigenvector, with corresponding eigenvalue

1. Bounds on the second eigenvalue can then be obtained from the sum of the diagonal entries of the 2x2 matrix, this is found to lie in the range $[\frac{3}{4}, 1]$. Hence the condition number of the TGM preconditioned system is given by

$$\kappa(M^{-1}A) = \frac{1}{3/4} = \frac{4}{3}. \tag{2.4.11}$$

Note that this is independent of the grid spacing parameters, h_f and h_c.

2.4.2 General convergence

Using the combination of conjugate gradients and TGM can be equivalently considered as either *a TGM preconditioned conjugate gradients* or *a conjugate gradient accelerated TGM*. In the latter form the convergence properties of this method are considered by Hackbusch[44, §10.8.3]. Assume for the TGM we have a convergence rate given by

$$\sigma(M_{\text{TGM}}) \leq \gamma < 1. \tag{2.4.12}$$

From the formulation (2.4.1) and equation (2.4.2) we relate the convergence rate of the TGM, i.e. γ, to the eigenspectrum of the preconditioned system, i.e. $W_{\text{TGM}}^{-1}A_f$, by using equation (2.4.12). This gives

$$1 - \gamma \leq \sigma(W_{\text{TGM}}^{-1}A_f) \leq 1. \tag{2.4.13}$$

Hence the condition number of the matrix A_f preconditioned by the two grid iteration, is given by $\kappa(W_{\text{TGM}}^{-1}) = \frac{1}{1-\gamma}$. Since k steps of the CG algorithm give a reduction in error of at least $2[(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)]^k$, then the overall convergence rate is given by

$$2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k = 2\gamma^k/\left(1+\sqrt{1-\gamma}\right)^{2k} \approx 2\left(\frac{\gamma}{4}+O(\gamma^2)\right)^k. \tag{2.4.14}$$

On the assumption that γ is small, it is clear that the convergence rate of the resulting method will be at least 4 times faster than the original multigrid method. This result is, to some extent, borne out by the numerical results presented later in figure 2-15 in §2.6.1.

In §2.6 we will present results for the use of TGM to precondition CG and compare with the TGM method discussed in §2.2.

2.5 A three grid method

A natural question to ask is "would a three grid method perform better than the TGM in these examples?" Despite the restriction on coarsening as discussed in §2.1, in our problems it is possible that three grids, fine, middle and coarse, can be constructed, giving 3 stiffness matrices denoted A_f, A_m, A_c. In this case it then becomes possible to produce a *three grid* method, which will henceforth be referred to as 3GM. We now need two prolongation and restriction matrices, denoted R^f, P^f for transfers between the fine and middle grids and R^c, P^c for middle to coarse grid transfers.

The extension from two to three grids is straightforward, we replace the original coarse grid correction step, i.e. (2.2.3), with a smoothing step on the middle grid, using A_m, and a coarse grid correction step using A_c. This gives a method which is often called a V-cycle, denoting the fact that we move down through the grids to the coarsest level and then back up again to the finest level. Note that an alternative implementation is possible, where we perform a coarse grid correction, smooth on the middle grid, and coarse grid correct again before transferring back to the fine grid. This version is commonly called a W-cycle.

For the implementation used here, the same smoother will be used on the middle and fine grids, along with the same number of smoother steps per three grid iteration. The coarse grid solve is again performed exactly, using pre-calculated LU factors.

2.6 Numerical Results

For the results presented here we consider a 2D groundwater flow problem for complex geological mediums with realistic contrasts in the permeability field. Results being given for the TGM with the various smoothers considered in §2.3, along with comparisons with the two grid preconditioned conjugate gradient algorithm of §2.4, and finally for

the 3GM of §2.5. More details, including pictures of the coarse grid, with $N_c \approx 3500$, are given in chapter 1, figure 1-1, with the permeability regions being shown in figure 1-2. Actual permeability coefficients for this grid are given in table 1.1.

2.6.1 Results for 2D example

Two combinations of grids are considered to compare the effectiveness of the four choices of smoothers from §2.3 and the TGM preconditioned conjugate gradient method of §2.4.

Problem 2.1 *Consider a 2D fine grid mesh with* $N_f = 58944$, *the coarse grid was obtained by roughly halving the number of elements in each spatial direction, resulting in* $N_c = 15021 \approx N_f/4$.

A second discretisation with an even coarser grid was also set up.

Problem 2.2 *Consider a 2D fine grid mesh with* $N_f = 58944$, *the coarse grid was obtained by essentially quartering the number of elements in each spatial direction, resulting in* $N_c = 3827 \approx N_f/16$.

Note that we expect problem 2.2 to require more overall smoother steps to produce a solution of the same accuracy, since the coarse solution will not represent the fine solution as well as in problem 2.1. This is confirmed by the results in figure 2-8. Further note that the coarsening used in problem 2.1 represents the approach commonly used in the literature.

All the results presented here were run on a Cray YMP. In all cases the methods are compared to the direct frontal solver, the Harwell MA32 code, which requires 45 seconds to produce a solution to the fine grid problem to the level of machine accuracy. This solution is assumed "exact" and will be used to estimate the error for the iterative methods. Where possible this direct solution time will be indicated on the graphs by a "∇", (see figure 2-6). The code MA32 was also used to find the LU factors of A_c that are needed for the coarse grid solves (see (2.2.3)). For $N_c = 15021$, which is the case in problem 2.1, the cost of this one-off factorisation was only 2.69 seconds. For all the runs

an upper limit of approximately 1400 applications of smoother steps was set, by which we mean one step of Richardson, Jacobi, CG or CR. Inside of this limit convergence was reasonably expected, although at this limit considerably more CPU time would have been used than for the frontal method.

The exact solution on the fine grid is available, which means that a stopping criteria could be based on a reduction in error. However, we prefer to choose a convergence criterion more appropriate to an iterative method where the solution is unknown. To this end our convergence criterion requires that

$$\frac{\| \mathbf{r}_i \|_{D^{-1}}}{\| \mathbf{r}_0 \|_{D^{-1}}} < 10^{-7} \tag{2.6.1}$$

where $\mathbf{r}_i = \mathbf{b} - A_f \mathbf{x}_i$, and D is the diagonal of the stiffness matrix A. This is one of the simplest choices, particularly for the CG-like methods where the quantity

$$\| \mathbf{r}_i \|_{D^{-1}} = \mathbf{r}_i^T D^{-1} \mathbf{r}_i \tag{2.6.2}$$

is computed as a direct process of the algorithm. (For the purposes of our graphs we will in fact plot the actual solution error, since this is available for our test problems). Note that the choice (2.6.2) of convergence test is superior to just using a relative residual convergence test, as is now discussed. Standard error analysis, see for example [7], gives the following

$$\frac{\| \mathbf{e}_i \|}{\| \mathbf{x} \|} \leq \kappa(A) \frac{\| \mathbf{r}_i \|}{\| \mathbf{r}_0 \|} \tag{2.6.3}$$

$$\frac{\| \mathbf{e}_i \|_A}{\| \mathbf{x} \|_A} \leq \kappa^{\frac{1}{2}}(D^{-1/2} A D^{-1/2}) \frac{\| \mathbf{r}_i \|_{D^{-1}}}{\| \mathbf{r}_0 \|_{D^{-1}}} \tag{2.6.4}$$

$$\frac{\| \mathbf{e}_i \|}{\| \mathbf{x} \|} \leq \kappa^{\frac{1}{2}}(A) \frac{\| \mathbf{e}_i \|_A}{\| \mathbf{x} \|_A}. \tag{2.6.5}$$

Now combining (2.6.4) and (2.6.5) gives

$$\frac{\| \mathbf{e}_i \|}{\| \mathbf{x} \|} \leq \kappa^{\frac{1}{2}}(A) \kappa^{\frac{1}{2}}(D^{-1} A) \frac{\| \mathbf{r}_i \|_{D^{-1}}}{\| \mathbf{r}_0 \|_{D^{-1}}}. \tag{2.6.6}$$

In addition a result in [35], shows that, for symmetric positive definite matrices A, with diagonal D,

$$\kappa(D^{-1/2} A D^{-1/2}) \leq \kappa(A). \tag{2.6.7}$$

If we compare (2.6.3) with (2.6.6) and bear in mind (2.6.7) we would expect our stopping test (2.6.1) to be at least as good as a test based on (2.6.3), and, depending on the reduction in the condition number after diagonally scaling, potentially significantly better.

From a practical point of view we would wish to stop when the relative error, i.e.

$$\| \mathbf{e}_i \| / \| \mathbf{e}_0 \|,$$

had been reduced to a level comparable with the error in the FEM discretisation, see §1.2.3. In all the cases presented here that converged, this level of error was in fact reached.

Results for the Smoothers

For each of the smoothers we present results for three different values μ and compare the overall solution time. Before presenting the results, we should point out that since the maximum eigenvalue of the fine grid matrix, A_f, is not known a priori, the damping coefficient ω used for the Richardson method (cf. §2.3 is taken as simply $2/\lambda_G$, where λ_G is a Gershgorin estimate of the largest eigenvalue of the diagonally scaled matrix A_f. Note that this choice is sufficient to guarantee that the Richardson method, and hence the two grid method, will converge, since Richardson is known to converge for all damping coefficients in the range $0 < \omega < (2/\lambda_{max})$ (see Hackbusch [44, pp82-83]).

Consider first the results for problem 2.1. The first thing to point out about the Richardson and gradient method smoothers is that they failed to reach the convergence criterion in all cases. In fact, as can be seen from figures 2-6 and 2-7, after an initial promising start, subsequent convergence of the error was very slow. It is, however, of some interest to note that the gradient method version shows very little dependence on the number of smoother steps performed prior to a coarse grid solve.

The results for the conjugate gradient method in figure 2-8 are much more satisfactory, the convergence criterion was met in all cases, moreover convergence was achieved in less than one third of the CPU time required for the direct solve for almost all of the

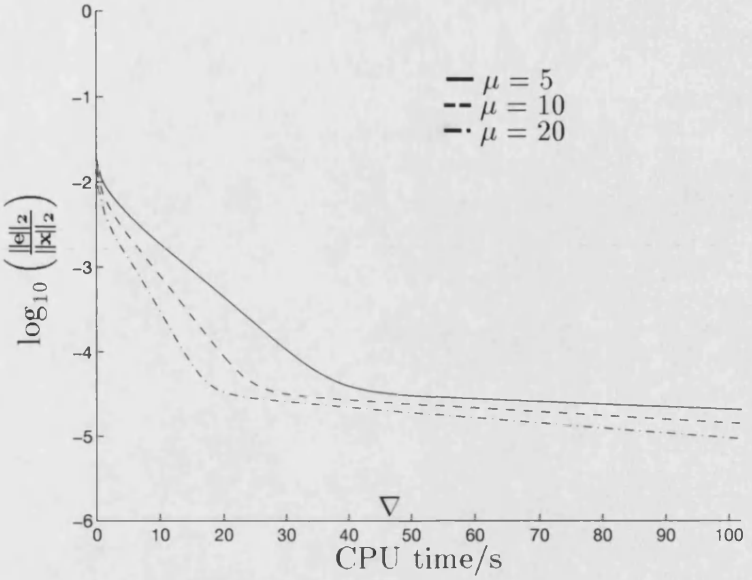

Figure 2-6: *Problem 2.1(2D): Richardson as smoother.*

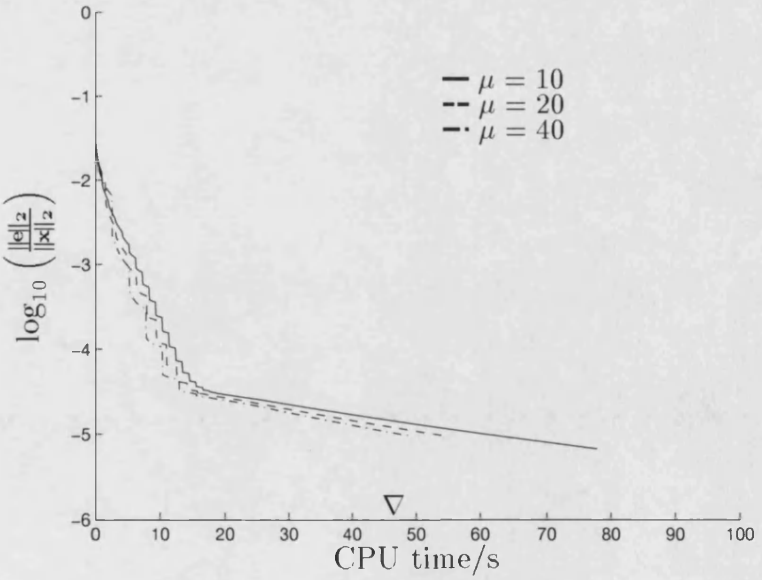

Figure 2-7: *Problem 2.1(2D): Gradient method as smoother.*

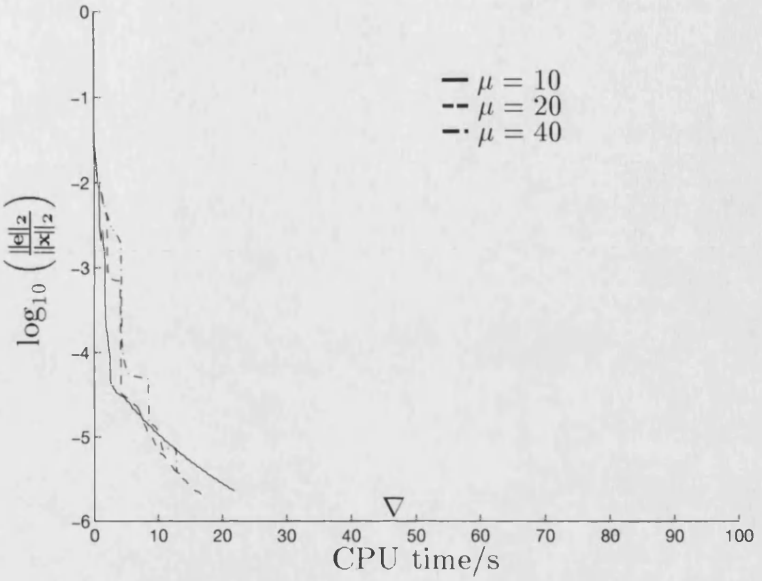

Figure 2-8: *Problem 2.1(2D): Conjugate gradient method as smoother.*

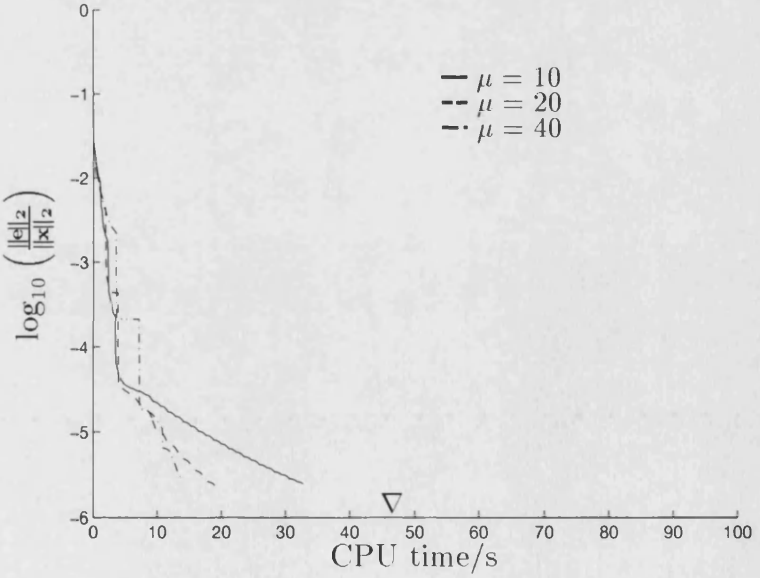

Figure 2-9: *Problem 2.1(2D): Conjugate residual method as smoother.*

number of smoother steps tested. In addition we see a marked difference in performance depending on the number of smoother steps used, the $\mu = 10$ case takes nearly 30% more time than the $\mu = 20$ case, while with $\mu = 40$ convergence is faster again. It is also far more apparent from these graphs at which point the coarse grid corrections occur, and the associated improvement that they give to the solution.

In a similar way to the conjugate gradients case, the conjugate residuals used as a smoother, are seen to be effective, see figure 2-9. In addition the CR smoothers also show an improved overall convergence rate for larger numbers of smoothing steps. However, comparisons of these two methods show little apparent difference in convergence rates.

In figure 2-10 the "best" results, in terms of convergence rate, from each of the four methods are compared. It is clear that the CG and CR smoothers are significantly better than either of the Richardson or gradient smoothers, despite the fact that they require more work per iteration. The indication is that relatively simple iterative methods are incapable of solving these problems efficiently, and since the more complicated Gauss-Seidel, SOR etc have already been ruled out, see §2.1, methods based on CG would seem to be the natural choice. Finally note that the gradient method performs better than the Richardson iteration, which is perhaps expected, since in our formulations the gradient method is essentially an "optimally" damped version of the Richardson iteration.

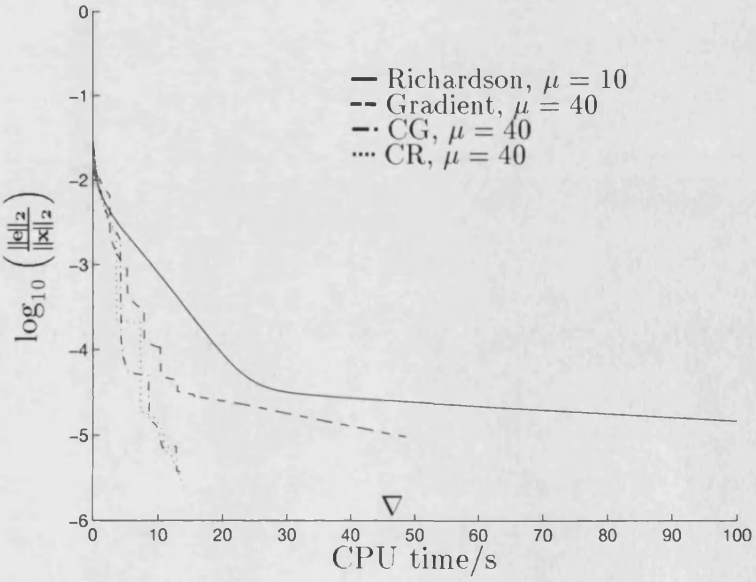

Figure 2-10: *Problem 2.1(2D): "Best" results from each smoother.*

Next consider the CG smoother for problem 2.2. In figure 2-11 we compare the results for CG with $\mu = 10, 20$ and 40. Again the performance with $\mu = 10$ is very poor, indeed in this case it now takes more time than the direct solver to converge. As for problem 2.1 it is again found that $\mu = 40$ gives the best performance.

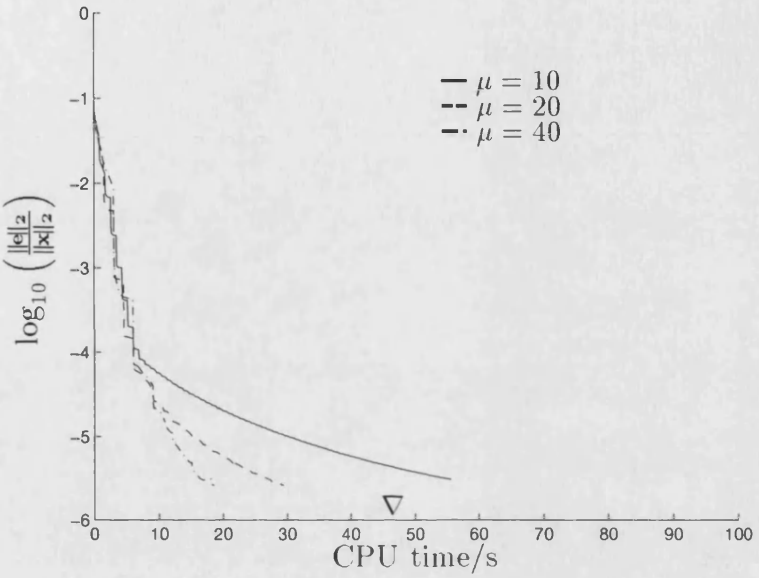

Figure 2-11: *Problem 2.2(2D): Conjugate Gradients as Smoother.*

As a final comparison, working again with problem 2.1, we compare the error reduction versus iterations, by which it is meant the number of smoother iterations, a comparison of these results with those in figure 2-10 show very little difference, despite the fact that both the conjugate gradient and conjugate residual algorithms require significantly more work, in terms of scalar products, than the other methods. This is an indicator of how much the matrix-vector multiplications dominate the overall work.

For all four of the smoothers considered here it is by no means obvious how to choose μ, the number of smoother steps per TGM step, for optimal convergence. Certainly for the CG and CR cases there is a marked difference in performance between the smallest and largest numbers. From the results presented here it would seem that choosing a large number ($\mu = 40$) is best.

Finally, to illustrate a point made earlier in §2.1, that diagonal scaling essentially removes the ill-conditioning due to variations in k, we compare the results for two related problems. The first is just problem 2.1, whilst the second uses the same grid as in problem 2.1, but here the permeability coefficient is set to a constant over the entire

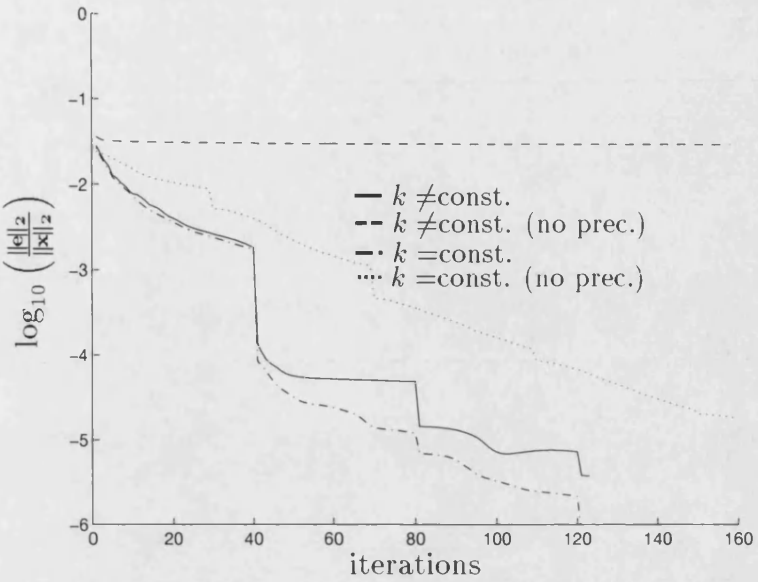

Figure 2-12: *Comparison of CG smoothing ($\mu = 40$) on Problem 2.1(2D) and CG smoothing on Problem 2.1(2D) with k constant on entire domain. Unless otherwise stated diagonal preconditioning is used for the smoothing steps.*

region. For both these problems TGM with 40 steps of CG smoothing are used to obtain a solution. Results in figure 2-12 show relatively little difference in convergence of the two problems, as was expected. Note that, for completeness, this graph also shows the convergence of both problems, with no diagonal preconditioning used. It is also clear from this that use of diagonal preconditioning for the smoother is vital to the convergence of the overall method.

TGM preconditioning (cf. §2.4)

First consider problem 2.1. For comparison purposes the results for the best CG smoothed TGM are also plotted, see figure 2-13. The CG smoothed code is clearly more effective, although it should be borne in mind that this is for the "best" choice of number of smoothing steps, which is not something that can be predicted prior to running the code. For comparison purposes again recall that the direct solver takes 45 seconds to solve the problem, and a ∇ is used to denote this direct solver time on the graphs.

In figure 2-14 the results for problem 2.2, which has the less refined coarse grid, are given. Again we find that the CG smoothed TGM algorithm is far superior.

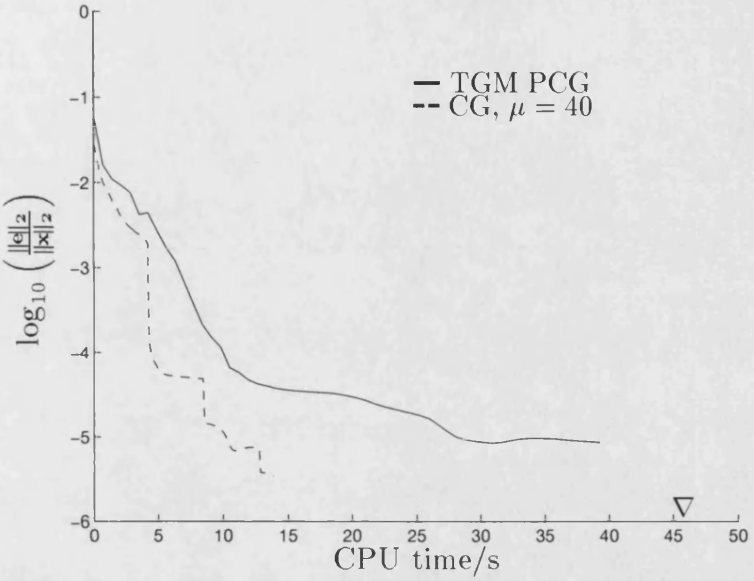

Figure 2-13: *Problem 2.1(2D): Comparison of TGM preconditioning and CG smoothing.*

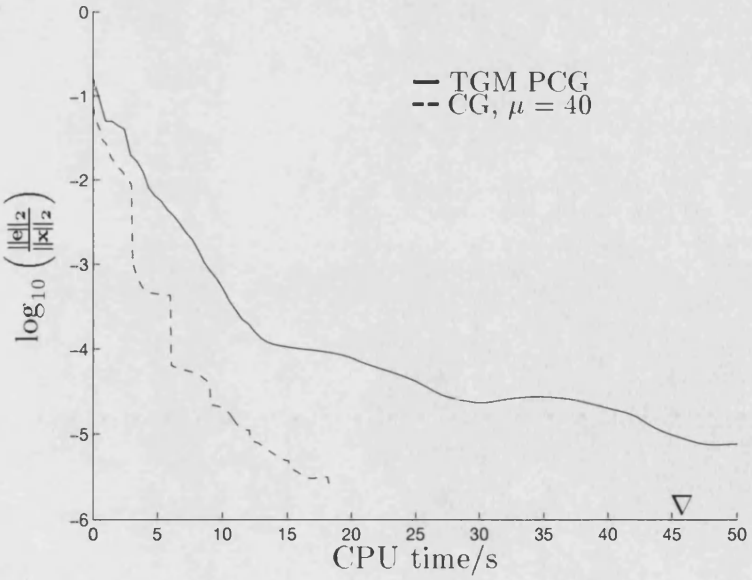

Figure 2-14: *Problem 2.2(2D): Comparison of TGM preconditioning and CG smoothing.*

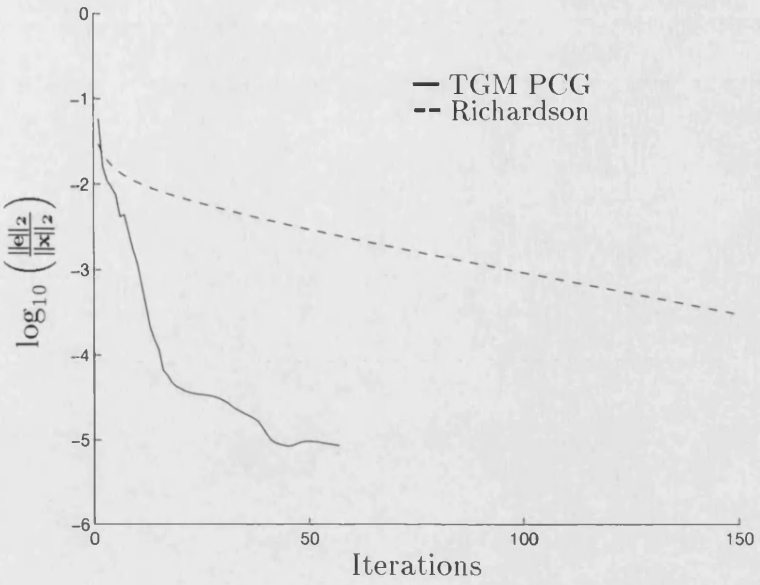

Figure 2-15: *Problem 2.1(2D): Comparison of TGM preconditioned CG and Richardson smoothing.*

It should be noted that for both these examples the convergence does become quite slow in the later stages, perhaps indicating that this method is not so robust as the CG smoothed TGM.

Finally in figure 2-15 a comparison of Richardson smoothed TGM, together with CG preconditioned by the same Richardson smoothed TGM is made. In this cases one step of pre-smoothing and one step of post-smoothing is used. The figure shows a plot of error versus iteration count and we do indeed see a speedup of at least 1/4 as predicted by the result from Hackbusch[44, §10.8.3].

2.6.2 A three grid method

We finally consider the performance of the 3GM on the 2D problem. As for the TGM, performance of the method is tested for a number of different smoother steps. The following combination of grids is used to test the code.

Problem 2.3 *Consider a 2D fine grid mesh with* $N_f = 58944$*, the middle and coarse grids were obtained by essentially halving the number of elements in each spatial direction, resulting in* $N_m = 15021 \approx N_f/4$ *and* $N_c = 3827 \approx N_f/16$.

For the results considered in this section, CG is used as a smoother both on the fine and middle grid. In addition the same number of smoothing steps is used on each of the grids.

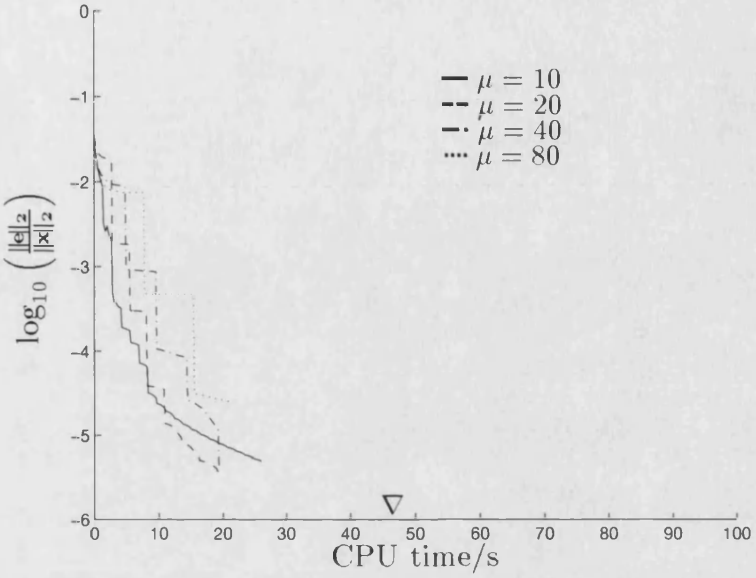

Figure 2-16: *Problem 2.3(2D): 3GM, with conjugate gradient smoother.*

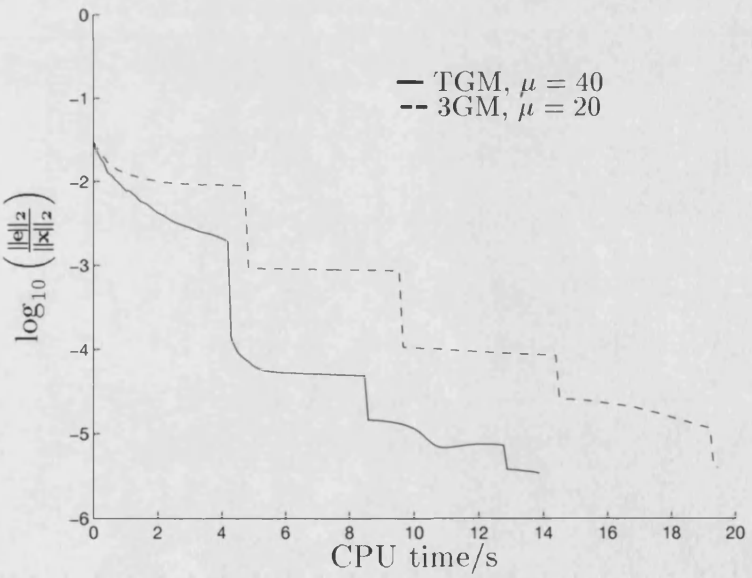

Figure 2-17: *Problem 2.3(2D): Comparison of TGM and 3GM, with conjugate gradient smoother.*

In figure 2-16 we compare 10,20,40 and 80 steps of CG smoothing. It is clear that, as in the two grid case, each of these cases converges in less time than the direct solver requires to solve the problem exactly. For this particular problem 20 steps seems to give the best convergence.

In figure 2-17 we compare the convergence of the TGM and the 3GM. The TGM is clearly faster than the 3GM, converging in 74% of the time required by the 3GM. However, the 3GM shows less of a tendency to "tail-off" at the later stages of convergence, this is clearly an important issue. It is possible that there are problems for which the TGM stops converging at all, but the 3GM can still solve well.

2.7 A non-symmetric extension

Finally in this chapter we very briefly examine a modification of the two grid approach to a non-symmetric problem. This arises from a problem first mentioned in §1.2, which involves modelling radionuclude transport in a saturated medium. Unlike the previous case of groundwater flow in a saturated medium this does not produce a symmetric linear system, as we now indicate.

In the steady-state case the nuclide concentration, N_α, is modelled by the following equation

$$\nabla.(\mathbf{q}N_\alpha) - \nabla.(\phi D_\alpha \nabla N_\alpha) = -\lambda_\alpha \phi R_\alpha N_\alpha + \lambda_{\alpha-1}\phi N_{\alpha-1} + \phi f_\alpha. \tag{2.7.1}$$

Here R_α and λ_α are constants, ϕ and f_α are functions of position, and D_α is a function of position and the Darcy velocity $\mathbf{q}$. In a normal situation the Darcy velocity $\mathbf{q}$ will be obtained after an initial steady-state calculation of the pressure, using the groundwater flow model discussed in §1.2, (c.f equations (1.2.2) and (1.2.3)). Note that solutions of the pressure equation form the primary goal of the work in this thesis.

A finite element discretisation of equation (2.7.1) would give rise to a linear system, with a non-symmetric stiffness matrix because of the $\nabla.(\mathbf{q}N_\alpha)$ term. It is clear that, apart from the smoother we use, the two grid method does not rely on the stiffness matrix A being symmetric, and so the coarse grid correction step, as outlined in §2.2, would proceed exactly as before. Note that the techniques for computing the prolongation and restriction are unchanged, since (2.7.1) is linear.

In order to use a two grid approach to solve the linear system we need to consider a smoother appropriate for non-symmetric matrices, with preferably the high frequency

damping properties that we desire from a smoother. The first option is to apply CG to the normal equations, that is

$$A^T A\mathbf{x} = A^T\mathbf{b}.$$

Clearly this linear system is now symmetric, so the convergence properties of CG now apply. Based on our experience in the symmetric case we would again advocate the use of diagonal scaling as a preconditioner. Hence by writing D as the digaonal of A, we therefore apply CG to the problem

$$D^{-1}A^T D^{-1}A\mathbf{x} = D^{-1}A^T D^{-1}\mathbf{b}.$$

It is well known that using CG on the normal equations may not give very satistactory results, since the condition number of $A^T A$ can be very large. However, as we have seen in the symmetric case, it is not a strict requirement that the smoother converges very well in order for the two grid method to be effective. Further options would be to consider other "matrix-free" iterative methods that can be applied to a non-symmetric system and use them as a smoother. For example Bi-CG [34], or GMRES [80], although clearly CGN is one of the simplest methods.

2.7.1 Results

We briefly test the two grid method with CG on the normal equations (CGN) for a very simple model problem, arising from (2.7.1), on a uniform mesh. We test the TGM with both 20 and 40 smoother steps per two grid iteration. This problem features 3600 degrees of freedom, and as usual the coarse grid is obtained by halving the number of elements in each spatial direction, resulting in $N_c \approx 900$. For comparison purposes we also show results for both the Bi-CG method of Fletcher[34] and CGN, where these iterative methods are applied as solutions techniques in their own right. At this stage, since we are only testing small problems, the interest lies in whether the overall method works effectively, hence we present the results in terms of iterations. From the experience in the symmetric problems, it is clear that much larger problems than these would be needed to see the iterative method taking less CPU time than the frontal method.

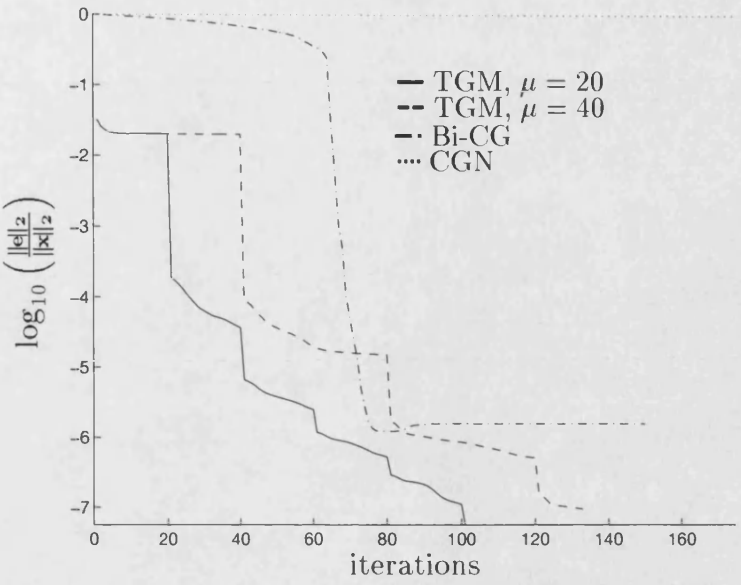

Figure 2-18: *Simple nonsymmetric model problem with TGM (with a CGN smoother), Bi-CG and CGN.*

In figure 2-18 we compare TGM, Bi-CG and CGN. It is clear that the TGM, with the smaller number of smoother steps has the fastest convergence rate. Although CGN does not appear to converge at all, it does in fact finally converge, but only after nearly 3000 iterations. Bi-CG would appear to competitive with TGM, although the convergence does tail off dramatically at the end.

We finally note that a small number of experiments, again using TGM with the CGN smoother, were performed on a larger, more difficult problem, arising in this case from the Jacobian formulated from applying Newton's method to a nonlinear pde. It was found that the convergence of the two grid method was very slow, although it was unclear exactly what the effect of the Newton iterations was on the overall method. In this case it is believed that a more effective smoother is needed to solve the problem, perhaps Bi-CG or GMRES.

2.8 Conclusions

For FEM discretisations of (1.2.3), where the grid is highly non-uniform and the coefficients k are discontinuous, the two grid approach with conjugate gradients as a smoother is an effective method. For the 2D problem a large number of smoother steps, would

seem to be the best choice.

For the large 2D problem considered here, with nearly 60,000 degrees of freedom and run on a Cray YMP, the two grid method with a conjugate gradient smoother gives a solution in less than 30% of the time taken by the MA32 direct solver (over 3 times faster). In addition using a two grid method to precondition a conjugate gradient method, whilst giving a method that converges, uses about twice as much CPU time as the standard two grid method, for the 2D problem tested here.

The three grid method was found to be no better than the two grid method, at least for the problem considered here. In fact the TGM took only 74% of the time the 3GM required to solve the 2D problem.

The two grid method was also applied to a small (approximately 3600 degrees of freedom) non-symmetric linear system, arising from the finite element discretisation of a radionuclide problem. Conjugate gradients applied to the normal equations was used as a smoother in this case and for this problem with resulting two grid method would appear to be effective. However, for a less trival problem the results are not so promising, perhaps indicating that more effective smoothers are required.

Chapter 3

Two grid method on a 3D problem

3.1 Introduction

In this chapter the two grid method(TGM), and the three grid method(3GM), discussed in chapter 2 are applied to a 3D problem.

The main elements of the theory from chapter 2 apply directly to this problem, apart from some of the eigenvalue considerations in §2.3, so essentially we shall merely present the results for a 3D problem.

It is, however, important to note that the 3D problem case does differ from the 2D case in a number of aspects. Consider the case where we denote the number of elements in a single spatial direction by n. Firstly the scale of the problem is very different. In 2D it is generally possible to make highly refined grids that model the physical regions very well, without requiring a prohibitive number of freedoms, since the size of the corresponding matrix problem is $O(n^2)$. For the same size region, but now in 3D, we would require $O(n^3)$ freedoms for the same level of discretisation. The effect is then either to force the model to be less refined or greatly increase the number of freedoms. As an example of this consider that $n \approx 240$ in the 2D case from chapter 2, whereas $n \approx 56$ in the 3D

case in this chapter.

The second issue concerns the use of the frontal solver method, for the results in this chapter the MA42 version is used. As discussed in chapter 1, the frontal solver work is $O(n^7)$ for 3D problems (in 2D it is only $O(n^4)$). Clearly the problem size does not need to be very big before a large amount of CPU time is needed to solve the problem exactly (see the "thought experiment" on this in §1.4.2). For our 2D problems we have been discussing CPU time on the scale of a minute; for the 3D problem in this chapter nearly an hour is needed to solve directly. Under some assumptions on the rate of convergence of the iterative method, it is not unreasonable to argue that the iterative method requires $O(n^3)$ operations in 2D and $O(n^4)$ in 3D, so that the "advantage" in 3D is $O(n^3)$, which is considerable higher than the advantage of only $O(n)$ in 2D. Hence we would expect the iterative method to be considerably faster than the direct solver in the 3D case.

3.2 Results on a 3D problem

The 3D problem discussed in this section arises from the Gorleben site in Germany, discussed in more detail in [22]. Various cross sections of a coarse grid for this problem are given in chapter 1 in figures 1-3 - 1-5. Note that these figures also show the four regions of rock types, these have permeabilities ranging from 10^{-5} to 10^{-10}.

The problem being solved is

Problem 3.1 *Consider a 3D fine grid mesh with* $N_f = 172254$. *The coarse grid was obtained by roughly halving the number of elements in each of the three spatial directions, resulting in* $N_c = 22667 \approx N_f/8$.

As in the 2D case in chapter 2, the results are again obtained using a Cray YMP, and are compared against the direct solver. In this case the direct solver requires 3525 seconds, nearly an hour, to solve the problem exactly. In order to factorise the coarse grid problem into an LU form, we now require 47.56 seconds of CPU time. Note that for these results only the TGM with the CG smoother is considered. As has already been

seen in the 2D results, the CG method is far superior to the Richardson and gradient methods, and also outperforms the CR method.

Once again, the exact solution is available to determine a stopping criterion, but again we use a criterion based on reduction in relative preconditioned residual, as in equation (2.6.1), although this time a slightly higher criterion of 10^{-8} is used. Since both the TGM and 3GM are considerably faster than the direct solver for the 3D problem it is not feasible to mark the direct solver time on the graphs we present in this section.

3.2.1 TGM on the 3D example

We first present results for the TGM for μ between 20 and 320 in figures 3-1 and 3-2. It is clear that, in a similar way to the 2D case, a relatively large number of smoother steps, $\mu = 160$, gives the fastest rate of convergence, in terms of CPU time. Indeed, the $\mu = 160$ case converges in less than 8% of the time the direct solver requires, a considerable saving of CPU time.

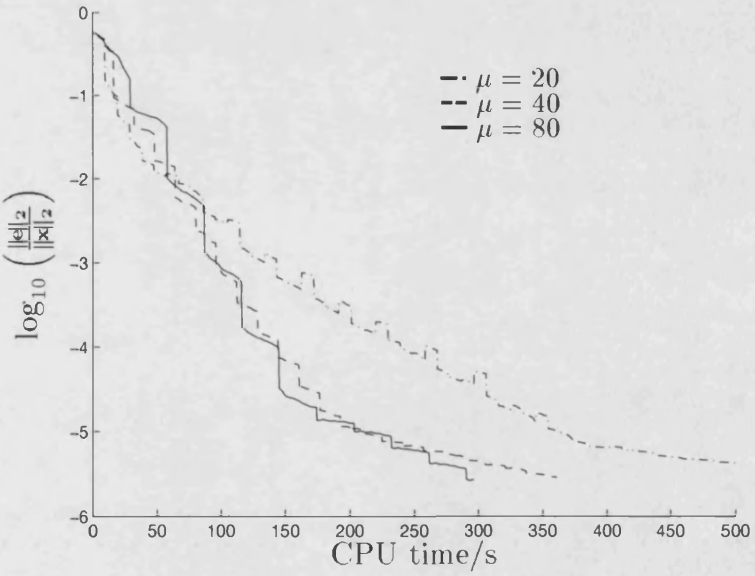

Figure 3-1: *Problem 3.1: TGM results for CG smoothing on a 3D problem.*

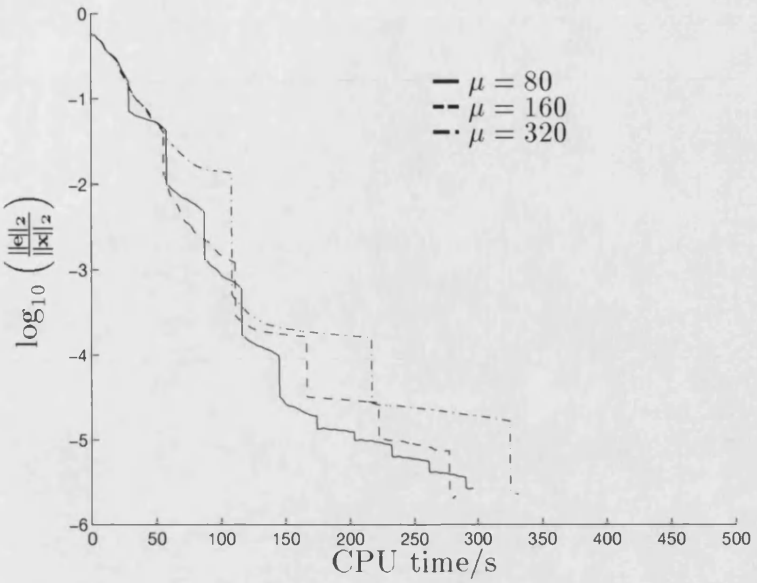

Figure 3-2: *Problem 3.1: TGM results for CG smoothing on a 3D problem.*

3.2.2 Different level of coarsening

A further question that we could address is related to the issue of coarsening. In the 2D case we coarsened such that the coarse grid had 1/4 of the freedoms that the fine grid had. A natural question might be to try coarsening in 3D such that we again have this balance between fine and coarse grids. In the following we will use the notation (a, b, c), to represent the coarsening used to generate the coarse grid, where a, b and c denote the coarsening factor in the x, y and z direction respectively. In this notation the situation for problem 3.1 would be denoted by $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.

In figure 3-3 this approach for the coarse grid is considered, comparing results for the original coarse grid, and grids that are coarsened in only in the y and z direction, and only the x and y direction. It is clear that this has some considerable effect on the convergence, and perhaps by carefully choosing the coarsening directions a faster converging method could be produced. Our choices of directions for the coarsening were quite arbitrary, and use no knowledge of the physics of the situation, so in a sense are very naive choices. In a practical situation we would invisage a coarse grid being set up and then refined in perhaps the areas needing higher accuracy. This of course would depend highly on the physics of the regions.

Finally, note that for these choices factorising the coarse grid problem now takes nearly

5 times longer than the choices discussed in the previous section.

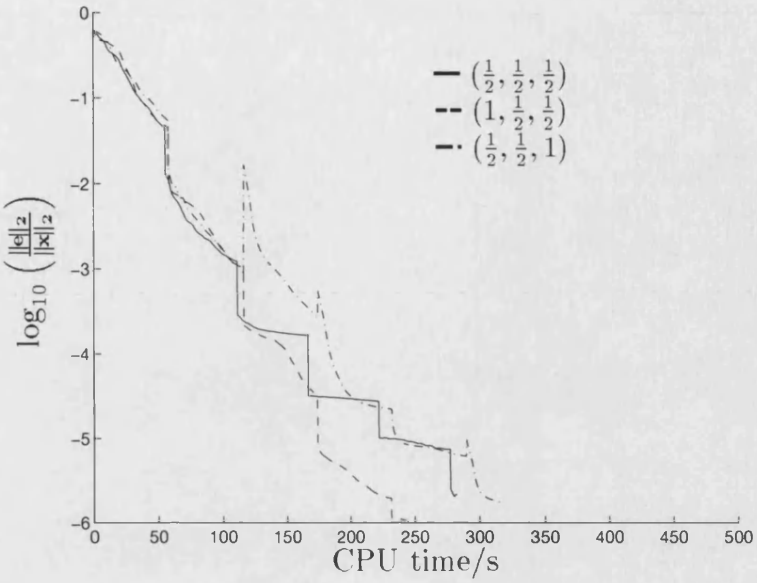

Figure 3-3: *Problem 3.1: TGM results for CG smoothing, with* $\mu = 160$, *on a 3D problem, with different levels of refinement on the coarse grid.*

3.2.3 3GM results

We finally consider the performance of the 3GM, discussed in chapter 2, on the 3D problem. As in the two grid case, performance of the code is tested for a number of different smoother steps. The following combination of grids is used to test the code.

Problem 3.2 *Consider a 3D fine grid mesh with* $N_f = 172254$. *The medium grid was obtained by roughly halving the number of elements in each of the three spatial directions, resulting in* $N_m = 22667 \approx N_f/8$. *The coarse grid was obtained from halving again to obtain* $N_c = 3126 \approx N_m/8$.

In figure 3-4 the 3GM is tested with a range of numbers of smoothing steps. As in the two grid case using the higher number of smoothing steps, i.e. $\mu = 80$ or $\mu = 160$, gives best convergence rates.

Figure 3-5 compares the fastest convergence for the TGM with the fastest convergence for the 3GM. The TGM is clearly faster, although the 3GM appears to have a more uniform convergence rate. For another problem this may be a significant issue, we might

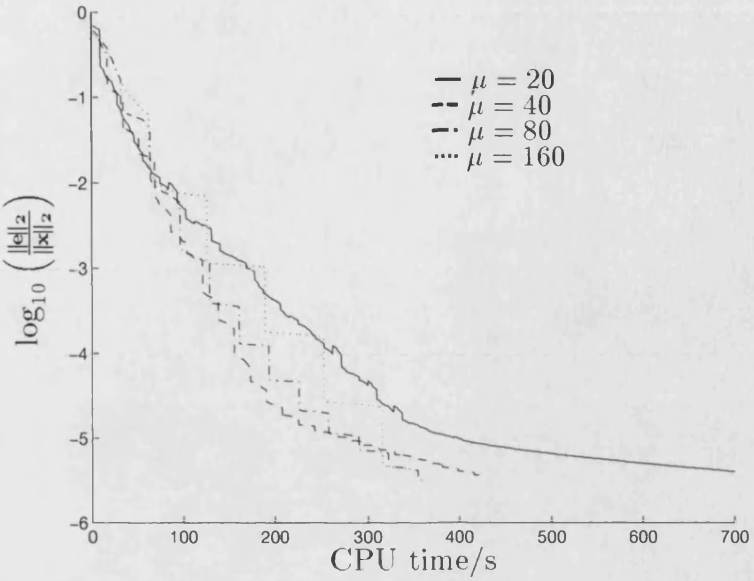

Figure 3-4: *Problem 3.2: 3GM results with CG smoothing, with a varying number of smoother steps.*

perhaps conceive of a case where the TGM converges to some level of accuracy and then stops converging, but the 3GM is able to converge to a higher level of accuracy.

Finally we compare the convergence of the 3GM for grids with different levels of refinement for the middle and coarse grids. The notation used is as in §3.2.2. In figure 3-6 we compare the convergence for 3 sets of grids, each using 160 smoother steps. As in the TGM case for this approach it is clear that this can have a marked affect on the convergence rate, although in this case there is no advantage. Again a more careful selection of coarse grid, using the underlying physics of the problem could have a marked effect on the convergence rate.

3.3 Prolongation calculation

We finally comment on the calculation of the prolongation and restriction matrices, P and R. As was seen in chapter 2, this involves calculating the quantity

$$\Phi_j(x_i) \quad j = 1, .., N_c, \; i = 1, .., N_f. \tag{3.3.1}$$

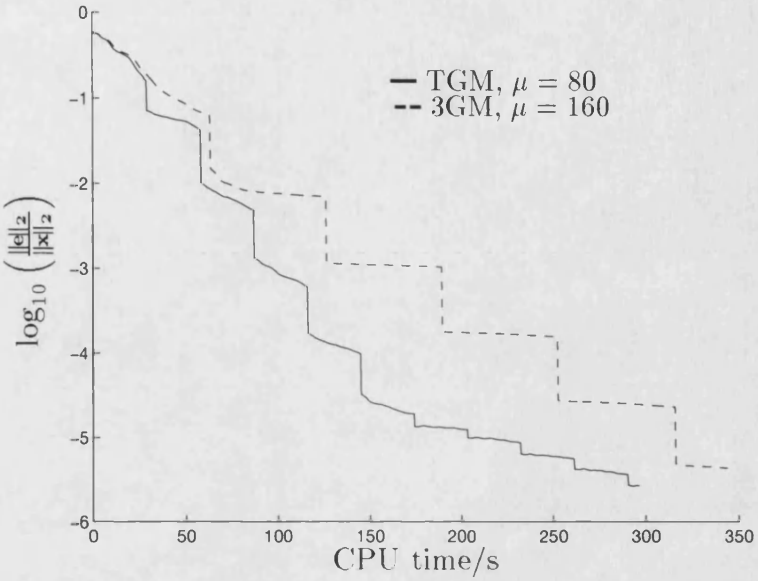

Figure 3-5: *Comparison of best results from TGM and 3GM.*

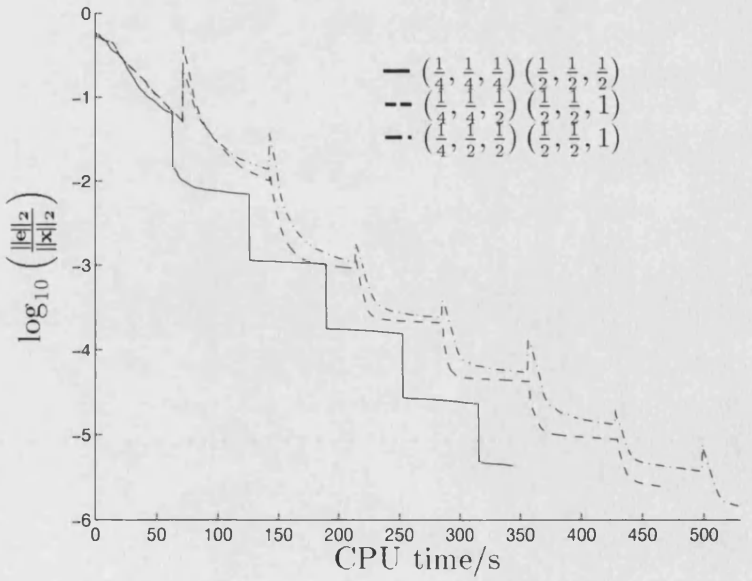

Figure 3-6: *Comparison of 3GM with different levels of coarsening for the middle and coarse grids. 160 smoother steps are used in each case.*

In practice each fine grid point, x_i, lies on only one coarse grid element, and so only a small number of calculations of the form (3.3.1) need to be performed. In fact for the 3D elements used here there will be 27 of these calculations per fine grid point. Hence we will need $27N_f$ basis function evaluations for the 3D problem in order to calculate the prolongation. In 2D the requirement was only $9N_f$ and the time to perform this calculation was not significant. However, in the 3D case this time has become very significant. For the TGM results in §3.2 460 seconds are needed to calculate the prolongation matrix, which is actually longer than the time taken to perform the iterative solve. Although the combined time is still far less than the frontal solver time.

The code used to calculate the basis function evaluation is one of the NAMMU internal routines, and one of the reasons why this time is so high is probably because this code was almost certainly never designed to perform so many evaluations in one go. It is probable that if this routine was optimised the CPU time spent in performing this calculation could be considerably reduced.

3.4 Conclusions

As one would expect, use of the TGM or 3GM gives a far more significant gain in 3D than in 2D. The 3D test problem, with over 172,000 degrees of freedom, is solved to an acceptable level of accuracy in less than 8% of the time required by the direct solver.

Again the 3GM approach is not competitive with the TGM, the TGM converges in less than 90% of the time required by the 3GM, although it is important to note that the 3GM does show a more uniform convergence rate than the TGM.

Chapter 4

Polynomial Preconditioning

4.1 Introduction

In this chapter we consider the use of polynomial preconditioners for the conjugate gradient algorithm to solve the positive definite linear systems arising from finite element discretisations of (1.2.3).

First recall that the principle requirements of the preconditioner are that

- $M^{-1}A \approx I$, i.e. the preconditioned problem is easier to solve than the original problem.
- the linear systems, $M\mathbf{z} = \mathbf{r}$, are easy to solve.

The essential idea of the polynomial preconditioning is to use a polynomial in A, of degree m, as the preconditioning matrix M^{-1}. That is, write

$$M^{-1} = P_m(A). \tag{4.1.1}$$

This form of preconditioning is attractive for two main reasons.

- Implementations of this form of preconditioning are relatively simple once the polynomial coefficients have been calculated, since only a matrix-vector multiplication routine is needed in order to implement the preconditioner.
- A major effect of this preconditioning is essentially to concentrate the solver work in the matrix-vector operations. Recall that conjugate gradients normally requires 1 matrix-vector multiplication and 2 scalar products per iteration.

The first point is of some relevance, since this preconditioning is then ideal for so called, "matrix-free" computations, where the matrix is not explicitly formed. This is principally the situation considered in this thesis, where the action of the matrix on a vector is available through a matrix-vector multiply.

The second point is an important issue for implementation on a supercomputer, such as the Cray YMP. As has been discussed before, the matrix-vector multiplication operation can be constructed to run very quickly. Scalar products, however, are serial operations and cannot generally be constructed to perform well on high performance computer architectures. At least one aspect of the effectiveness of the method will be related to how quickly the matrix-vector operations are performed compared to the scalar products.

Perhaps the simplest example of a polynomial preconditioner can be seen by considering the following:

Example 4.1 *Suppose we can write the matrix A as $A = (I - G)$, where it is assumed that $\| G \| < 1$, for some matrix norm. Then using the Neumann series the following is obtained*

$$(I - G)^{-1} = A^{-1} = I + G + G^2 + G^3 + ... \tag{4.1.2}$$

So in this case the preconditioner polynomial could be taken as

$$P_m(A) = I + G + G^2 + G^3 + ... + G^m \tag{4.1.3}$$

i.e. a simple truncation of this series.

We will see in §4.2 why (4.1.3) might be considered a reasonable preconditioner.

Polynomial preconditioning is not a new, or indeed even recent, idea, one of the early references was made by Lanczos, in the 1950s, see Lanczos[65]. With the advent of the high performance parallel computing polynomial preconditioners have been increasingly important, and much research has been done in this area. Recent work has been done by Ashby, Manteuffel, and Saylor in [3, 4, 8, 9]. See also work by Johnson et al in [57], Saad[78], Fisher and Freund[33], Ruggiero[77], and O'Leary[72].

The remainder of this chapter proceeds as follows: §4.2 discusses some of the choices of polynomial, in particular a least squares polynomial and the Chebyshev minimax polynomial. In §4.3 some numerical results for the polynomial approach are given. Finally in §4.4 we sum up our experience with polynomial preconditioning.

4.2 Choosing the polynomial

Having defined what is meant by a polynomial preconditioner in (4.1.1) we now consider how to select the polynomial $P_m(A)$. First recall that one of our aims in finding a good preconditioner M is to choose it such that

$$M^{-1}A \approx I$$

in some sense. To this end it is required that the quantity given by

$$I - P_m(A)A \quad \text{is "small"} \tag{4.2.1}$$

in some sense. For example, in the context of the Neumann series with $\| G \| < 1$ considered previously in example 4.1,

$$I - P_m(A)A = G^{m+1},$$

and $\| I - P_m(A)A \|$ would be small for sufficiently large m.

Now transform the condition (4.2.1), into an equivalent scalar condition by requiring that

$$1 - P_m(\lambda)\lambda \quad \text{is "small"} \tag{4.2.2}$$

for all eigenvalues, λ, in the eigenspectrum of A. However, in general the entire eigenspectrum of A is not known, but we might expect to have, at worst, a reasonable approximation to the smallest and largest eigenvalues, which will be denoted by λ_1 and λ_N respectively. These approximations might perhaps be produced by performing a few steps of CG and using the Lanczos connection (§1.6) to obtain the estimates.

It is now possible to produce our preconditioner by selecting a norm and finding the polynomial that minimises (4.2.2) in that norm, for $\lambda \in [\lambda_1, \lambda_N]$.

Two choices of norm will now be considered, both of which yield well known polynomials. Note that since this polynomial is to define a preconditioner for the CG method, we ask that it be strictly positive on the eigenspectrum of A. Hence $P_m(A)$ will be positive definite, and the CG method is guaranteed, in the absence of rounding error, to converge.

4.2.1 Least Squares Polynomials

We first consider choosing a norm so as to minimise the overall error in (4.2.2). This is accomplished by choosing a norm defined by the following:

$$\| f(\lambda) \|_w^2 = \int_{\lambda_1}^{\lambda_N} |f(\lambda)|^2 w(\lambda) d\lambda \tag{4.2.3}$$

for some suitable weight function, $w(\lambda)$, that is strictly positive on the eigenspectrum of A. This is usually referred to as the least squares norm.

We now state a result from [57] that allows us to show a class of polynomials that satisfy the positivity condition required by the preconditioner.

Theorem 4.2.1 *Let $s_i(\lambda)$, $i = 1, 2, ..., m+1$ be orthonormal with respect to the arbitrary weight $w(\lambda) > 0, \lambda \in [\lambda_1, \lambda_N]$ and be normalised so that $s_i(0) > 0$. The solution q^* to the problem of finding a polynomial that minimises*

$$\int_{\lambda_1}^{\lambda_N} (1 - q(\lambda))^2 w(\lambda) d\lambda$$

is positive on $[\lambda_1, \lambda_N]$ whenever each s_i, $i = 1, 2, ...m+1$ attains its maximum on $[\lambda_1, \lambda_N]$

at $\lambda = \lambda_1$.

Proof See [57] §5 for details. □

For certain specific choices of weight function the solution to the resulting polynomial approximation problem is well known. In particular for a Jacobi weight, i.e.

$$w(\lambda) = (\lambda_N - \lambda)^\alpha (\lambda - \lambda_1)^\beta \tag{4.2.4}$$

with $\alpha \geq -1$ and $\beta \geq -1$, then the form for the polynomials is well known, see [21]. In particular they can be produced by a simple recursion of the form given in (4.2.8).

If α and β are now restricted such that the condition

$$\beta \geq \alpha \geq -\frac{1}{2} \tag{4.2.5}$$

holds, then from a result in Szegö[84] about orthogonal polynomials, it is possible to show that all Jacobi polynomials in this set achieve their maximum at λ_1. Hence an application of theorem 4.2.1 shows that the resulting polynomial is strictly positive and hence can be used as a conjugate gradient preconditioner. It should be pointed out that in [57] condition (4.2.5) is incorrectly stated as $\alpha \geq \beta \geq -\frac{1}{2}$. Surprisingly this incorrect form has propagated through a considerable amount of the literature on this subject, although since nearly all practical methods have considered the cases where $\alpha = \beta$ this has not caused any problems. Finally note that the commonly used cases where $\alpha = \beta = 0$ and $\alpha = \beta = -\frac{1}{2}$ are generally referred to as the Legendre and Chebyshev least squares weights respectively.

A further advantage of using Jacobi weighted polynomials is that the polynomial can be calculated in a recursive fashion, since the Jacobi polynomials satisfy a three term recursion and the actual coefficients of $P_m(\lambda)$ need never be calculated. Following the approach in [8], we consider the *residual* polynomials defined as

$$R_{m+1}(\lambda) = 1 - P_m(\lambda)\lambda. \tag{4.2.6}$$

First note that these polynomials are orthogonal with respect to the weight function

$w(\lambda)\lambda$ and consequently also satisfy a three term recursion which will be taken to be of the form

$$R_{k+1}(\lambda) = (\phi_k\lambda + \psi_k)R_k(\lambda) - \xi_k R_{k-1}(\lambda) \tag{4.2.7}$$

with the coefficients ϕ_k, ψ_k, ξ_k being generated recursively. In §4.2.1 we will discuss this recursion in more details.

Jacobi Least Squares polynomials

In order to calculate the coefficients ϕ_k and ξ_k we follow the approach given in [57]. First require that there exists a three term recurrence relation for the polynomials $\{s_m\}$ which are orthogonal with respect to the weight function $w(\lambda)$ on $[\lambda_1, \lambda_N]$ and normalised such that $s_m(0) = 1$. To obtain this form we start by considering the interval $[-1, 1]$ where the Jacobi polynomials, $J_j^{\alpha,\beta}(\mu)$, (which we shall denote as $J_j(\mu)$), with weight function $w(\mu) = (1-\mu)^\alpha(1+\mu)^\beta$, , satisfy the following three term recursion

$$\begin{aligned} 2\nu(j+1)(\nu-j+1)J_{j+1}(\mu) &= (\nu+1)[\nu(\nu+2)\mu + \alpha^2 - \beta^2]J_j(\mu) \\ &\quad -2(j+\alpha)(j+\beta)(\nu+2)J_{j-1}(\mu) \end{aligned} \tag{4.2.8}$$

where $\nu = \alpha+\beta+2j$ and $J_{-1}(\lambda) = 0$, $J_0(\lambda) = 1$, and $J_1(\lambda) = \frac{1}{2}(\alpha+\beta+2)\mu + \frac{1}{2}(\alpha-\beta)$.

Now shift and scale these polynomials by writing

$$s_j(\lambda) = \frac{J_j(\mu)}{J_j(\mu_0)} \tag{4.2.9}$$

$$\text{where} \quad \mu = -1 + 2\frac{(\lambda-\lambda_1)}{(\lambda_N-\lambda_1)} \tag{4.2.10}$$

$$\text{and} \quad \mu_0 = -\frac{\lambda_N-\lambda_1}{\lambda_N+\lambda_1} \tag{4.2.11}$$

and combine (4.2.9) and (4.2.8) and re-arrange to produce a three term recurrence for the s_j

$$\begin{aligned} -(\nu+1)\nu(\nu+2)\mu s_j(\lambda)J_j(\mu_0) &= (\nu+1)(\alpha^2-\beta^2)s_j J_j(\mu_0) \\ &\quad -2\nu(j+1)(\nu-j+1)s_{j+1}(\lambda)J_{j+1}(\mu_0) \end{aligned}$$

$$-2(j+\alpha)(j+\beta)(\nu+2)s_{j-1}(\lambda)J_{j-1}(\mu_0). \tag{4.2.12}$$

Re-arranging and using (4.2.10) and (4.2.11) the following is obtained

$$\begin{aligned} -\lambda s_j(\lambda) &= -\left[\frac{(j+1)(\nu-j+1)(\lambda_N-\lambda_1)}{(\nu+1)(\nu+2)}\delta_j\right] s_{j+1}(\lambda) \\ &\quad -\left[\frac{(\lambda_1+\lambda_N)}{2} - \frac{(\alpha^2-\beta^2)(\lambda_N-\lambda_1)}{2\nu(\nu+2)}\right] s_j(\lambda) \\ &\quad -\left[\frac{(j+\alpha)(j+\beta)(\lambda_N-\lambda_1)}{\nu(\nu+1)}\delta_{j-1}^{-1}\right] s_{j-1}(\lambda) \end{aligned} \tag{4.2.13}$$

where

$$\delta_j = \frac{J_{j+1}(\mu_0)}{J_j(\mu_0)} \tag{4.2.14}$$

which itself satisfies the recursion

$$\begin{aligned} \delta_j &= \frac{(\nu+1)[\nu(\nu+2)\mu_0+\alpha^2+\beta^2]}{2\nu(j+1)(\nu-j+1)} \\ &\qquad -\frac{(j+\alpha)(j+\beta)(\nu+2)}{\nu(j+1)(\nu-j+1)}\delta_{j-1}^{-1} \end{aligned} \tag{4.2.15}$$

$$\text{with} \quad \delta_0 = \frac{(\alpha+\beta+2)\mu_0+\alpha-\beta}{2}. \tag{4.2.16}$$

This then gives the recursion for $s_j(\lambda)$, for some j, as

$$\lambda s_j(\lambda) = r_j s_{j+1}(\lambda) - (r_j+t_j)s_j(\lambda) + t_j s_{j-1}(\lambda) \tag{4.2.17}$$

where r_j and t_j are defined from (4.2.13).

Again following the form of [57] note that the optimal polynomial, of degree m, that satisfies the minimal norm condition can be expressed as

$$1 - q_{m+1}^*(\lambda) = \frac{v_{m+1}(\lambda)}{v_{m+1}(0)} \tag{4.2.18}$$

(4.2.19)

where

$$v_m(\lambda) = \frac{s_{m+1}(\lambda) - s_m(\lambda)}{\lambda} \tag{4.2.20}$$

Hence given that $s_j(\lambda)$ satisfies a recursion of the type in (4.2.17) the $v_j(\lambda)$ satisfy a related recursion of the form

$$-\lambda v_j(\lambda) = r_{j+1}v_{j+1}(\lambda) - (t_{j+1} + r_j)v_j(\lambda) + t_j v_{j-1}(\lambda). \tag{4.2.21}$$

Finally, the residual polynomials given by

$$R_{m+1}(\lambda) = 1 - q^*_{m+1}(\lambda) = 1 - P_m(\lambda)\lambda$$

satisfy the recursion

$$\begin{aligned} R_{j+1}(\lambda) &= -\frac{t_j v_{j-1}(0)}{r_{j+1}v_{j+1}(0)} R_{j-1}(\lambda) \\ &\quad + \frac{(t_{j+1} + r_j)}{r_{j+1}} \frac{v_j(0)}{v_{j+1}(0)} R_j(\lambda) \\ &\quad - \lambda \frac{v_j(0)}{r_{j+1}v_{j+1}(0)} R_j(\lambda) \end{aligned} \tag{4.2.22}$$

So that, in the context of (4.2.7), the following hold

$$\psi_k = \frac{(t_{k+1} + r_k)}{r_{k+1}} \frac{v_k(0)}{v_{k+1}(0)} \tag{4.2.23}$$

$$\phi_k = -\frac{v_k(0)}{r_{k+1}v_{k+1}(0)} \tag{4.2.24}$$

$$\xi_k = \frac{t_k v_{k-1}(0)}{r_{k+1}v_{k+1}(0)} \tag{4.2.25}$$

where all of these coefficients can be calculated in the recursive fashion shown in this section.

To illustrate these polynomials we have chosen the interval $[0, 10]$ and plotted the residual polynomials, $R_{m+1}(\lambda)$ for varying m, α and β. See figures (4-1, 4-2, 4-3). In particular two commonly used choices of α and β are illustrated. Firstly, the $\alpha = \beta = -0.5$

case, for which the corresponding polynomials are often referred to as the Chebyshev least squares polynomials. This is the case demonstrated by Saad [78] and considered by Ashby et al [8]. Secondly, we show the $\alpha = \beta = 0$ case, which give the Legendre least squares polynomials, this choice is the main case considered by Johnson et al [57].

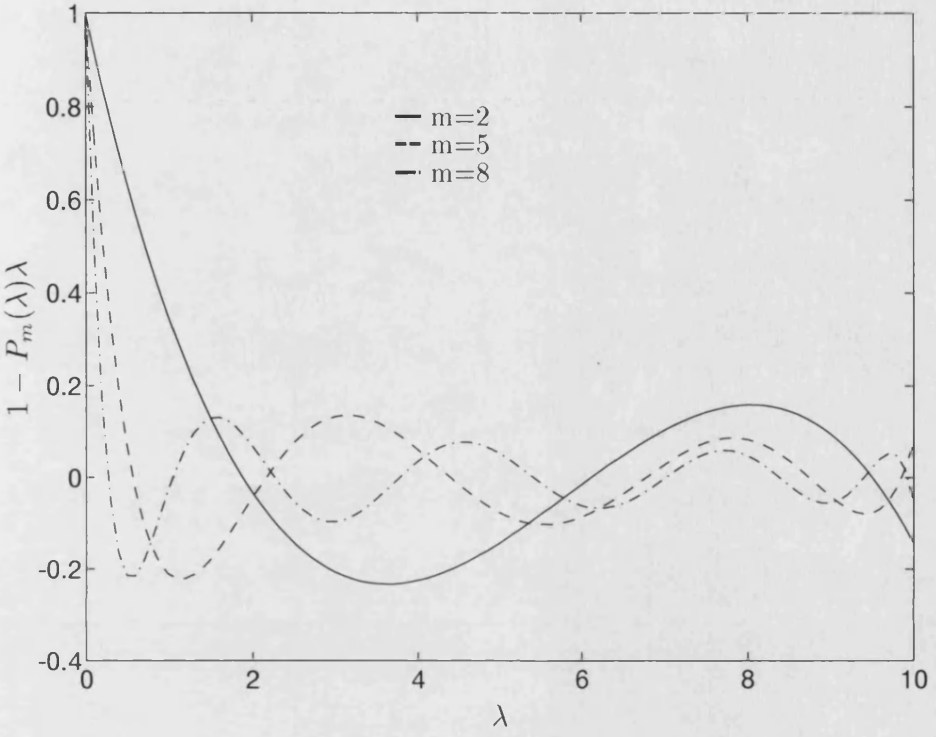

Figure 4-1: *Residual polynomials,* $R_{m+1}(\lambda) = 1 - P_m(\lambda)\lambda$ *with* $\alpha = \beta = -0.5$

It is an open question which choice of α and β will give optimal performance for a particular eigenspectrum. From figure 4-3 it is clear that increasing β over α will reduce the residual at the higher end of the spectrum but only at the cost of increasing it on the lower spectrum. This perhaps indicates that this using this preconditioning, with a suitable selection of α and β might be appropriate for use as a smoother in the two grid method of chapter 2. In addition it would appear that the Chebyshev choice provides a best choice for an overall minimum, biased towards neither end of the spectrum.

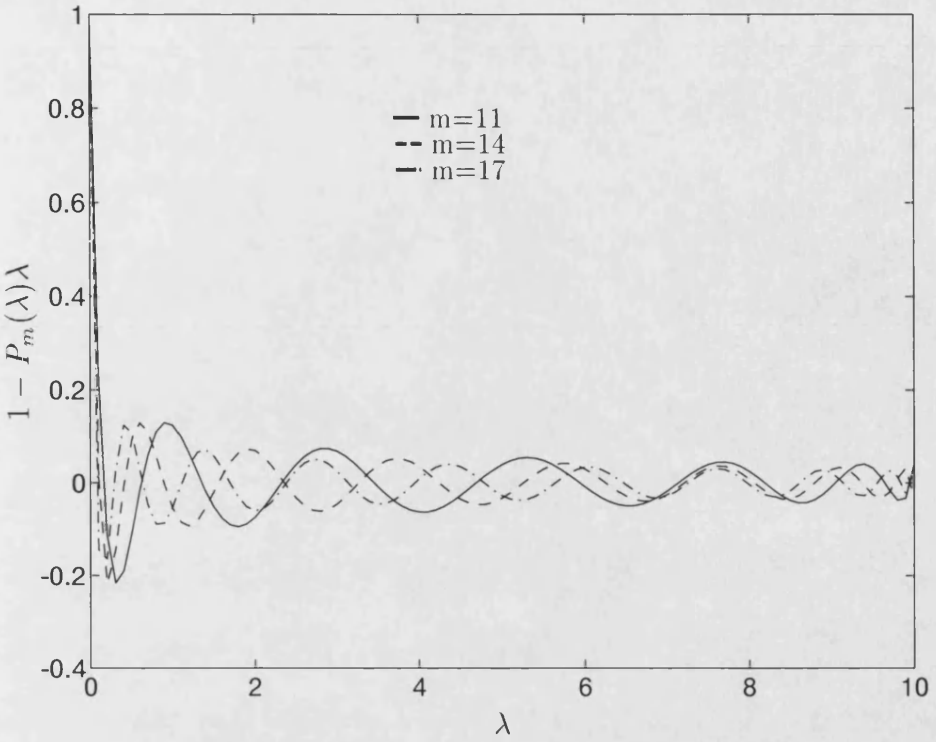

Figure 4-2: *Residual polynomials,* $R_{m+1}(\lambda) = 1 - P_m(\lambda)\lambda$ *with* $\alpha = \beta = -0.5$

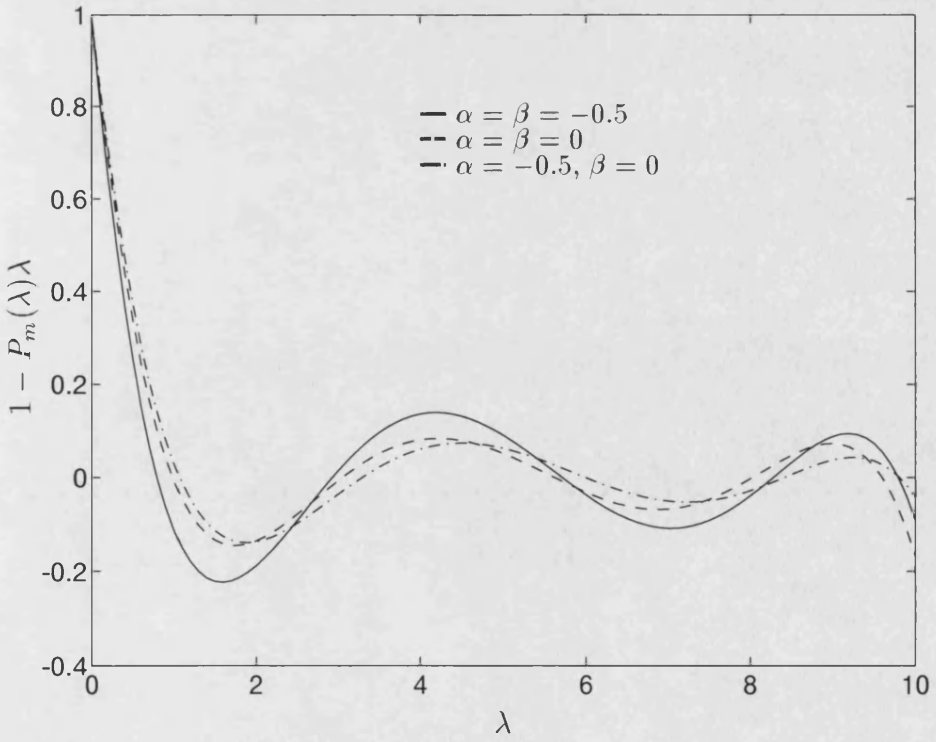

Figure 4-3: *Residual polynomials,* $R_{m+1}(\lambda) = 1 - P_m(\lambda)\lambda$ *with* $m = 4$ *and varying* α, β

Effectiveness of the least squares polynomials

It is known, see §1, that the conjugate gradient algorithm converges, at least in a heuristic sense, in $O(\sqrt{\kappa(A)})$ iterates. This leads us to the question of how a least squares polynomial of degree m affects the condition number and hence the number of iterates to convergence. We first use the fact that polynomial solutions of the least squares problem (4.2.3) minimise $\| R_{m+1} \|_w$, over all polynomials R of degree $\leq m+1$, such that $R(0) = 1$. Hence

$$\| R_{m+1}(\lambda) \|_w \leq \| Q_{m+1}(\lambda) \|_w \tag{4.2.26}$$

for all polynomials Q of degree $m+1$, such that $Q_{m+1}(0) = 1$.

Now examine the polynomial given by

$$Q(\lambda) = \left(1 - \frac{\lambda}{\frac{1}{2}(\lambda_1 + \lambda_N)}\right)^{m+1}. \tag{4.2.27}$$

Clearly,

$$Q_{m+1}(\lambda) \leq \left(\frac{\frac{1}{2}(\lambda_1 + \lambda_N) - \lambda_1}{\frac{1}{2}(\lambda_N + \lambda_1)}\right)^{m+1}, \text{ for } \lambda \in [\lambda_1, \lambda_N].$$

Hence, from (4.2.26),

$$\| R_{m+1} \|_w \leq \| \left(1 - \frac{\lambda}{\frac{1}{2}(\lambda_1 + \lambda_N)}\right)^{m+1} \|_w \leq \| 1 \|_w \left(\frac{(\lambda_N - \lambda_1)}{(\lambda_N + \lambda_1)}\right)^{m+1}.$$

This implies that the norm of R_{m+1} tends to zero as m tends to infinity at least geometrically. Which in turn implies that $P_m(A)A$ tends to I as m tends to infinity (cf. §1.6.5).

Another question that might be asked is how the extra matrix-vector multiplies required for each step of the CG algorithm affect the overall work count to produce a solution.

Heuristically we can argue as follows: Assume CG converges in k_{CG} steps, which in turn means k_{CG} matrix-vector operations and $2k_{CG}$ scalar products. Similarly for PCG with

degree m polynomial, we require $k_{\mathrm{PCG}}(m+1)$ matrix-vector operations and $2k_{\mathrm{PCG}}$ inner products.

Next note that

$$\begin{aligned} k_{\mathrm{CG}} &= O\left(\sqrt{\kappa(A)}\right) \\ k_{\mathrm{PCG}} &= O\left(\sqrt{\kappa(P_m(A)A)}\right) \end{aligned}$$

and since both A and $P_m(A)$ are symmetric, positive definite,

$$\begin{aligned} \kappa(A) &= \lambda_N/\lambda_1 \\ \kappa(P_m(A)A) &= (P_m(\lambda_N)/P_m(\lambda_1))(\lambda_N/\lambda_1) \text{ for } m \text{ even.} \end{aligned}$$

For the following case consider the Jacobi weighted Least Squares polynomials with $\alpha = \beta = -0.5$.

m	$\sqrt{\frac{P_m(\lambda_N)}{P_m(\lambda_1)}}$	$(m+1)\sqrt{\frac{P_m(\lambda_N)}{P_m(\lambda_1)}}$
0	1	1
1	0.5590	1.1180
2	0.3780	1.1339
4	0.2335	1.1677
8	0.1325	1.1921
16	0.0710	1.2071

Table 4.1: *Table of relative condition numbers and relative work counts*

In table 4.1 it can be seen that as the degree, m, of the preconditioning polynomial is increased the conditioning of the preconditioned system decreases, as expected. However, when we consider what this means in terms of the number of matrix-vector multiplies it can be seen that as the degree of polynomial is increased the overall number of matrix-vector multiplication operations increases.

So heuristically PCG requires more matrix-vector operations to reach comparable errors, for this choice of polynomial. Although of course in reaching this degree of error we will require fewer inner-products, which, as has been mentioned before, is one of the main aims of this type of preconditioning, as we are considering the inner-products to

be relatively expensive compared to the matrix-vector products.

Finally recall that conjugate gradients with no preconditioning can itself be considered as a polynomial iterative method. Where the polynomial is dependent on both the matrix and initial residual and is effectively chosen to minimise the error in the $\| \, . \, \|_A$. Since this polynomial is optimal, in this A-norm, preconditioning with some other polynomial, based on some other norm, would seem to be a conflicting idea. Again the key idea lies in the saving of the inner-products. The net result is then to minimise the error in some norm that is a combination of the $\| \, . \, \|_A$ and the norm we construct our polynomial preconditioner from.

Practical least squares

In practice maximum and minimum eigenvalues, i.e. λ_1 and λ_N, would not be known, and although a good approximation to the largest eigenvalue could be obtained reasonably cheaply by, for example, the power method, a good approximation to the smallest would be difficult to obtain cheaply. However, Saad[78] has advocated using least squares preconditioners where λ_1 is taken as 0 and a Gershgorin estimate is used for the largest eigenvalue. This is motivated by the fact that the least squares polynomial is only marginally affected by the smallest eigenvalue. This method is attractive in that once the preconditioner is set up no further processing is required, and in addition no estimates of eigenvalues are ever needed, except the relatively cheap calculation of the Gershgorin estimate.

More recent work by Fischer and Freund[33] has used a more general weight function that is designed to reflect the full distribution of eigenvalues. In essence the method would build up an approximation to the eigenspectrum using a Lanczos or similar method, this eigenspectrum would then be used to create a suitable weight function. In this case, since the weight function is of a more general form, we cannot produce an exact formula for the preconditioning polynomial and hence are required to calculate it in some iterative fashion. Their results suggest that this approach shows some promise, and certainly appears more effective than simple using a simple Legendre, or Chebyshev weight. In contrast O'Leary[72] investigates the use of a polynomial preconditioner

based on the residual polynomial arising from use of the conjugate gradient algorithm. She suggests that this technique has an advantage over the least squares approach in that no eigenvalue estimates are ever needed. She then goes on to comment that, from her experiments, it would appear that "the least squares polynomial preconditioner often provides a saving in matrix-vector multiplications, but not much saving in number of iterations."

4.2.2 Chebyshev Polynomials

We now seek a polynomial that minimises

$$\| 1 - P_m(\lambda)\lambda \|_\infty = \max_{\lambda \in [\lambda_1, \lambda_N]} |1 - P_m(\lambda)\lambda|. \tag{4.2.28}$$

Solutions to this are well known, and are in fact given by the Chebyshev minimax polynomials.

Choosing these minimax polynomials for our preconditioner has two main advantages. Firstly, this polynomial has an equioscillation property in that the oscillations of $P_m(\lambda)\lambda$ are equal about 1, in $[\lambda_1, \lambda_N]$. This means that the resulting preconditioning polynomial has no bias in its reduction of the eigenspectrum. This is in contrast to the least squares polynomials, where the reduction tends to be greatest in the upper end of the spectrum. A second advantage of the Chebyshev choice is that as long as the estimates for λ_1 and λ_N strictly contain the entire spectrum of A, then

$$P_m(\lambda)\lambda \subset [1 - \epsilon_m, 1 + \epsilon_m] \tag{4.2.29}$$

where

$$\epsilon_m = \| 1 - P_m(\lambda)\lambda \|_\infty < 1, \ \lambda \in [\lambda_1, \lambda_N]. \tag{4.2.30}$$

Hence the preconditioned matrix, $P_m(A)A$, is necessarily positive definite, this being one of our main requirements for the preconditioner.

Using the norm given in (4.2.28), we now write our polynomial $P_m(\lambda)$ as a shifted and

scaled Chebyshev polynomial

$$1 - P_m(\lambda)\lambda = \frac{T_m\left(\frac{\lambda_N+\lambda_1-2\lambda}{\lambda_N+\lambda_1}\right)}{T_m\left(\frac{\lambda_N+\lambda_1}{\lambda_N-\lambda_1}\right)} \tag{4.2.31}$$

where $T_m(\mu)$ is the mth Chebyshev polynomial. In a very similar fashion to the least squares polynomial, this also can be computed from a three-term recursion.

The Chebyshev preconditioner also satisfies the following property

Theorem 4.2.2 *The polynomial, P, that minimises*

$$\kappa(P(A)A) = \frac{\max_{\lambda\in\sigma(A)}|P(\lambda)\lambda|}{\min_{\lambda\in\sigma(A)}|P(\lambda)\lambda|} \tag{4.2.32}$$

where $\sigma(A)$ is the eigenspectrum of A, is given by the Chebyshev preconditioning polynomial (4.2.31)

Proof We follow the proof given in [8]. First note that we may assume that $P(\lambda)\lambda > 0$ for $\lambda \in \sigma(A)$. Now consider only polynomials, P, where

$$1 - \min_{\lambda\in\sigma(A)}(P(\lambda)\lambda) = \max_{\lambda\in\sigma(A)}(P(\lambda)\lambda) - 1 = \epsilon$$

So now (4.2.32) becomes a problem of minimising the quantity $\frac{1+\epsilon}{1-\epsilon}$. Which is equivalent to finding a solution to the earlier problem (4.2.2). □

In figure 4-4 we show the residual polynomials of degrees 2,5 and 8 on the interval $[1, 10]$. Comparing these polynomials with similar ones for the least squares polynomials in figure 4-1, it would appear to indicate that the Chebyshev polynomials are far superior. However, note the considerable difference when the interval is changed to $[0.1, 10]$ in figure 4-5. Unlike the least squares polynomials, the Chebyshev polynomials are very sensitive to changes in the smallest eigenvalue.

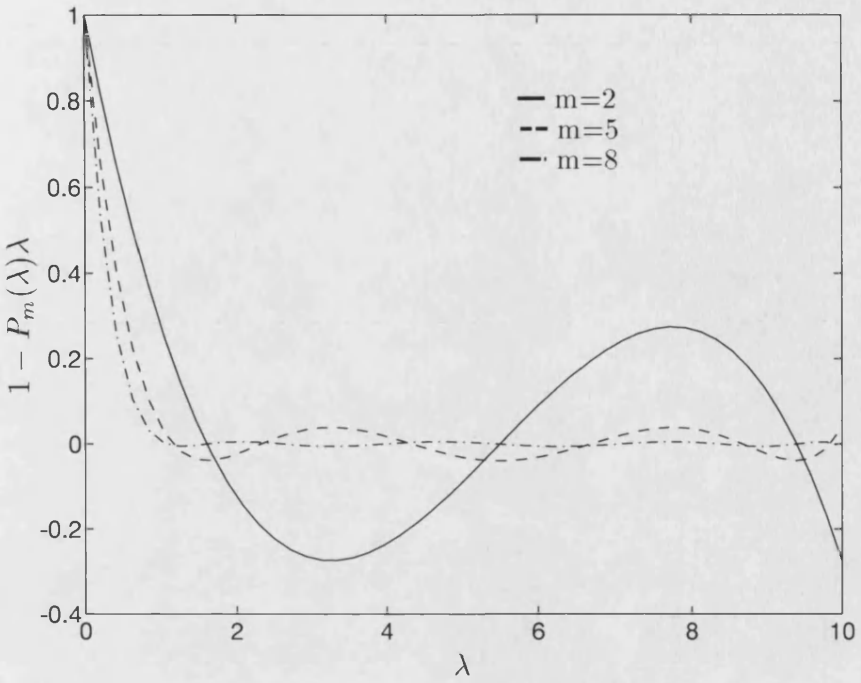

Figure 4-4: *Chebyshev Residual polynomials,* $R_{m+1}(\lambda) = 1 - P_m(\lambda)\lambda$, *on the region* $[1, 10]$.

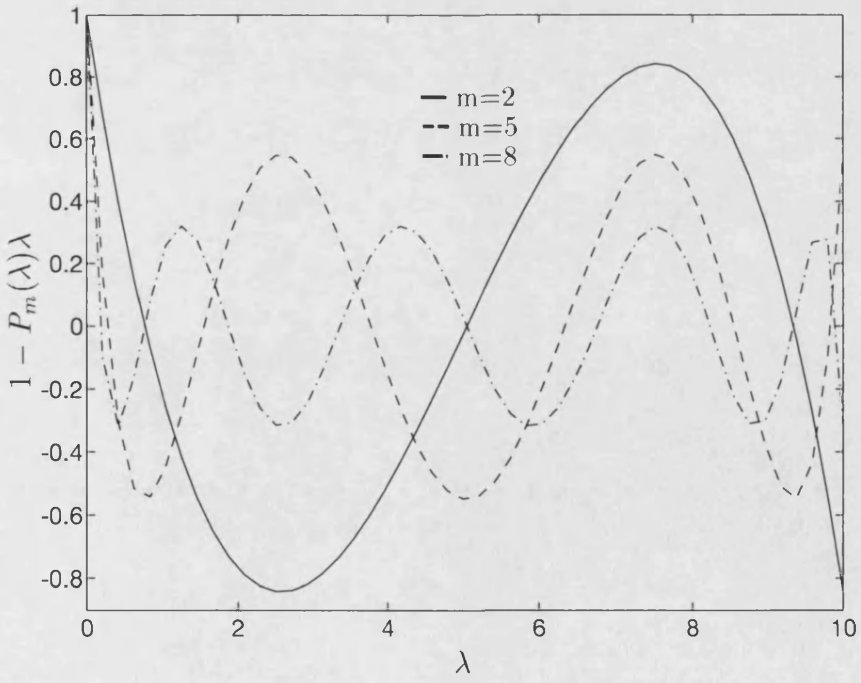

Figure 4-5: *Chebyshev Residual polynomials,* $R_{m+1}(\lambda) = 1 - P_m(\lambda)\lambda$, *on the region* $[0.1, 10]$.

Practical Chebyshev

To work effectively this type of polynomial preconditioner generally requires good estimates of the upper and lower eigenvalues bounds. In particular the bounds must contain the entire spectrum or the polynomial cannot be guaranteed to be positive definite. Considerable work on this type of polynomial preconditioning has been done by Ashby[3, 8, 4], with an emphasis on adaptive algorithms that estimate the upper and lower eigenvalues as the CG iteration proceeds, modifying the polynomial accordingly. For problems with eigenvalues spread relatively uniformly around the interval, Ashby[8] reports good results for this form of polynomial.

4.2.3 Implementation

We conclude this section by discussing how the polynomial preconditioning would be implemented in practice. From the residual polynomial formulation, e.g. (4.2.7), it is then possible to actually implement the polynomial preconditioning. Note that the technique used is similar to that in [4]. Suppose that we were using a polynomial iterative method to solve the linear system $A\mathbf{x} = \mathbf{v}$. This polynomial method would then have an associated residual polynomial, $R_m(A)$, such that

$$R_m(A) = I - P_m(A) = I - C(A)A. \tag{4.2.33}$$

We wish to use the polynomial $C(A)$ as our preconditioner, and hence need to be able to compute the quantity $C(A)\mathbf{v}$.

Assume that the polynomial iterative method has iterates $\mathbf{x}_0, \mathbf{x}_1, ...$, with associated residuals $\mathbf{r}_m$, such that

$$\mathbf{r}_m = \mathbf{v} - A\mathbf{x}_m = R_m(A)\mathbf{r}_0. \tag{4.2.34}$$

If we assume that the iteration method starts with $\mathbf{x}_0 = \mathbf{0}$, then clearly

$$\mathbf{r}_m = R_m(A)\mathbf{v} = (I - C(A)A)\mathbf{v} \tag{4.2.35}$$

and hence

$$\mathbf{e}_m = \mathbf{x} - \mathbf{x}_m = (I - C(A)A)\mathbf{e}_m \tag{4.2.36}$$

$$= (A^{-1} - C(A))\mathbf{v}, \tag{4.2.37}$$

since

$$A^{-1}\mathbf{r} = \mathbf{e}.$$

So, using (4.2.35) and (4.2.37), we have

$$\mathbf{x}_m = C(A)\mathbf{v}. \tag{4.2.38}$$

Hence the m^{th} step of the polynomial iterative method gives us the preconditioned vector we desire.

Finally, using a result in Hageman[47, pp40-41], the three term recursion for the residual polynomial, i.e. (4.2.7), can be used to formulate a corresponding three term recursion for the iterates, see [47, theorem 3-2.1] for more details.

We now present an algorithm for implementing the polynomial preconditioning using the three term recursion in (4.2.7).

Algorithm 4.1 [$\mathbf{w} = P_m(A)\mathbf{v} = C(A)\mathbf{v}$] This algorithm multiplies the vector $\mathbf{v}$ by the polynomial in A, given by the three term recursion for the residual polynomial (4.2.7).

$$\begin{aligned}
\mathbf{w}_0 &= 0 \\
\mathbf{r}_0 &= \mathbf{v} \\
\boldsymbol{\Delta}_0 &= -\phi_0\mathbf{r}_0 \\
\text{loop } & i = 0, \ldots, m-1 \\
\mathbf{w}_{i+1} &= \mathbf{w}_i + \boldsymbol{\Delta}_i \\
\mathbf{r}_{i+1} &= \mathbf{v} - A\mathbf{w}_{i+1} \\
\boldsymbol{\Delta}_{i+1} &= \xi_{i+1}\boldsymbol{\Delta}_i - \phi_{i+1}\mathbf{r}_{i+1}
\end{aligned}$$

endloop

$$\mathbf{w} = \mathbf{w}_m$$

It is also worth noting that the quantities r_m and Δ_m, in algorithm 4.1, do not need to be calculated, thereby saving one matrix-vector multiply.

4.3 Numerical results

For the results presented here we first consider a 2D problem with a very simple mesh of the form given in figure 1-9, and constant permeability over the entire region. This problem has 56616 degrees of freedom. A second 2D problem arising from a physical problem is also considered. This problem has 58944 degrees of freedom. In order to improve convergence the stiffness matrices for both problems were symmetrically diagonally scaled prior to running the conjugate gradient code. As was seen in chapter 2, this diagonal scaling greatly mitigates the effect of the discontinuities in the permeability coefficient, k.

The polynomials that we decided to use were least squares polynomials based on Jacobi weights, with $\alpha = \beta = -0.5$. Where we use the Saad idea of taking $\lambda_1 = 0$ and a Gershgorin estimate for the largest eigenvalue, λ_N.

The reasoning behind trying this form of polynomial is that its implementation is the most straightforward and the literature comparing and describing various polynomial preconditioners [8, 33] would suggest that whilst other types of polynomials can produce faster solutions, in terms of CPU time, this relatively simple version of preconditioning is indicative of the effects of such preconditionings.

Various degrees of polynomial were tried, and in all cases we compare the $\|\,.\,\|_2$ of the error at each successive iterate with the CPU time used.

For the first problem we use polynomials of degrees 5, 10 and 20. In figure 4-6 we compare CG with polynomial preconditioned CG. It is clear that in all cases the method

converges well, although the fastest rate of convergence is given by CG with no polynomial preconditioner at all. For this problem the frontal solver requires approximately the same amount of CPU time as the CG. As a final comparison for this problem in figure 4-7 we compare the TGM from chapter 2, with 20 smoothing steps, with CG. It is immediately clear that the TGM is considerably faster in this case.

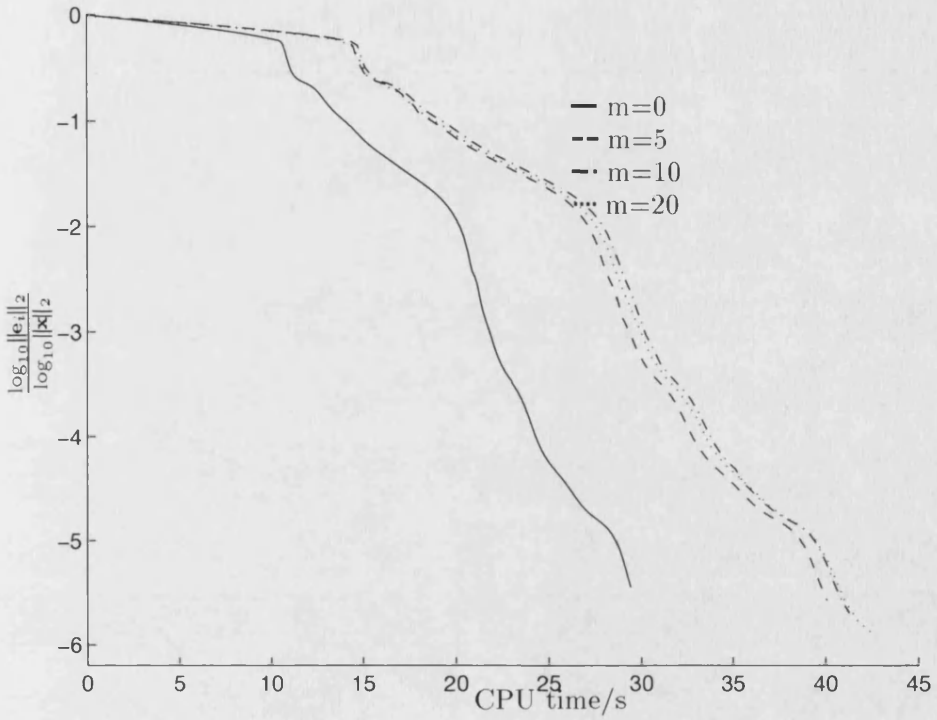

Figure 4-6: *Error vs. CPU time for CG and PCG on a simple mesh problem.*

In figures 4-8, 4-9, 4-10 we plot $\log_{10} \| \mathbf{e} \|_2$ versus CPU time used in seconds. In all cases we plot the output for CG with no polynomial preconditioning (solid line) for comparison purposes.

In figure 4-8 we see that degree 4 and 7 polynomials have very little effect at all, the lines diverge very slightly in the favour of the polynomial method. Figures 4-9 and 4-10 show marginally more difference, with the polynomials again performing very slightly better.

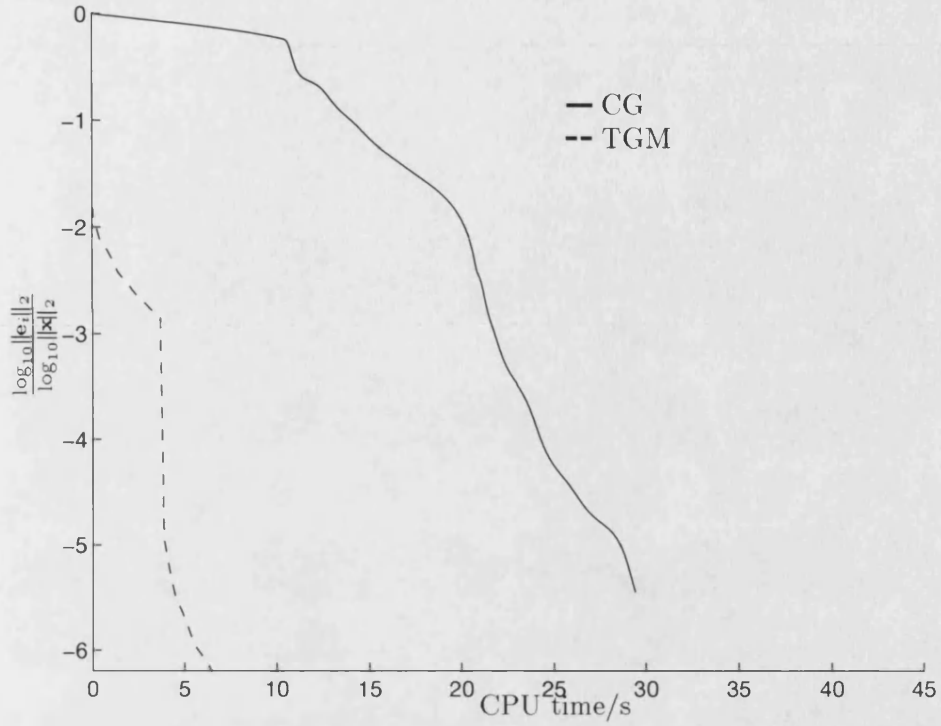

Figure 4-7: *Error vs. CPU time for CG and TGM of chapter 2.*

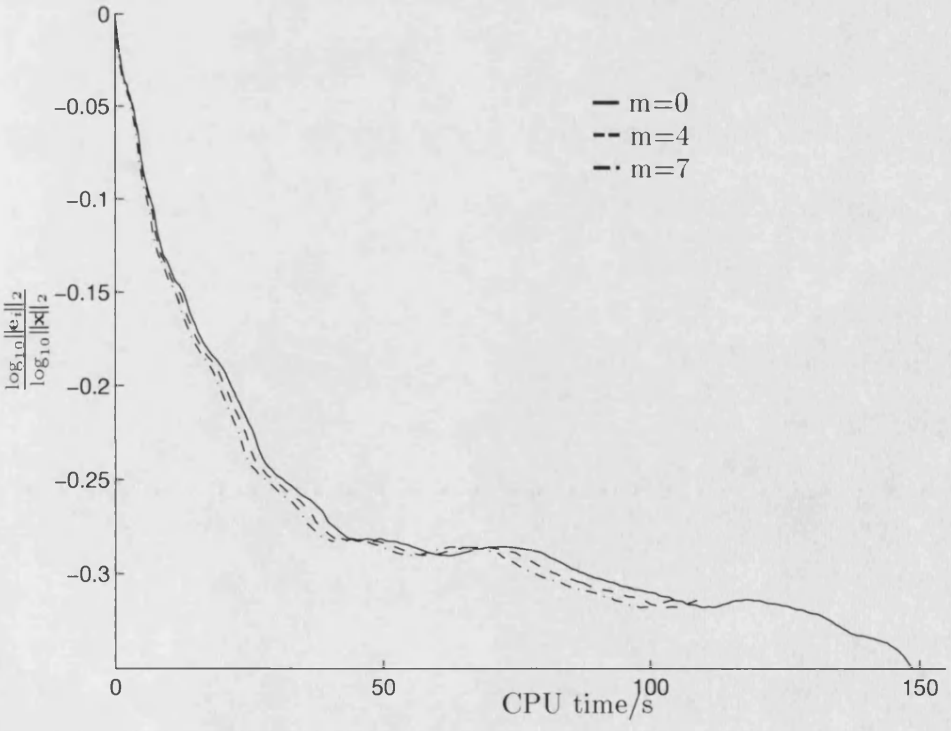

Figure 4-8: *Error vs. CPU time for CG and PCG*

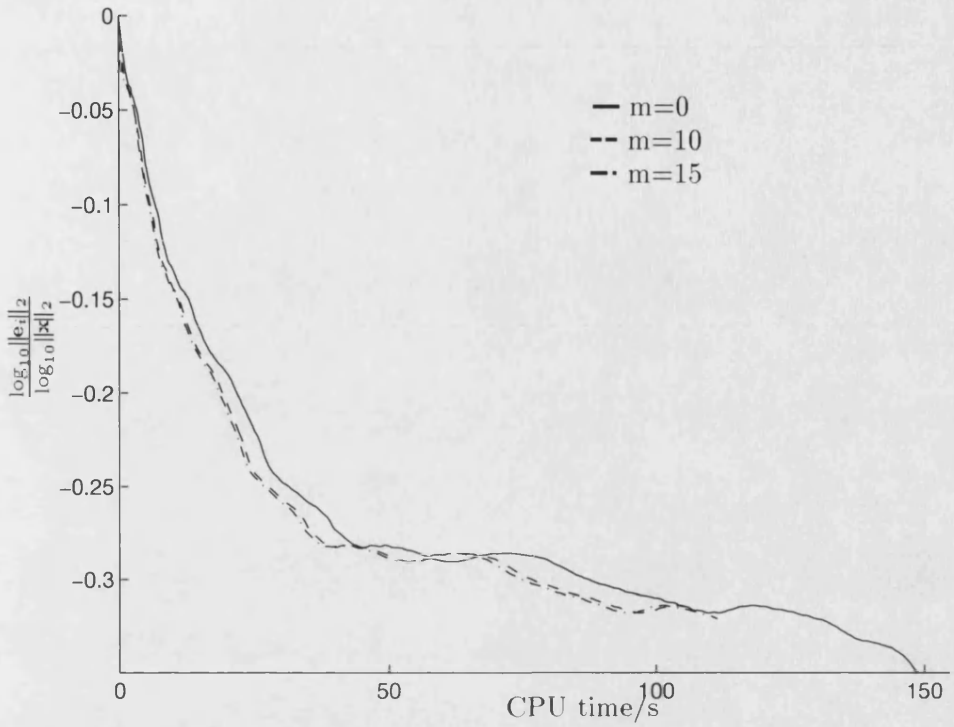

Figure 4-9: *Error vs. CPU time for CG and PCG*

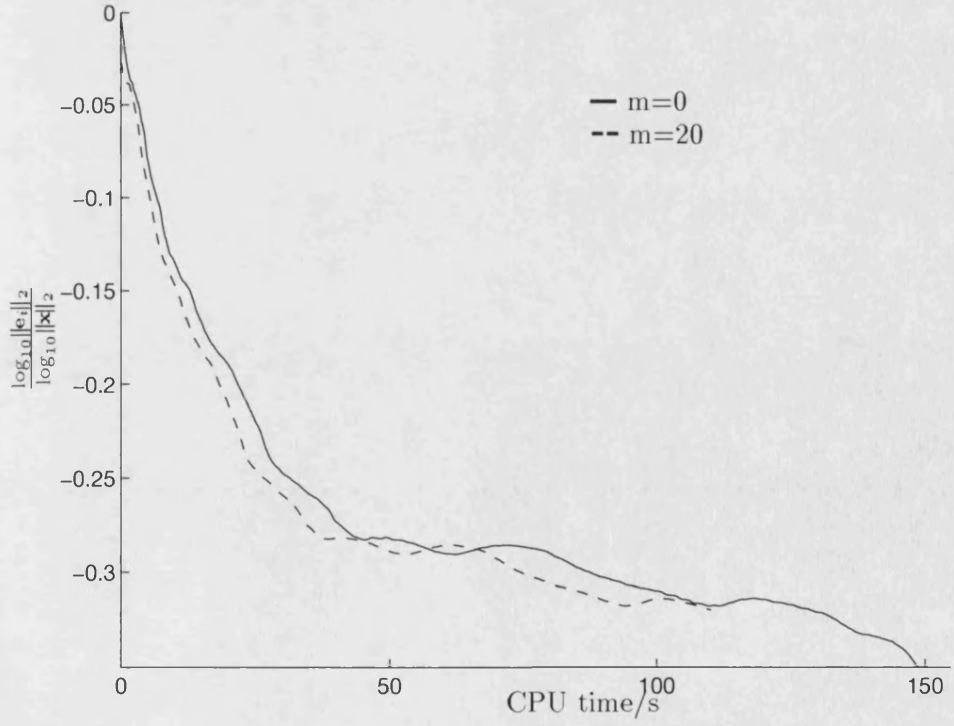

Figure 4-10: *Error vs. CPU time for CG and PCG*

4.4 Conclusions

As can be seen from the graphs, CG with no preconditioning, as applied to the symmetrically scaled system, performs very poorly on the "hard"problem. When we apply polynomial preconditioning there is little apparent change in the convergence rate based on error reduction vs. CPU time. There is some apparent difference in error reduction between CG and PCG, in all cases for the realistic problem the polynomial preconditioned CG performed at least as well as standard CG, this is perhaps reassuring, since in all preconditioned cases more matrix-vector multiplies were required to achieve the same level of accuracy.

It is believed that with a better implementation of the matrix-vector multiplication routine this difference could be increased, although it should be pointed out that the architecture of the computer plays a major factor in any speedup. However, it would seem that polynomial preconditioning is unable to affect the underlying convergence of the CG method.

For this reason we shall consider the polynomial preconditioning as a potentially effective *implementation* of the CG method rather than a *preconditioning* of the system in the sense that convergence is improved.

Chapter 5

ILU Preconditioning

5.1 Introduction

In this chapter the incomplete LU (ILU) factorisation, eg [62], is used as a preconditioner for the conjugate gradient (CG) method, to solve the linear systems arising in the finite element discretisation of the groundwater flow model discussed in chapter 1. Here L and U denote lower and upper triangular matrices respectively.

To motivate the preconditioner we first recall the two important properties of preconditioners for the CG algorithm. Firstly, given a preconditioner M, it should be possible to solve the system $M^{-1}A$ quicker, in terms of CPU time, than the original system involving A. An important factor in this is that computations of the form $M^{-1}\mathbf{v}$ are simple and quick to compute.

Clearly we could choose $M = A$, and perform the LU factorisation of M to enable us to compute the action of the preconditioner. This would satisfy some of the requirements outlined above, although this is obviously not a practical choice, due to the high cost of the LU factorisation. Suppose, however, that some preconditioner M were stored as an LU factorisation, but that both L and U were sparse in some sense. Then the computations involving the action of the preconditioner would take little time to perform.

In the ILU method we form a preconditioner M by calculating an approximation to the standard LU factorisation of A, based on Gaussian elimination, e.g. [38, §3.2], so that we have

$$\hat{L}\hat{U} \approx LU = A. \tag{5.1.1}$$

Here $\hat{L}$ and $\hat{U}$ reflect something of the sparsity pattern in A. The CG preconditioner M is then given by writing $M = \hat{L}\hat{U}$. This approximation is generally called an *incomplete factorisation.*

By controlling how the $\hat{L}\hat{U}$ factors are produced, particularly considering stability and storage issues, various types of incomplete factorisations can be created.

The incomplete factorisation is very widely used as a preconditioner, with notable early work by Meijerink and van der Vorst in [68], Kershaw[62] and Gustafsson[42]. Since then considerable work in the area has been done by Axelsson et al in [14, 13, 15, 16], with other references including [69, 90, 25, 70, 71, 50, 81, 88], although there is a vast amount of literature available on this subject. One of the main reasons for the popularity of the incomplete factorisation is the fact that it is a "black box" preconditioner, in the sense that no assumptions about the problem involved need to be made. The matrix is merely "fed" to the incomplete factorisation code and a preconditioner is created. This is both a strength and a weakness in the method, a "black box" technique is very useful, but the lack of reliance on the problem means that it can often not work at all, usually "fine-tuning" is needed to obtain satisfactory results.

It must be made clear that the main thrust of this chapter is not to re-invent the incomplete factorisation, nor to produce a new method or code. Rather we merely wish to view the effectiveness of a standard, easily implemented, "off the shelf" ILU code, for the groundwater flow problem and compare the results with the two grid method from chapter 2. In particular the following issues need to be addressed:

(a) convergence of the method

(b) performance in terms of CPU time

(c) storage requirements

(d) overall comparison of the method with both a direct, ie exact, solver and an

effective iterative solver, e.g. the two grid method, discussed in chapter 2.

Point **(a)** is clearly very important, as is seen in chapter 1 §1.6, where we considered merely diagonal scaling as a preconditioning technique. An effective preconditioner is vital to ensure robust convergence for the problems considered here. Points **(b)** and **(c)** relate more to the effect of using sparse matrix techniques than the ILU factorisation method itself. Point **(d)** is an issue of direct relevance, since the two grid method, from chapter 2, has already been shown to be a competitive method.

The structure of the chapter proceeds as follows: in §5.2 a general overview of the incomplete factorisation approach for symmetric positive definite systems is given, together with some discussion of the problems caused by the matrices not being diagonally dominant or M matrices. In §5.3 further issues relating to the particular NAMMU code implementation are reviewed, although this is mainly concerned with the effect of using a sparse storage scheme. In §5.4 the ILU code is tested with a relatively simple problem, and comparisons are made between results from this and simple diagonally scaled conjugate gradients. In §5.5 brief comments are made about the element-by-element approach to forming a preconditioner. Finally we summarise our experience with the incomplete factorisation in §5.6.

5.2 Incomplete LU

5.2.1 The basic approach

The simplest approach to forming the incomplete factors in (5.1.1) is to use *zero fill-in*: exactly follow the standard LU factorisation method, except that only entries in the LU factors that correspond to non-zero entries in the original matrix are allowed to be written to with any other entry being discarded. Note that, in general, an LU factorisation of a very sparse matrix will require considerably more storage than the original matrix. The *zero fill-in* approach clearly means that the sparsity, and therefore storage requirements, of the original matrix are preserved in the preconditioner. The potential for lost entries in the approximate LU factors can be seen in a more precise way (see [14, pp38-39]) by examining the envelope of non-zero entries of the sparse

matrix. Suppose that we have an $N \times N$ symmetric matrix A. Define the following

$$m(i) = \min \{j; (1 \leq j \leq i) \cap (a_{ij} \neq 0)\}, \quad i = 1, 2, .., N \tag{5.2.1}$$

i.e. $a_{i,m(i)}$ is the first non-zero entry in the i^{th} row. Then the envelope of entries containing all the non-zero entries of A are given by the set of index pairs

$$S = \{(i, j) \cup (j, i); m(i) \leq j \leq i, \ 1 \leq i \leq N\}. \tag{5.2.2}$$

The area of fill-in produced during the LU factorisation is then contained within this envelope. This point is illustrated in figure 5-1, where the diagonal lines show the envelope of potential fill-in.

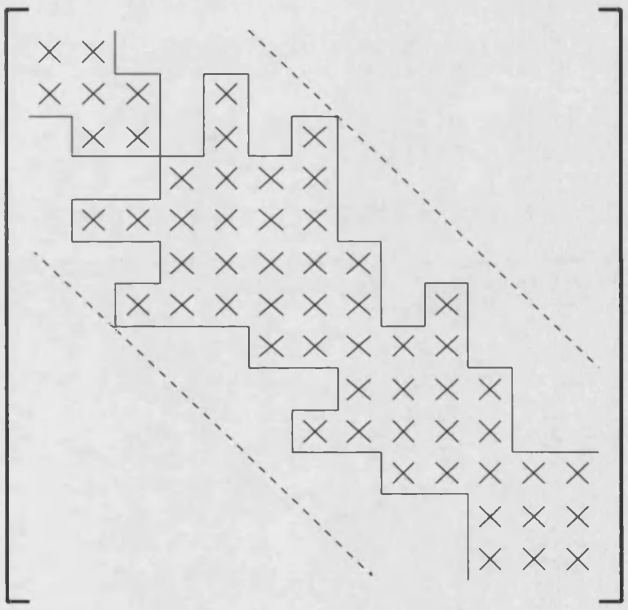

Figure 5-1: *The envelope of a symmetric matrix.*

For a 2D or 3D finite element problem the resulting stiffness matrix typically has a large envelope that contains mostly zero entries. In the following simple example we demonstrate a finite element matrix with a large envelope. For convenience the L and U factors are displayed in one matrix.

Example 5.1 *Consider the stiffness matrix arising from the two dimensional finite element discretisation of Laplace's equation, using biquadratic basis functions on quadri-*

lateral elements. For a problem with 1056 degrees of freedom, the sparsity pattern of the matrix can be seen in figure 5-2. If we perform a full LU decomposition on this matrix it is found that the sparsity pattern of this decomposition is now of the form seen in figure 5-3. Note that whilst the original matrix required only 15996 non-zeros, the full LU factors require 140182 non-zeros, a gain of nearly a factor of 10.

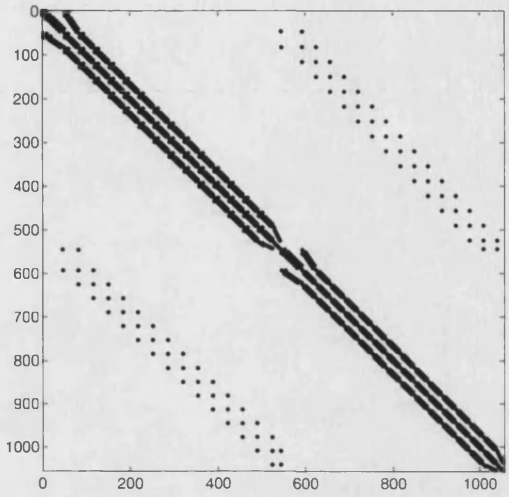

Figure 5-2: *Sparsity pattern of a finite element stiffness matrix for Laplace's equation in 2D, with 1056 degrees of freedom.*

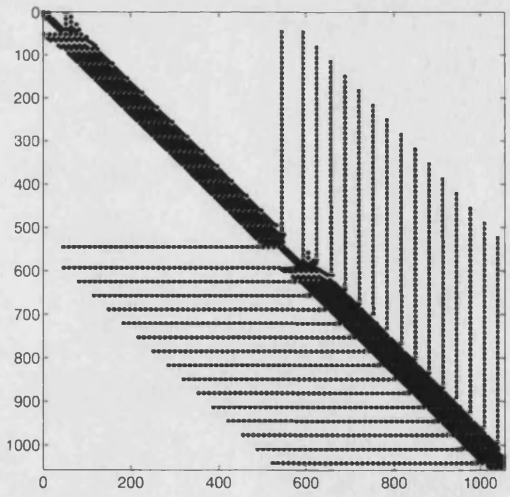

Figure 5-3: *Sparsity pattern of storage for full LU decomposition of the matrix in figure 5-2.*

In our case the matrix under consideration is symmetric positive definite, and hence it makes more sense to consider an *Incomplete Cholesky* factorisation, often denoted in the standard literature as an IC factorisation. If this is then used as preconditioner in a CG method then the resulting algorithm is denoted by ICCG.

5.2.2 Modified ILU

Many modifications of the simple ILU factorisation approach outlined in §5.2.1 exist. In some versions the entries in the LU factors that would have caused fill-in are not discarded but are instead added to the corresponding diagonal entry on that row. The algorithm presented below incorporates this feature. Gustafsson[42] and Axelsson and Munksgaard[17] examined this particular modification. It is generally found that for matrices arising from finite element methods this approach is more effective than the zero-fill in approach partly because more of the matrix properties, such as row sums, are preserved.

Axelsson and Barker[14, §7.2] define a method that they term the *modified incomplete factorisation*. Here the incomplete factorisation described previously is modified such that the new method is mathematically equivalent to applying the incomplete factorisation, as in §5.2.1, to a matrix with a shifted diagonal, i.e. $A + D$. This method was principally developed with matrices from finite element methods in mind, since after selecting the shifting diagonal in a particular fashion, they are able to prove a reduction in the condition number, κ, of the finite element stiffness matrix, if the matrix belongs to a certain class of M-matrices. More specifically they show that

$$\kappa(M^{-1}A) = O(\kappa(A)^{\frac{1}{2}}), \quad N \to \infty \tag{5.2.3}$$

where N is the size of the matrix. Note that, using finite elements on our problem, we would expect to have $\kappa(A) = O(N^2)$. Since the number of iterations to convergence of the conjugate gradient algorithm is proportional to $\sqrt{\kappa}$, this can be a very powerful form of preconditioner.

Another approach to improve the effectiveness of the factorisation is to allow a certain degree of fill-in to take place, using some pre-determined strategy. One simple approach to this is to select some area within the envelope of fill-in, based perhaps on knowledge of the structure of the matrix. This method is often denoted by ICCG(n) or ICCG(n,m), as in [68] and [69, §2.1.2 - §2.1.6], where n and m denote the levels of extra fill-in allowed by the algorithm, perhaps determined by knowledge of the sparsity pattern. A second, and perhaps more general approach, is to discard only those fill-in entries that lie outside

some given tolerance, for example as compared to the size of the corresponding diagonal entry. This tolerance is sometimes referred to as a *drop tolerance*, see Axelsson[13, §7.1]. By varying the tolerance the degree of fill-in can be controlled. The major disadvantage with this method is that the amount of required storage, or indeed the sparsity pattern of the storage, cannot be easily predicted prior to performing the factorisation.

Algorithm 5.1, given below, is a typical incomplete factorisation algorithm, taken from [14]. The set of indices J specifies the sparsity of the incomplete factorisation. Clearly setting $J \supseteq S$, where S is the envelope of *fill-in* given in (5.2.2), would give complete LU factors of A. Conversely, if $J \subset S$ then, in general, the LU factors are incomplete. The simplest *zero fill-in* choice involves setting $J = \{(i,j); A(i,j) \neq 0\}$, this is the choice we shall use for our implementation.

Algorithm 5.1 [A =MILU(A, J)] This algorithm overwrites the $N \times N$ sparse matrix A with a modified incomplete factorisation of A. Where fill-in, outside of the set J, would have taken place the entry is instead added to the relevant diagonal.

```
for r = 1, N − 1
   d = A(r, r)
   for i = r + 1, N
      if (i, r) ∈ J and A(i, r) ≠ 0 then
         e = A(i, r)/d
         A(i, r) = e
         for j = r + 1, N
            if (r, j) ∈ J and A(r, j) ≠ 0 then
               if (i, j) ∈ J then
                  A(i, j) = A(i, j) − e × A(r, j)          (5.2.4)
               else
                  A(i, i) = A(i, i) − e × A(r, j)          (5.2.5)
               end
            end
         end
      end
   end
end
```

Note that (5.2.5) corresponds to the simple modification of adding entries, that would

normally have been discarded, to the diagonal.

5.2.3 Stability of ILU : M-matrices and Non M-matrices

Two important issues with the incomplete factorisations are:

(a) Is the factorisation process *numerically stable*?

(b) Does the factorisation exist? By this we mean that the algorithm does not break down, i.e. $\hat{L}\hat{U}$, exists in the sense that none of the diagonal entries in the factorisation ever become 0.

For (a), the idea of stability is taken to mean that the incomplete factorisation is equal to an exact factorisation of the original problem perturbed by some small amount. We introduce the concept of a "growth factor"[14, §1.4,p42], defined by

$$q =: \max_{i,j,r} |a_{ij}^{(r)}| / \max_{ij} |a_{ij}| \geq 1. \tag{5.2.6}$$

Here $a_{ij}^{(r)}$ is the ij^{th} entry in the partially factored matrix A, created after r pivot steps of the incomplete factorisation process. Stability is now defined by requiring that the "growth factor" remains reasonably bounded.

We illustrate this concept with the following simple example.

Example 5.2

$$A = \begin{bmatrix} 10^{-7} & -10^{-4} & -10^{-4} \\ -10^{-4} & 1 & 0 \\ -10^{-4} & 0 & 1 \end{bmatrix}.$$

Note that this matrix is symmetric positive definite, but is not diagonally dominant although it is an M-matrix. For convenience, we have written the combined upper and lower triangular LU factors as one matrix, i.e. L+U (the diagonal of L is not considered

in this as they are all ones). A full LU decomposition of this matrix would be given by

$$\begin{bmatrix} 10^{-7} & -10^{-4} & -10^{-4} \\ -10^{3} & 0.9 & -0.1 \\ -10^{3} & -0.1111 & 0.8889 \end{bmatrix}.$$

Apply an ILU factorisation of the form given in algorithm 5.1, with no modification and no fill-in, to the matrix A. The first, and last ILU step (since the second step would merely have produced fill-in), produces

$$A^{(1)} = \begin{bmatrix} 10^{-7} & -10^{4} & -10^{-4} \\ -10^{3} & 0.9 & 0 \\ -10^{3} & 0 & 0.9 \end{bmatrix}.$$

Giving, in this case, a growth factor $q = 10^3/1 = 1000$.

The stability of the incomplete factorisation is closely related to the stability of the full decomposition, performed without any pivoting, a fact which is discussed in more detail in [68].

The standard theory for the stability of the incomplete factorisation methods is generally given for specific classes of matrices. For example, Axelsson and Barker[14] give the majority of their theory for a class of matrices that they call *diagonally dominant* $\hat{M}$ matrices.

Firstly, recall that diagonal dominance was defined in definition 1.1. We also introduce the following notation for any matrix A such that $a_{ii} \neq 0$, $i = 1, ..., N-1$,

$$n(i) = \max\{j; (1 \leq j \leq N) \cap (a_{ij} \neq 0)\}, \quad i = 1, ..., N-1.$$

Hence $a_{i,n(i)}$ is the last non-zero entry in the i^{th} row.

Then we have the following,

Definition 5.1 *An* $N \times N$ *matrix* A *is an* $\hat{M}$*-matrix if*

(1) $a_{ii} > 0, \quad i = 1, 2, ..., N-1; a_{NN} \geq 0.$

(2) $a_{ij} \leq 0, \quad i, j = 1, 2, ..., N, i \neq j.$

(3) $n(i) > i, \quad i = 1, ..., N-1.$

Note that this definition of an $\hat{M}$-matrix is similar to the more standard concept of an M-matrix, defined in §1.3. However, a matrix being an M-matrix does not necessarily imply that it is an $\hat{M}$-matrix, or vice versa. This is illustrated by the following examples

Example 5.3 *a) The matrix given by*

$$A = \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix},$$

is an $\hat{M}$-matrix, but is not an M-matrix, since it does not satisfy condition (3) of the definition.

b) The matrix given by

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix},$$

is an M-matrix, but is not an $\hat{M}$-matrix, since it does not satisfy condition (3) of the definition.

Axelsson and Barker then give the following result:

Theorem 5.2.1 (Thm 1.15, p44 of [14]) *The incomplete factorisation is a stable numerical process, (in the sense that the "growth factor", i.e. (5.2.6), is exactly 1), if A is a diagonally dominant $\hat{M}$-matrix.*

Proof See p44 of [14]. □

Other authors consider further special classes of matrices, Notay[70] considers incomplete factorisations of non-singular (or singular) Stieltjes matrices. These are positive

definite (or semidefinite) matrices with non-positive off-diagonal elements. Note, that a symmetric M-matrix is also a non-singular Stieltjes matrix, although an $\hat{M}$-matrix is not, necessarily, a Stieltjes matrix, as can be seen from example 5.3. Meijerink and van der Vorst[68] also give theory and results for M-matrices, in particular they give the following theorem on p152,

Theorem 5.2.2 *If A is an M-matrix, then the construction of an incomplete LU decomposition is at least as stable as the construction of a complete decomposition $A = LU$, without any pivoting.*

The existence of the factorisation, i.e. point (b) above can also be proved for specific classes of matrices. For example Axelsson and Barker [14, Theorem 1.17] choose the following:

Theorem 5.2.3 *Let A be a diagonally dominant $\hat{M}$-matrix with symmetric structure, and suppose that at least one row sum is positive. Then the factorisation exists.*

Proof See [14, p47]. □

Kershaw [62, p48] does discuss incomplete factorisations of matrices that are merely symmetric positive definite, the algorithm he proposes dynamically modifies the factors, to ensure the factorisation does not break down, at each stage. It is unclear what effect this has on the resulting factorisation. Axelsson and Lindskog [15, p482] also briefly mention the existence of incomplete factorisations for M-matrices. Varga, Saff and Mehrmann[90] discuss a version of incomplete Cholesky factorisations for H-matrices, incorporating a graph technique for fill-in. The emphasis of their paper is on the "regularity", or existence, of the factorisation. Buoni[25] gives a condition for the existence of incomplete factorisations of M-matrices, his paper also discusses a method of fill-in using a matrix graph technique. Notay[71] discusses the robustness of the incomplete factorisation, particularly for diagonally dominant Stieltjes matrices.

For a general symmetric positive definite matrix there is nothing in the literature about the stability, or indeed existence, of the incomplete factorisation. In our case, as has been discussed in chapter 1, the possible use of basis functions, in the finite element

discretisation, that are not linear means that the resulting stiffness matrix cannot be guaranteed to be either diagonally dominant or an M matrix. In this case the construction of the incomplete factorisation can fail as a result of non-positive diagonal entries being produced by addition of negative values. In addition the creation of small positive diagonal entries can also cause stability problems in the factorisation. Three relatively standard approaches to this problem are given in Meijerink and van der Vorst[69, §3]:

(i) If a diagonal entry of less than a prescribed positive value is encountered during the construction of the LL^{T} decomposition then some already computed off-diagonal entries in the corresponding column of L^{T} are set to zero.

(ii) The diagonal entry is enlarged if necessary, e.g. by neglecting some of the Gaussian elimination corrections, (i.e. if a Gaussian elimination step would cause problems with a small or negative diagonal entry then the diagonal entry is left alone).

(iii) We can also add αI to the matrix, i.e. perform an ILU decomposition on $A + \alpha I$. If α is large enough then the problem will not occur.

The incomplete factorisation code used in this chapter, uses a simple version of approach (ii). Specifically, diagonal entries of the factorisation that are less than or equal to zero are simply set to unity, thereby avoiding the difficulty. Whilst this is perhaps not the best choice for our particular problem, it should be emphasised that the purpose of this chapter is to compare a relatively simple incomplete factorisation method with a method that is already known to be effective, namely the Two Grid method in chapter 2. In fact for the actual problem that we solve this criterion is not in fact ever used.

5.3 Matrix storage considerations

In previous chapters we considered the situation where the matrix was only required in order that matrix-vector multiplications could be carried out. As a result of this the matrix could be stored in an element form, precisely as was produced by the finite element code. This had a number of advantages, as was seen in chapter 1, most notably the savings in storage for the Cray YMP and the speed of the matrix-vector multiplications, which on the Cray could be made to vectorise very well.

However, in the case of the incomplete factorisation, the matrix is required to be fully assembled in order to carry out the incomplete factorisation. Hence, in order to implement any incomplete factorisation preconditioner, it is necessary to (a) decide on a sparse storage technique and (b) using the storage technique, produce efficient versions of both a matrix-vector multiplication routine and an incomplete factorisation routine.

For the purposes of this chapter, at least initially, the main interest is whether using the incomplete factorisation is sufficient to ensure that the preconditioned conjugate gradient method can be made to converge satisfactorily. For this reason it was decided to use a standard package that could perform an incomplete factorisation of a general matrix and use it as a preconditioner for CG. The Sparse Linear Algebra Package (SLAP), discussed in detail in [82], was found to provide the necessary routines. The matrix is supplied to these routines in a particularly simple format, called the SLAP Triad Format. In this format each entry in the matrix requires two integers to specify the column and row, and one real value for the entry itself. Note that the entries can appear in any order, so this form is very simple to set up. More complete details can be found in [82, §10.2]. In fact this particular storage format is not very effective on the Cray YMP, as it does not allow the multiply routine to vectorise at all, which will seriously limit the performance. However, the standard routines are designed to convert the SLAP triad storage format to another method called the SLAP column format, where the entries appear in a more ordered fashion, sorted by columns. This format allows a more efficient computation of both the matrix-vector multiplies and the LU backsolves since the operations now lend themselves more readily to vectorisation. We illustrate these storage formats with the following example, which is non-symmetric to fully illustrate the format.

Example 5.4 *Start with the original* 5×5 *matrix*

$$\begin{bmatrix} 11 & 12 & 0 & 0 & 15 \\ 21 & 22 & 0 & 0 & 0 \\ 0 & 0 & 33 & 0 & 35 \\ 0 & 0 & 0 & 44 & 0 \\ 51 & 0 & 53 & 0 & 55 \end{bmatrix}$$

The SLAP Triad format could store this matrix as follows:

A:	51	12	11	33	15	53	55	22	35	44	21
IA:	5	1	1	3	1	5	5	2	3	4	2
JA:	1	2	1	3	5	3	5	2	5	4	1

Conversely, the SLAP column format would store the matrix as:

A:	11	21	51	22	12	33	53	44	55	15	35
IA:	1	2	5	2	1	3	5	4	5	1	3
JA:	1	4	6	8	9	12					

Notice that for the column storage format, the columns are sorted by order, but with the diagonal appearing first, the "IA" entry specifies the row for each entry and the "JA" entry is used to determine where each column starts, the last "JA" entry specifies the number of non-zero entries in A.

One of the main problems with this method of storage is that it is not immediately clear, at the time of starting to assemble the matrix in sparse format, just how much storage will be required, this being a consequence of converting from an element format. Again this is one of the drawbacks of using sparse storage techniques.

It is clear that, unlike the element storage technique, integer storage requirements are much higher with the sparse storage, as pointers are needed for each entry. On architectures where integers require the same storage space as reals, such as the Cray YMP, this is an important issue, which we now explore.

5.3.1 An example of storage requirements

To illustrate the storage requirements for the two methods, we will briefly consider a relatively small 2D problem with 14736 degrees of freedom. The grid and finite element discretisation used results in 3684 9x9 element matrices, which when fully assembled give a stiffness matrix with 122992 non-zero entries. In table 5.1 a comparison of storage for the matrix is made, between both the SLAP Triad format, the SLAP column format

and the element storage format. Note that the sparse scheme only stores the lower triangle of the matrix, but the element scheme stores each entire element and therefore the entire matrix. It is clear that, for this problem, the Triad format requires most storage, then the element scheme and finally the Column scheme. However, it should be made clear that, as is discussed in chapter 1, larger problems always require more storage for sparse schemes of the type discussed here. This is mainly due to the high numbers of integer pointers required for the sparse format.

Method	Element	SLAP Triad	SLAP Column
REALS	298404	122992	122992
INTS	33156	245984	137728
TOTAL	331560	368976	260720

Table 5.1: *Table showing storage requirements for the stiffness matrices for element and sparse matrix storage schemes. Note that this problem features 3684 elements each requiring a* 9×9 *matrix of reals and* 9 *integers for the element scheme. The sparse scheme only stores the lower triangle of this symmetric matrix, which has 122992 non-zero entries.*

Now consider the situation for the incomplete factorisation and compare this with the requirements for the two grid method. Storage of the incomplete factorisation clearly doubles the requirement, since we now have to store the matrix and its incomplete factorisation. For the two grid method it is only necessary to store the matrix, the rest can be reasonably stored out of main memory with no significant performance loss, as this data is only needed for the coarse grid solve step, see chapter 2 for more details on this. In table 5.2 the storage requirements for the two grid method are compared with those for the incomplete factorisation. It is clear from this that the incomplete factorisation method requires more storage than the two grid method even for a small problem like this one. As discussed in chapter 1 §1.5 the sparse scheme will always require more storage than the element scheme, so that use of the incomplete factorisation, where the matrix is effectively stored twice, always has a higher storage requirement.

Method	Two Grid	ILU
REALS	298404	260720
INTS	33156	260720
TOTAL	331560	521440

Table 5.2: *Table of total storage requirements for the two grid method and incomplete factorisation method, neglecting CG vector requirements. The two grid method only requires storage of the stiffness matrix (as elements) whilst the ILU method requires storage of the stiffness matrix and the incomplete factorisation (both in sparse format).*

5.4 Numerical Experiments

For the results presented here we consider a model problem for groundwater flow in a complex geological medium with realistic contrasts in the permeability field. A sample grid, with only a few thousand degrees of freedom, is given in chapter 2, figure 1-1.

Comparisons are made between the sparse and element matrix storage schemes and the direct solver, the MA32 frontal solver from the Harwell subroutine library. Where appropriate, time plot graphs will show the time taken by the direct solver with a "∇".

The sample problem considered here has 14736 degrees of freedom, the direct solver requires 4.3 seconds to solve this problem exactly. Note that, for this problem, use of a simple diagonal scaling preconditioner is not sufficient to achieve reasonable rates of convergence. Figure 5-4, shows the rate of convergence of diagonally preconditioned conjugate gradients, using the element storage scheme. Although the method converges quite well for this problem, nearly 10 times more CPU time is used than the direct solver. As this is a relatively small problem this is not too discouraging a result, since if this rate of convergence were maintained for larger problems then the iterative solver is highly likely to be significantly faster than the direct solver. Nonetheless, in order for the incomplete factorisation preconditioner to be considered effective, we would expect to see a marked improvement in the rate of convergence.

When considering the sparse storage scheme, and the incomplete factorisation, there is an additional timing overhead involved in assembling the matrix into the sparse format and computing the incomplete factorisation. Indeed for the problem considered here, 62.48 seconds are required to form the factorisation, which already is considerably more

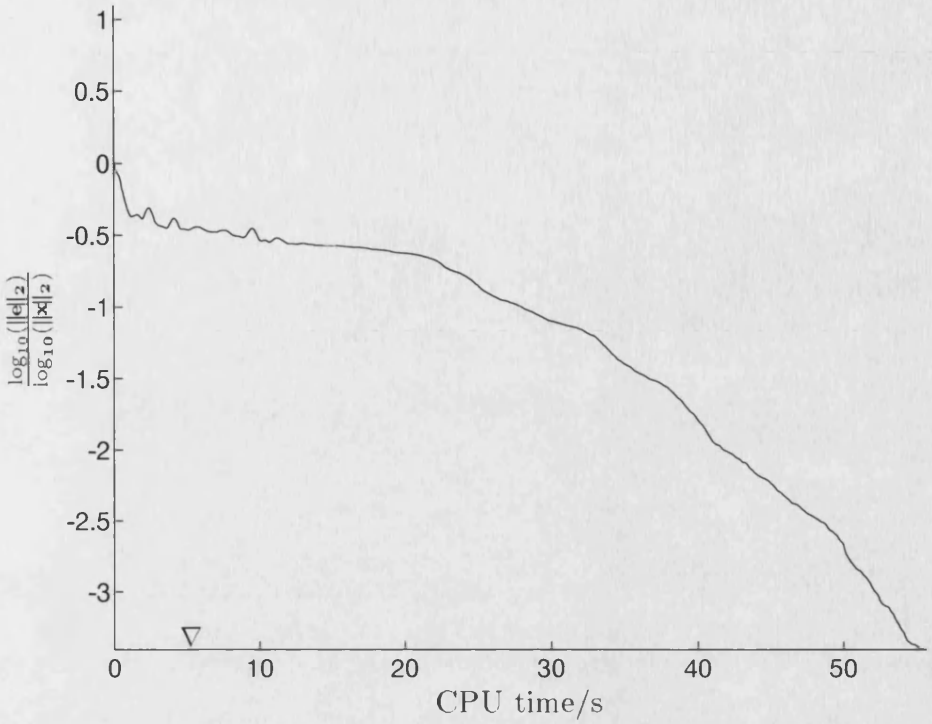

Figure 5-4: *Diagonally preconditioned conjugate gradients, using element matrix storage.*

time than the direct solver needs to exactly solve the problem. This time should, to some extent, be ignored since the implementation does not appear to be very effective on the Cray YMP.

To see fully the effect on performance due to using the sparse storage scheme we firstly compare the results for a diagonally preconditioned conjugate gradient method using both the element scheme and the sparse scheme. In terms of iterations we would expect these to give almost identical results, since they are both working on the same problem. Any difference would be wholly due to numerical difference in the two methods due to rounding errors. Results in figure 5-5 demonstrate that the element scheme is superior, in terms of error reduction achieved for processor time used. In this case the time to compute the assembled sparse matrix is neglected. The sparse storage scheme takes approximately 3.5 times longer to perform the same number of iterations as the element storage scheme.

In figure 5-6 a comparison of diagonal preconditioning and incomplete LU preconditioning is made, based on error reduction against iterations. It is clear from this that the

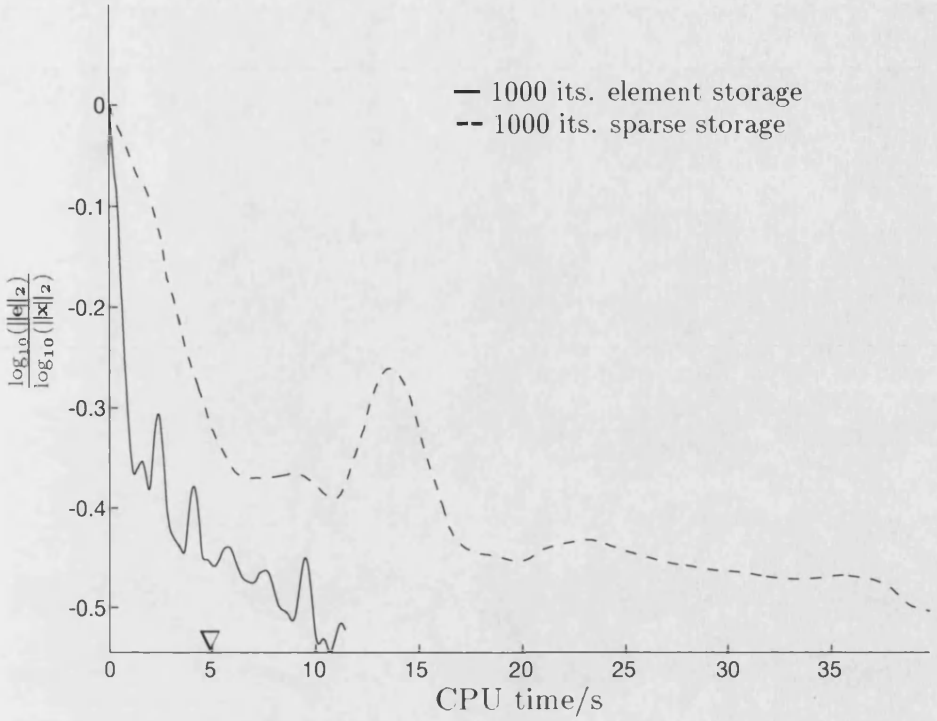

Figure 5-5: *Comparison of timings for diagonal preconditioned CG, with element and sparse storage schemes, using 1000 iterations.*

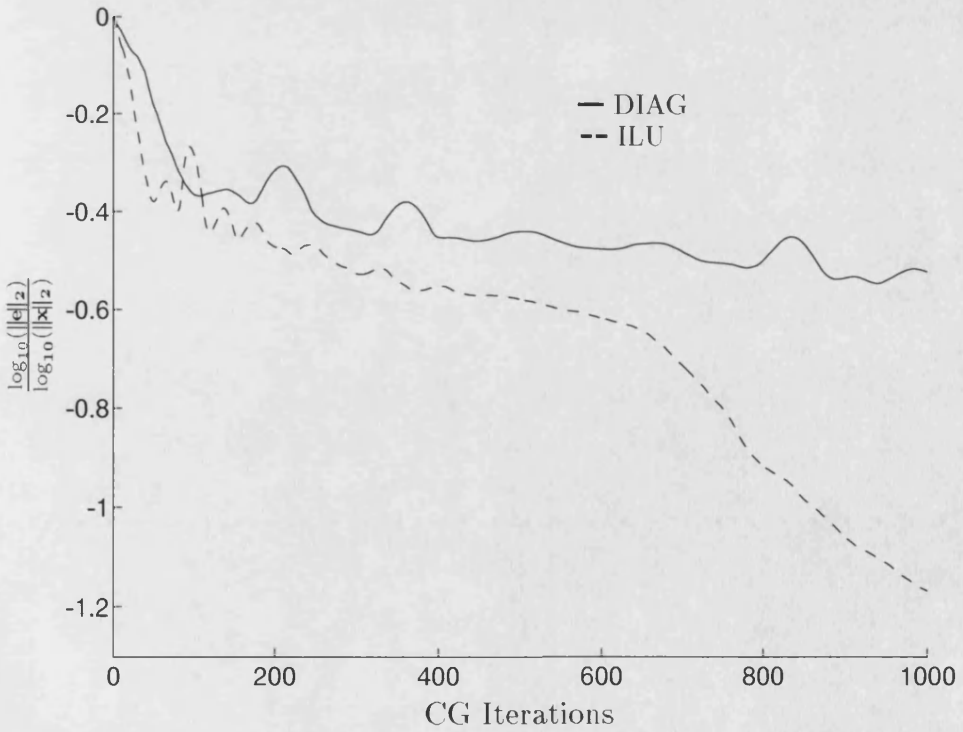

Figure 5-6: *Comparison of diagonal preconditioned CG and incomplete LU preconditioned CG.*

incomplete factorisation does indeed give faster convergence, but not significantly faster as might perhaps be hoped. A comparison of error reduction against CPU time used is given in figure 5-7. Here we see nearly 5000 iterations of diagonally preconditioned CG, in the element storage format, compared with 1000 iterations of the incomplete method. Note that in this figure the sparse matrix and incomplete factorisation setup times are neglected. Figure 5-8 again shows the same results as the previous figure but now the setup times for the ILU method are included. Note the considerable difference between this and the previous figure, nearly a factor of 2.

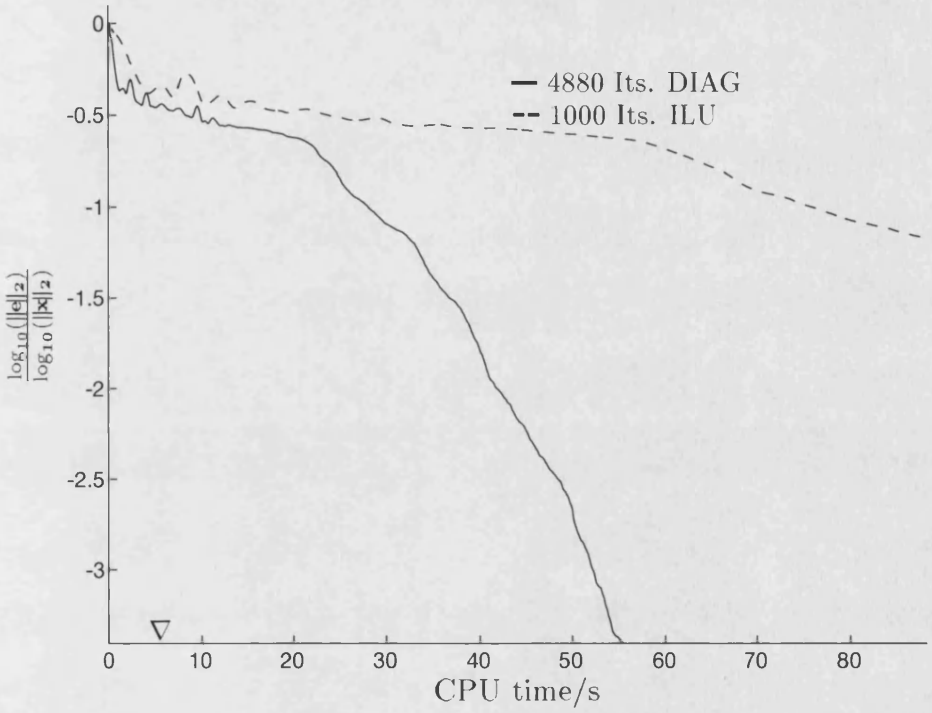

Figure 5-7: *Comparison of timings for element stored diagonal preconditioned CG and incomplete LU preconditioned CG.*

We finally comment on how this method compares with the two grid method from chapter 2. For the problem of this size the two grid method is able to achieve convergence in almost the same time as the direct solver. This is a considerably better situation than we have for the incomplete factorisation method, which, after this same amount of CPU time, had barely even begun to converge. Figure 5-9 shows how the two grid and ILU methods compare, in terms of convergence against CPU time.

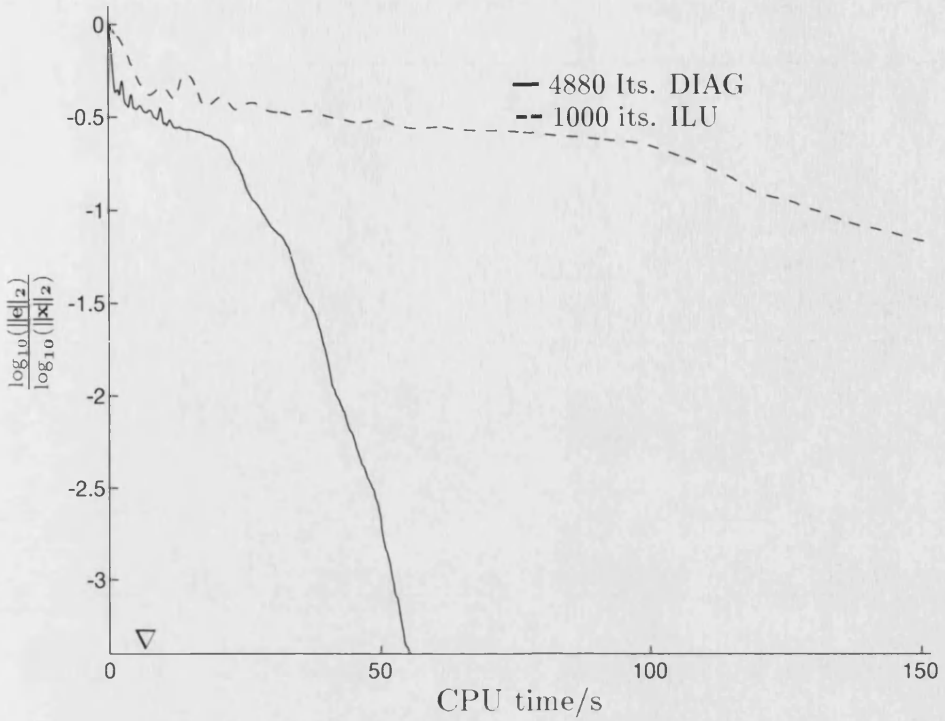

Figure 5-8: *Comparison of timings for element stored diagonal preconditioned CG and incomplete LU preconditioned CG, including setup times for the ILU version.*

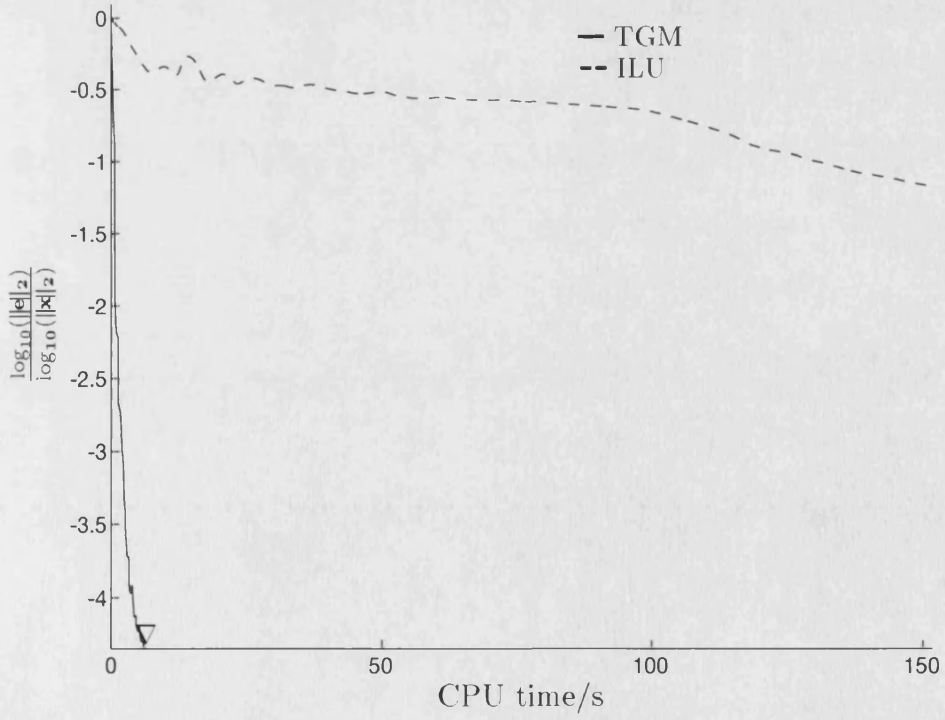

Figure 5-9: *Comparison of timings for two grid method and incomplete LU preconditioned CG, including setup times for the ILU version.*

5.5 Element-by-element preconditioning

We conclude this chapter by briefly discussing a further approach to incomplete factorisation preconditioning, which would seem to by-pass at least some of the problems that were discussed in the previous sections. In this approach an LU factorisation of each element is produced and these are then combined to form a preconditioner for the conjugate gradient method. This results in an element-by-element preconditioned conjugate gradient method, or EBEPCG. For more details on these methods see work by Hughes et al[54, 53, 52, 51], and Wathen[66, 92].

As an example of an element-by-element factorisation, using a similar approach to that given in [92, §2], we see that the method proceeds in the following way: Assume the fully assembled stiffness matrix, A, is assembled from small element matrices E_e. Then denote A^e as the assembled contribution to the full matrix from the element e, hence

$$A = \sum_e A^e. \tag{5.5.1}$$

Now write each A^e as an LU decomposition

$$A^e = L^e U^e \tag{5.5.2}$$

and so the global element-by-element preconditioner is written as

$$M = \left[\prod_{e=1}^{n_{\text{el}}} L^e\right] \left[\prod_{e=1}^{n_{\text{el}}} U^e\right], \tag{5.5.3}$$

where n_{el} is the total number of elements.

Clearly, with this approach, use of the preconditioner, M, to solve $M\mathbf{z} = \mathbf{r}$ involves a series of forward and back substitutions.

This method would certainly appear to solve some of the problems that have been mentioned earlier, in particular we no longer need to fully assemble the matrix and so can avoid having to use the slower, and less efficient, sparse storage format. However, as is discussed in some detail by Lee and Wathen in [66, §4], problems arise with EBE ap-

proaches when there is a discontinuous coefficient in the pde. In particular they report that the EBE preconditioning becomes increasingly worse, in the sense that the condition number of the preconditioned matrix is poorly reduced, as the discontinuity grows worse. They conclude by saying that "For problems where coefficient discontinuities do not play a major role, it (EBEPCG) appears to be a competitive method, particularly on advanced hardware.". Unfortunately, as has been discussed in detail in chapter 1, the pde we are mainly concerned with here is of the form

$$-\nabla.(k\nabla p) = 0$$

where the coefficient k is highly discontinuous. Hence, following the advice in [66], the EBE method would appear to be of no practical use. For this reason we chose not to develop and test code for the EBE approach.

5.6 Conclusions

From the results in §5.4 it can be seen that use of the sparse storage scheme, i.e. the SLAP column format, severely limits the performance of the conjugate gradient code. With the stiffness matrix stored in the fully assembled sparse format, a single matrix-vector multiply, which is the core routine of the conjugate gradient code, requires 3.5 times as much CPU time as when the matrix is stored by elements. In addition the time to perform the incomplete factorisation appears to be very high, nearly 65 seconds for a problem with 14,000 degrees of freedom, indicating that the implementation is not very effective for the Cray YMP architecture. Storage requirements for the incomplete factorisation are also much higher, which is partly a result of using a sparse storage technique and partly due to needing additional storage for the incomplete factors.

Our test example, with 14,000 degrees of freedom, shows very poor performance of a relatively simple incomplete factorisation code. Whilst convergence of the conjugate gradient algorithm is faster than for the simple diagonal preconditioning, the extra time required to perform the incomplete factorisation, coupled with the much slower matrix-vector multiply performance, results in a method that hardly begins to be competitive with the direct solver, MA32. In addition the ILU method does not compare well

with the Two Grid method, which, for this problem, converges in the same time as the direct solver. Other, more complicated, techniques for forming the incomplete factorisation exist, as was discussed in §5.2, and it is possible that use of these methods, coupled with a better implementation, may allow a better performance of the incomplete factorisation. However, the main point of this chapter was to investigate the effectiveness (or non-effectiveness) of a simple, standard, ILU code. Our results indicate that a simple implementation is not sufficient.

Chapter 6

Future Work

The results in chapters 2 and 3 indicate that the two grid method is very effective at solving the linear systems arising from the groundwater flow problem (see §1.2). A more complete evaluation of the two grid method for much larger problems is needed, mainly to confirm that there are no problems with the convergence of the method for larger problems. A more thorough investigation of the effect of the number of smoothing steps per two grid step is also needed, our numerical experience so far indicates that larger (or harder) problems need more smoothing steps to give best rates of convergence. The three grid method would also benefit from testing on large problems, again to see how it converges in this case and contrasts with the two grid method. A W-cycle implementation of the three grid method would also be of some interest.

The non-symmetric problem briefly discussed in §2.7 needs much more consideration. A successful implementation of a non-symmetric two grid solver would be of considerable interest to Harwell, since many of the NAMMU applications produce non-symmetric linear systems (including the non-linear pdes which are solved by Newton's method). Further approaches to finding an effective method would certainly include considering other non-symmetric iterative methods, for example Bi-CG, GMRES, or QMR. The non-symmetric case might also benefit from trying a three grid approach, again with the various choices of smoothers, and using a V-cycle or W-cycle approach. A further possible area of research concerns the coarse grid correction step, the natural choice

for prolongations and restrictions for a finite element method outlined in §2.2 is not the only possible choice, for example we could use a higher order interpolation to give the prolongation matrix. It is unclear whether this would be advantageous. A further possibility, only appropriate in the three grid case, is to use a damped coarse grid correction, where we replace the coarse grid correction step (c.f 2.2.4) by

$$\mathbf{x}_{\frac{1}{2}} = \mathbf{x}_0 + \omega P \mathbf{d}_c,$$

for some damping coefficient ω. This approach is discussed in more detail in Hackbusch [44, §10.8.2].

Bibliography

[1] L. ADAMS, *M-step preconditioned conjugate gradient methods*, SIAM J. on Scientific and Statistical Computing, 6 (1985), pp. 452–463.

[2] F. ALVARADO AND H. DAĞ, *Sparsified and incomplete sparse factored inverse preconditioners.* Proc. Copper Mountain Conference on Iterative Methods, April 1992.

[3] S. F. ASHBY, *Polynomial Preconditioning for Conjugate Gradient methods*, PhD thesis, University of Illinois, 1987.

[4] ——, *Minimax polynomial preconditioning for Hermitian linear systems*, SIAM J. on Matrix Anal. and Appl., 12 (1991), pp. 766–789.

[5] S. F. ASHBY, R. D. FALGOUT, S. G. SMITH, AND A. F. B. TOMPSON, *Modeling groundwater flow on MPPS*, in Proceedings of the scalable parallel libraries conference, 1993.

[6] S. F. ASHBY AND M. H. GUTKNECHT, *A matrix analysis of conjugate gradient algorithms*, Tech. Report UCRL-JC-113560, Lawrence Livermore National Laboratory, March 1993.

[7] S. F. ASHBY, M. J. HOLST, T. A. MANTEUFFEL, AND P. E. SAYLOR, *The role of the inner product in stopping criteria for conjugate gradient iteration*, Tech. Report UCRL-JC-112586, Lawrence Livermore National Laboratory, November 1992.

[8] S. F. ASHBY, T. MANTEUFFEL, AND J. S. OTTO, *A comparison of adaptive Chebyshev and least squares polynomial preconditioning for Hermitian positive definite linear systems*, SIAM J. on Scientific and Statistical Computing, 13 (1992), pp. 1–29.

[9] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, *Adaptive polynomial preconditioning for Hermitian indefinite linear systems*, BIT, 29 (1989), pp. 583–609.

[10] ——, *A taxonomy for conjugate gradient methods*, SIAM J. on Numer. Anal., 27 (1990), pp. 1542–1568.

[11] O. Axelsson, *A survey of preconditioned iterative methods for linear systems of algebraic equations*, BIT, 25 (1985), pp. 166–187.

[12] ——, *Bounds of eigenvalues of preconditioned matrices*, SIAM J. on Matrix Anal. and Appl., 13 (1992), pp. 847–862.

[13] ——, *Iterative solution methods*, CUP, 1994.

[14] O. Axelsson and V. A. Barker, *Finite element solution of boundary value problems - theory and computation*, Academic press, 1984.

[15] O. Axelsson and G. Lindskog, *On the eigenvalue distribution of a class of preconditioning methods*, Numer. Maths., 48 (1986), pp. 479–498.

[16] ——, *On the rate of convergence of the preconditioned conjugate gradient method*, Numer. Maths., 48 (1986), pp. 499–523.

[17] O. Axelsson and N. Munksgaard, *A class of preconditioned conjugate gradient methods for the solution of a mixed finite element discretization of the biharmonic operator*, Internat. J. Numer. Methods Engrg., 14 (1979), pp. 1001–1019.

[18] R. E. Bank, *A multi-level iterative method for nonlinear elliptic equations*, in Elliptic Problem Solvers, M. H. Schultz, ed., Academic Press, New York, 1981, pp. 1–16.

[19] R. E. Bank and C. C. Douglas, *Sharp estimates for multigrid rates of convergence with general smoothing and acceleration*, SIAM J. on Numer. Anal., 22 (1985).

[20] R. Beauwens and M. B. Bouzid, *On sparse block factorization iterative methods*, SIAM J. on Numer. Anal., 24 (1987).

[21] P. Beckmann, *Orthogonal polynomials for engineers and physicists*, Golem Press, 1973.

[22] P. BOGARINSKI, ED., *INTRAVEL Phase II. working group 3 report*, tech. report, Nuclear Energy Agency, 1994. In preparation.

[23] D. BRAESS, *On the combination of the multigrid method and conjugate gradients*, in Multigrid Methods II, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, 1985.

[24] A. BRANDT, *Multi-level adaptive solution to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

[25] J. J. BUONI, *Incomplete factorizations of singular M-matrices*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 193–198.

[26] N. N. CHAN AND K.-H. LI, *Diagonal elements and eigenvalues of a real symmetric matrix*, J. Math. Anal. and Appl., 91 (1983), pp. 562–566.

[27] M. T. CHU, *A simple application of the homotopy method to symmetric eigenvalue problems*, Linear Algebra and its applications, 59 (1984), pp. 85–90.

[28] P. G. CIARLET, *The finite element method for elliptic problems*, North-Holland, 1978.

[29] P. CONCUS, G. H. GOLUB, AND G. MEURANT, *Block preconditioning for the conjugate gradient method*, SIAM J. on Scientific and Statistical Computing, 6 (1985), pp. 220–252.

[30] P. M. DE ZEEUW, *Matrix-dependent prolongations and restrictions in a blackbox multigrid solver*, Journal of computational and applied mathematics, 33 (1990), pp. 1–27.

[31] H. DECONINCK AND C. HIRSCH, *A multigrid finite element method for the transonic potential equation*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, 1981.

[32] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct methods for sparse matrices*, Oxford University Press, 1986.

[33] B. FISCHER AND R. W. FREUND, *On adaptive weighted polynomial preconditioning for Hermitian positive definite matrices.* To appear.

[34] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, in Lecture Notes in Mathematics 506, Springer-Verlag, Berlin, Heidelberg, New York, 1976, pp. 73–89.

[35] G. E. FORSYTHE AND E. G. STRAUS, *On best conditioned matrices*, Proc. Amer. Math. Soc., 6 (1955), pp. 340–345.

[36] L. FOX AND D. F. MAYERS, *Computing methods for scientists and engineers*, Oxford, 1968.

[37] G. H. GOLUB AND D. P. O'LEARY, *Some history of the conjugate gradient and Lanczos algorithms: 1948-1976*, computer science technical report series, University of Maryland, June 1987.

[38] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations, second edition.*, John Hopkins University Press, 1989.

[39] A. GREENBAUM, *A comparison of splittings used with the conjugate gradient algorithm*, Numer. Maths., 33 (1979), pp. 181–194.

[40] ——, *Diagonal scalings of the Laplacian as preconditioners for other elliptic differential operators*, SIAM J. on Matrix Anal. and Appl., 13 (1992), pp. 862–846.

[41] A. GREENBAUM, C. LI, AND H. Z. CHAO, *Parallising PCG algorithms*, Comp. Phys. Comm., 53 (1989), pp. 295–309.

[42] I. GUSTAFSSON, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[43] W. HACKBUSCH, *Multigrid Methods and Applications*, Springer-Verlag, 1985.

[44] ——, *Iterative Solution of Large Sparse Systems of Equations*, Springer-Verlag, 1994.

[45] W. HACKBUSCH AND U. TROTTENBURG, eds., *Multigrid Methods*, Springer-Verlag, 1982. From the International Conference on Multigrid Methods, Cologne-Porz.

[46] ——, eds., *Multigrid Methods II*, Springer-Verlag, 1986. From the 2nd European Conference on Multigrid Methods, held at University of Cologne.

[47] L. A. HAGEMAN AND D. M. YOUNG, *Applied iterative methods*, Academic Press, 1981.

[48] L. J. HARTLEY AND C. P. JACKSON, *NAMMU (Release 6.1) User Guide*, AEA-D&R 0472.

[49] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[50] M. C. HILL, *Solving groundwater flow problems by conjugate-gradient methods and the strongly implicit procedure*, Water resources research, 26 (1990), pp. 1961–1969.

[51] T. J. R. HUGHES, *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[52] T. J. R. HUGHES, R. M. FERENCZ, AND J. O. HALLQUIST, *Large scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients*, Comp. Meth. Appl. Mech. Engrg., 61 (1987), pp. 215–248.

[53] T. J. R. HUGHES, I. LEVIT, AND J. WINGET, *Element-by-element implicit algorithms for heat conduction*, J. Engrg. Mech., 109 (1983), pp. 576–585.

[54] ——, *An element-by-element solution algorithm for problems of structural and solid mechanics*, Comp. Meth. Appl. Mech. Engrg., 36 (1983), pp. 241–254.

[55] B. M. IRONS, *A frontal solution program for finite-element analysis*, Int. J. Numer. Meth. Engr., 2 (1970), pp. 5–32.

[56] C. JOHNSON, *Numerical solution of Partial Differential Equations by the Finite Element method*, CUP, 1987.

[57] O. G. JOHNSON, C. A. MICCHELLI, AND G. PAUL, *Polynomial preconditioners for conjugate gradient calculations*, SIAM J. on Numer. Anal., 20 (1983), pp. 362–376.

[58] T. L. JORDAN, *Conjugate gradient preconditioners for vector and parallel processors*, 1984.

[59] E. F. KAASCHIETER, *A practical termination criterion for the conjugate gradient method*, BIT, 28 (1988), pp. 308–322.

[60] ——, *Preconditioned conjugate gradients for solving singular systems*, J. Comp. and Appl. Maths, (1988), pp. 265–275.

[61] ——, *A general finite element preconditioning for the conjugate gradient method*, BIT, 29 (1989), pp. 824–849.

[62] D. S. KERSHAW, *The incomplete Cholesky - conjugate gradient method for the iterative solution of systems of linear equations*, J. Comput. Phys., 26 (1978), pp. 43–65.

[63] R. KETTLER, *Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, 1981.

[64] J. R. KNIGHTLY AND I. P. JONES, *A comparison of conjugate-gradient preconditionings for 3-d problems on a Cray-1*, Computer Physics Communications, 37 (1985), pp. 205–214.

[65] C. LANCZOS, *Chebyshev polynomials in the solution of large-scale linear systems.* Proc. of the Association for Computing Machinery, Toronto, Sauls Lithograph Co., Washington D.C, 1953.

[66] H. LEE AND A. J. WATHEN, *On element-by-element preconditioning for general elliptic problems*, Comp. Meth. in Appl. Mechanics and Engineering, 92 (1991), pp. 215–229.

[67] M. M. MAGOLU, *Modified block-approximate factorization strategies*, Numer. Maths., 61 (1992), pp. 91–110.

[68] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of computation, 31 (1977), pp. 148–162.

[69] ——, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comp. Phys., 44 (1981), pp. 134–155.

[70] Y. NOTAY, *Incomplete factorizations of singular linear systems*, BIT, 29 (1989), pp. 682–702.

[71] ——, *On the robustness of modified incomplete factorization methods*, Int. J. Comp. Meth., 40 (1992), pp. 121–141.

[72] D. P. O'Leary, *Yet another polynomial preconditioner for the conjugate gradient algorithm*, Linear Algebra and its applications, 154 (1991), pp. 377–388.

[73] B. N. Parlett, *The symmetric eigenvalue problem*, Prentice Hall, 1980.

[74] A. Ramage and A. J. Wathen, *On preconditioning for finite-element equations on irregular grids*, SIAM J. on Matrix Anal. and Appl., 15 (1994), pp. 909–921.

[75] J. K. Reid, *On the method of conjugate gradients for the solution of large sparse systems of linear equations*, in Large sparse sets of linear equations, J. K. Reid, ed., Academic Press, 1971.

[76] T. J. Rivlin, *The Chebyshev polynomials*, Wiley, 1974.

[77] V. Ruggiero, *Polynomial preconditioning on vector computers*, Applied mathematics and computation, 59 (1993), pp. 131–150.

[78] Y. Saad, *Practical use of polynomial preconditioning for the conjugate gradient method*, SIAM J. on Scientific and Statistical Computing, 6 (1985).

[79] ——, *Krylov subspace methods on supercomputers*, SIAM J. on Scientific and Statistical Computing, 10 (1989), pp. 1200–1232.

[80] Y. Saad and M. H. Shultz, *GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM J. on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[81] S. Sauter, *The ILU method for the finite-element discretisations*, J. Comp. Appl. Maths., 36 (1991), pp. 91–106.

[82] M. K. Seager, *A SLAP for the masses*, in Parallel Supercomputing: Methods, Algorithms and Applications, G. F. Carey, ed., Wiley, 1989, ch. 10, pp. 135–155.

[83] G. Strang, *Introduction to applied mathematics*, Wellesley Cambridge Press, 1986.

[84] G. Szegö, *Orthogonal polynomials*, vol. XXIII, American Mathematical Society Colliquiem publications, 3rd ed., 1967.

[85] M. TISMENETSKY, *A new preconditioning technique for solving large sparse linear systems*, Linear Algebra and its applications, 154-156 (1991), pp. 337–353.

[86] A. VAN DER SLUIS, *Condition numbers and equilibration of matrices*, Numer. Maths., 14 (1969), pp. 14–23.

[87] A. VAN DER SLUIS AND H. A. VAN DER VORST, *Rate of convergence of conjugate gradients*, Numer. Maths., 48 (1986), pp. 543–560.

[88] H. A. VAN DER VORST, *Conjugate gradient type methods and preconditioning*, J. of Computational and Applied Maths, 24 (1988), pp. 73–87.

[89] R. S. VARGA, *Matrix Iterative Analysis*, Prentice Hall, 1962.

[90] R. S. VARGA, E. B. SAFF, AND V. MEHRMANN, *Incomplete factorization of matrices and connections with H-matrices*, SIAM J. on Numer. Anal., 17 (1980), pp. 787–793.

[91] A. J. WATHEN, *Realistic eigenvalue bounds for the Galerkin mass matrix*, IMA J. Numer. Anal., 7 (1978), pp. 449–457.

[92] ——, *An analysis of some element-by-element techniques*, Comp. Meth. Appl. Mech. Engrg., 74 (1989), pp. 271–287.

[93] J. H. WILKINSON, *The algebraic eigenvalue problem*, Oxford, 1965.