Preconditioned Conjugate Gradient Methods for Three Dimensional Linear Elasticity

by

John Kenneth Dickinson

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1993

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

A finite element modelling of three dimensional elasticity problems gives rise to large sparse matrices. To improve upon direct solution methods, various new preconditioning methods are developed and examined, as well as some generally standard techniques, for use in preconditioned conjugate gradient iterative solution techniques. Developments of incomplete factorizations based on levels of fill, drop tolerance, and a two level hierarchical basis are used to build the preconditioning matrices. The problem of non-positive pivots occurring during factorization is also addressed by the use of several techniques. Computational tests are carried out for problems generated using unstructured tetrahedral meshes with quadratic basis functions. The performance of the iterative methods is compared to a standard direct sparse matrix solver. Various problems with up to 70,000 degrees of freedom are considered during which the effect of a range of average element aspect ratios, including small ($\ll 1$) aspect ratios, on the performance of the PCG method is examined. A brief review is also made of stopping criteria for conjugate gradient solvers. One method based on the norm of the residual and an estimate of the smallest eigenvalue of the matrix system was implemented and tested with poor results.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Preconditioned Conjugate Gradient Methods

Three dimensional finite element stress analysis gives rise to large, sparse matrices. Though the cost of generating the matrix system for solution can be expensive, for larger analyses the overall cost is dominated by the cost of solving the resulting large sparse system. In many fields, such as computational fluid dynamics [43, 18, 32], petroleum reservoir simulation [13, 27] and semiconductor device simulation [15], iterative methods are typically used for the solution of three dimensional problems due to their less demanding requirements on a computer's CPU and memory resources. However, for elasticity problems, direct methods are still commonly used.

There is a growing use of finite element methods to analyse fully three dimensional elasticity problems. The cost of solution and storage requirements when using direct methods increases dramatically when moving from two dimensional to three dimensional problems. Consequently, there has been a recent upsurge of interest in applications of iterative methods for three dimensional elasticity problems

[39, 46, 29, 35, 2, 38, 44, 31, 11, 42, 16].

Iterative methods have been quite successful in some cases, especially in comparison to direct solvers. However, there are situations where the performance of iterative methods has been demonstrated to be quite poor [42]. In general this seems to occur for problems where the elements have a small ($\ll 1$) aspect ratio [35, 42], resulting in matrices which have a large condition number. Although it is advisable to produce meshes which result in well shaped elements in order to avoid ill-conditioned systems of equations, in practice, this may not always be possible. Also, many problems which were traditionally modelled using two dimensional plate and shell elements, can now be modelled using recently developed quasi-three-dimensional elements. This is particularly useful for analysing laminated composite materials [10, 42]. In this case, any practical finite element mesh will be composed of elements with poor aspect ratios.

The objective of this thesis is to investigate, develop and compare some existing iterative techniques for three dimensional elastic analysis, which can be applied to problems with unstructured, tetrahedral meshes. Previous work has shown that some iterative techniques will provide highly varying levels of performance depending on the particular problem the technique is applied to. Thus the goal of this thesis is not to find an "ultimate" technique but rather one that is robust and capable of reliably outperforming direct methods.

It is expected that for various types of problems, different techniques will show more promise. By using some knowledge of the problem to be solved, it should be possible to choose a better suited technique than just a generally robust method. Thus, any significant trends discovered in the performance of individual techniques are also highlighted in this work.

In particular, the focus of this research is on studying iterative solvers for the case of three dimensional elasticity problems on unstructured tetrahedral meshes. This is of special interest due to the extensive variety of geometries that can be accurately modelled by a tetrahedral mesh. It is also of interest because an unstructured tetrahedral mesh does not lend itself to easy application of some popular hierarchical or multi-grid [31, 16] approaches which combine elements to generate a coarser "level" mesh. Connell and Holmes [20] have recently developed an algorithm which can refine a tetrahedral mesh, but because of the apparent complexity of the algorithm it would not be easy to immediately implement. Consequently, in this work, attention will be restricted to PCG methods.

The convergence of a PCG method is strongly influenced by the type of preconditioning. Both level based [13, 27, 23, 24] and drop tolerance based [33, 40, 1, 24, 22] incomplete LU factorization (ILU) preconditioning will be examined. In addition, the ILU methods will be combined with a two level hierarchical basis [7, 9, 8, 30, 5, 6, 14, 45]. If storage limitations permit, the PCG method's solutions and CPU costs will also be compared with a direct method's solution and CPU cost.

The above methods have been tested on a variety of example problems. The selected problems range from simple model problems (a cube with varying aspect ratio) to fairly realistic complex three dimensional objects. These problems have from 1,000 to 70,000 degrees of freedom and all results were obtained on a Sun SPARCserver 670MP.

The tests were done using single precision storage for both the matrix A and the preconditioning matrix unless otherwise noted. All other terms for the conjugate gradient iterative technique were stored in double precision. This was done to help reduce memory requirements.

1.2 Stopping Criteria

The main focus of this thesis was an investigative search for a robust iterative solution technique. However, when using an iterative technique it is necessary to have some form of convergence check in order to decide when an acceptable solution has been found.

While studying the various iterative techniques, a drop of the initial residual by a factor of 10^{-6} was used as the convergence criteria. This proved to work well for the iterative tests. Examination of the solution vectors, compared to those found using a residual drop of 10^{-9}, showed that no further significant gain in the accuracy of the solution could be achieved without storing the matrices in double, rather than single, precision.

Included in this thesis is a brief study of stopping criteria for iterative methods. This review of stopping criteria includes some tests done using a modified technique presented by Kaasschieter in [36]. To properly run these tests it was necessary to use double precision storage for the matrix element terms. This is discussed in more detail in Chapter 6.

Chapter 2

The Problem

As has already been mentioned, elasticity problems on unstructured tetrahedral meshes are of particular interest. It is fairly obvious why it is difficult to combine elements in an unstructured tetrahedral mesh to achieve a coarser discretization for a multigrid approach. It is not as apparent as to why a mesh can't be refined instead.

It seems to be common knowledge that dividing a tetrahedra into several smaller tetrahedra often results in elements with poor aspect ratios. Doing the obvious thing and chopping four tetrahedra off the corners of the original results in an eight-triangular-faced polygon remaining in the center which must then also be modelled by tetrahedra. This can result in elements with a poor aspect ratios even if the initial element had a good aspect ratio. As mentioned earlier, Connell and Holmes [20] describe a method to avoid poor resultant aspect ratios but these techniques seem non-trivial to implement.

2.1 Formulation

The three dimensional linear elasticity problem for an isotropic material is given (in terms of the displacement vector $\mathbf{u}$) by

$$\mu\nabla^2\mathbf{u} + (\lambda + \mu)\nabla\nabla\cdot\mathbf{u} = 0 \tag{2.1}$$

where the boundary conditions are stipulated by specifying some values of $\mathbf{u}$, some surface tractions, or by allocating natural boundary conditions (freedom of value) to occur. In the above, $\mathbf{u}$ is the vector of displacements in the three co-ordinate directions, and

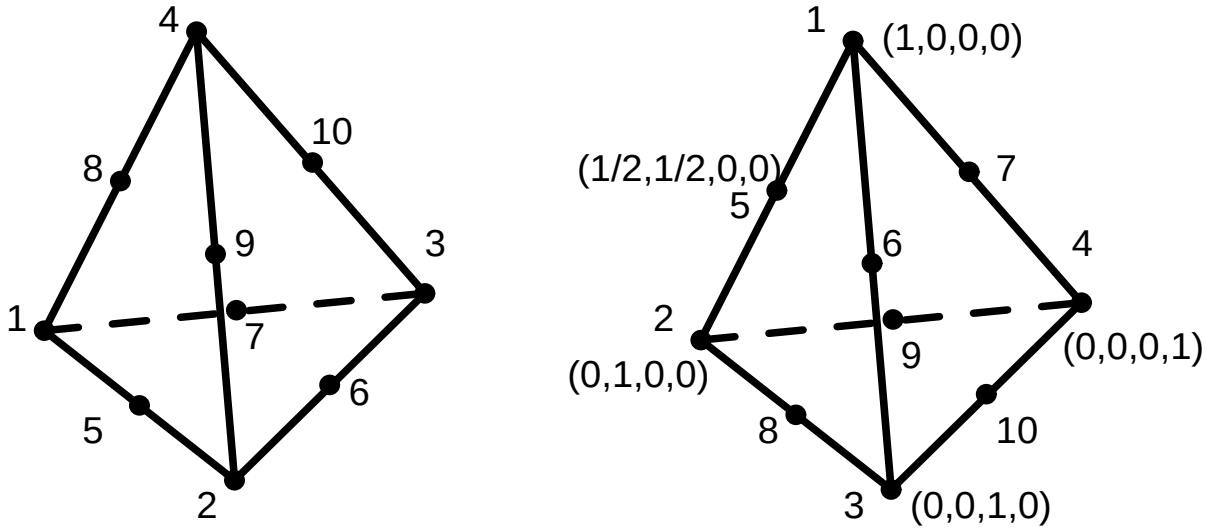
$$\begin{aligned} \mu &= \frac{E}{2(1+\nu)} \\ \lambda &= \frac{E\nu}{(1+\nu)(1-2\nu)} \end{aligned}$$

where E is Young's modulus, and ν is Poisson's ratio.

To develop a system of linear equations in the matrix form $A\mathbf{u} = \mathbf{b}$ to be solved, a standard finite element method using some defined basis can be applied to the partial differential equation (2.1). Basis functions are dependent on both the mesh elements being used and the desired polynomial degree for the basis being chosen.

2.2 Hierarchical Basis Functions

Tetrahedra are the logical mesh elements for general three dimensional bodies when compared to other common three dimensional elements including hexahedra and triangular prisms. This preference is due to the fact that the tetrahedra's geometry allows better mesh approximations of irregular objects.



Standard Nodal Numbering Nodal Numbering in Thesis

Figure 2.1: Nodal Numbering Conventions for Tetrahedra

For reasons of sign consistency when calculating the volume of a tetrahedral element (using determinants), or during coordinate transformations, most books and papers use a standard numbering convention for element nodes (Figure 2.1). This requires the proper assignment of local node numbers to an arbitrarily ordered set of four points describing a tetrahedra. To achieve this, extra calculations must be performed, or a pre-determined order for supplying the vertices of an element must be set. However, by simply using just the magnitude of any calculation dependent on these conditions, most sign problems can be avoided. During this research, this was a more practical solution as the test meshes came from several sources. Thus the nodal numbering convention was not adhered to in this research.

Figure 2.1 also demonstrates the tetrahedral barycentric coordinate convention ($\mathbf{L} = (L_1, L_2, L_3, L_4)$) for describing a point's position within an element. Barycentric coordinates (also known as the *natural coordinate system*) for tetrahedra are based on four coordinates which range from 0 to 1. The advantage of using this coordinate system is that common basis functions can be easily expressed in terms of these coordinates.

Barycentric coordinates are defined such that for each vertex i, $L_j = 1$ for $j = i$ and $L_j = 0$ for $j \neq i$ ($i, j \in \{1, 2, 3, 4\}$). Due to the linearity of the coordinate system it is also possible to show that $L_1 + L_2 + L_3 + L_4 = 1$, and that for any point P, the coordinates are $L_i = V_i/V$, $i \in \{1, 2, 3, 4\}$ (where V_i is the volume of the tetrahedra described by P and all vertex nodes except i and V is the volume of the whole tetrahedral element).

It is necessary to be able to determine the value for any unknown(s)/variable(s) $\mathbf{u}$ at position $\mathbf{x}_P$ (or in barycentric coordinates $\mathbf{L}_P$) in an element based on values calculated at the nodes ($\mathbf{u}_i$). Note that $\mathbf{u}$, and therefore $\mathbf{u}_i$, can be a scalar or a vector depending on the number of degrees of freedom in the system. The three

dimensional elasticity problem described in equation (2.1) has three degrees of freedom for each node representing the displacements in each coordinate direction. For any general basis with N nodes $\mathbf{u}(\mathbf{x}_P)$ is:

$$\mathbf{u}(\mathbf{x}_P) = \sum_{i=1}^{i=N} \mathbf{u}_i \phi_i(\mathbf{x}_P) \tag{2.2}$$

where $\phi_i(\mathbf{x}_P)$ are the basis functions at coordinate $\mathbf{x}_P$. For the standard (4 node) linear and (10 node) quadratic tetrahedral element cases using barycentric coordinates $\mathbf{u}(\mathbf{L}_P)$ is:

$$\begin{aligned} \mathbf{u}(\mathbf{L}_P) &= \sum_{i=1}^{i=4} \mathbf{u}_i \phi_i(\mathbf{L}_P) \\ &\quad \text{with: } \phi_i(\mathbf{L}_P) = L_i \end{aligned} \tag{2.3}$$

and

$$\begin{aligned} \mathbf{u}(\mathbf{L}_P) &= \sum_{i=1}^{i=10} \mathbf{u}_i \phi_i(\mathbf{L}_P) \\ &\quad \text{for } i \in \{1,2,3,4\} \quad \phi_i(\mathbf{L}_P) = L_i(2L_i - 1) \\ &\quad \text{and } i \in \{5,6,...,10\} \quad \phi_i(\mathbf{L}_P) = 4L_{v1}L_{v2} \end{aligned} \tag{2.4}$$

where $v1$ and $v2$ are each midpoint's neighbouring vertex nodes. The ϕ_i basis equations shown in equations (2.3 and 2.4) are the standard Lagrangian basis functions with polynomial degrees of 1 and 2 respectively. For these basis functions, ϕ_i equals 1 at node i and 0 at all other nodes.

As mentioned earlier, when a standard finite element method using these Lagrangian polynomial basis functions is applied to equation (2.1) the result is a system of linear equations

$$A\mathbf{u} = \mathbf{b} \tag{2.5}$$

where A is a symmetric positive definite matrix.

But there is no reason to be limited to using the standard Lagrangian quadratic basis functions (2.4). Instead, a hierarchical quadratic basis as in [7, 47, 9, 35, 5, 38, 31] could also be used.

The hierarchical basis ($\hat{\phi}_i$) used in this thesis was defined to be the standard linear basis for equations $i \in \{1,2,3,4\}$ and the standard quadratic basis for equations $i \in \{5,6,...,10\}$. In other words:

$$\begin{aligned} \hat{\phi}_i(\mathbf{L}_P) &= L_i \quad i \in \{1,2,3,4\} \\ \hat{\phi}_i(\mathbf{L}_P) &= 4L_{v1}L_{v2} \quad i \in \{5,6,...,10\} \end{aligned} \tag{2.6}$$

where $v1$ and $v2$ are each midpoint's neighbouring vertex nodes. Note that this means:

$$\begin{aligned} \hat{\phi}_i &= 1 \text{ at node } i \\ &= 0 \text{ at nodes } j \ , j \neq i \\ &\qquad i,j \in \{1,2,3,4\} \end{aligned} \tag{2.7}$$

and

$$\begin{aligned} \hat{\phi}_i &= 1 \text{ at node } i \\ &= 0 \text{ at nodes } j \ , j \neq i \\ &\qquad i \in \{5,6,...,10\} \ ; \ j \in \{1,2,...,10\} \end{aligned} \tag{2.8}$$

and that $\hat{\phi}_i, i \in \{1,2,3,4\}$ does not vanish at the neighbouring midside nodes as in the Lagrangian basis. In fact, $\hat{\phi}_i = \frac{1}{2}$ for $i \in \{1,2,3,4\}$ at the neighbouring midside nodes.

Thus for this hierarchical basis, the analogue of equation (2.4) is

$$\begin{aligned} \mathbf{u}(\mathbf{L_P}) &= \sum_{i=1}^{i=10} \hat{\mathbf{u}}_i \hat{\phi}_i(\mathbf{L}_P) \\ &\quad \mathbf{u}_i = \hat{\mathbf{u}}_i \quad i = 1,...,4 \\ &\quad \mathbf{u}_i = \hat{\mathbf{u}}_i + \frac{\hat{\mathbf{u}}_{v1} + \hat{\mathbf{u}}_{v2}}{2} \quad i = 5,...,10 \end{aligned} \tag{2.9}$$

where $v1, v2$ are the neighbouring vertex nodes to a midside node and $\hat{\mathbf{u}}_i$ are the "displacement" vectors calculated at each node in the element. Note that $\hat{\mathbf{u}}_i$ will not be equal the actual displacement vector $\mathbf{u}_i$ at the mid-edge nodes because of the basis functions being used.

The hierarchical structure found in the basis functions can also be preserved in an equivalent hierarchical version of the matrix equation (2.5):

$$A^H \hat{\mathbf{u}} = \begin{bmatrix} A_{mm} & A_{mv} \\ A_{vm} & A_{vv} \end{bmatrix} \begin{Bmatrix} \hat{\mathbf{u}}_m \\ \hat{\mathbf{u}}_v \end{Bmatrix} = \begin{Bmatrix} \hat{\mathbf{b}}_m \\ \hat{\mathbf{b}}_v \end{Bmatrix} . \tag{2.10}$$

To achieve this the midside unknowns are ordered first, and the vertex unknowns last. The block element A_{vv} in equation (2.10) is the usual stiffness matrix developed when using the linear basis functions in equation (2.3) to solve equation (2.1).

Since the usual quadratic basis can be written as linear combination of the hierarchical basis, it is possible to simply transform equation (2.5) into a new stiffness matrix [17]. To develop the transform matrix, let the matrix T be defined so that $\hat{\phi} = T\phi$ for one element. In other words, T is the transform from the old element basis ϕ to the new element basis $\hat{\phi}$. Then, proceeding in a fashion similar to [17] (where N_b is number of equations in the basis):

$$\begin{aligned} \text{since } \hat{\phi} &= T\phi \\ \text{then } \hat{\phi}_i &= \sum_j^{N_b} T_{i,j}\phi_j \\ \text{and since } \mathbf{u} &= \sum_i^{N_b} \mathbf{u}_i\phi_i \text{ or } \sum_i^{N_b} \hat{\mathbf{u}}_i\hat{\phi}_i \\ \text{then } \mathbf{u} = \sum_i^{N_b} \hat{\mathbf{u}}_i\hat{\phi}_i &= \sum_i^{N_b} \hat{\mathbf{u}}_i \left(\sum_j^{N_b} T_{i,j}\phi_j \right) \\ &= \sum_i^{N_b}\sum_j^{N_b} T^t_{j,i}\hat{\mathbf{u}}_i\phi_j \end{aligned}$$

$$
\begin{aligned}
& & & = & \sum_{l}^{N_b}\sum_{i}^{N_b} T^t_{i,l}\hat{\mathbf{u}}_l\phi_i \\
\text{and since } \sum_{i}^{N_b}\mathbf{u}_i\phi_i & = \mathbf{u} & & = & \sum_{i}^{N_b}\sum_{l}^{N_b} T^t_{i,l}\hat{\mathbf{u}}_l\phi_i \\
& \Rightarrow & \mathbf{u}_i & = & \sum_{l}^{N_b} T^t_{i,l}\hat{\mathbf{u}}_l \\
& \Rightarrow & \mathbf{u} & = & T^t\hat{\mathbf{u}} \quad . \qquad (2.11)
\end{aligned}
$$

This shows that when T is defined as the basis transform matrix such that $\hat{\phi} = T\phi$ then $\mathbf{u} = T^t\hat{\mathbf{u}}$. By inspection of the $\mathbf{u}$ and $\hat{\mathbf{u}}$ terms of equation (2.9) it is possible to see that on an element by element basis, for just one degree of freedom per node, the non-symmetric transform matrix (T') for the hierarchical basis and node numbering used in this work is:

$$
T' = \begin{bmatrix}
1 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
0 & 0 & 1 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\
0 & 0 & 0 & 1 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \quad . \qquad (2.12)
$$

For three degrees of freedom, the transformation matrix T' must be applied separately to each degree of freedom. In this thesis, the coordinate displacement terms for each node were ordered consecutively. Thus, to construct T, each 1 entry in the matrix T' would become a 3×3 identity matrix ($\mathrm{I}_{3\times3}$) and each $\frac{1}{2}$ would become

a $\frac{1}{2} \times \mathrm{I}_{3\times 3}$ matrix while the rest of the matrix entries would become 3×3 zero matrices.

The right-hand side terms $\mathbf{b}_j$ are the result of inner product $(\phi_j, \mathbf{f})$ defined as the surface integral $\int \mathbf{f}\phi_j \mathrm{d}s$, where $\mathbf{f}$ is the specified surface traction vector applied to the element. Intuitively, the surface integral determines the component of the surface traction vector $\mathbf{f}$ is applied at each node. To find $\hat{\mathbf{b}}$ in terms of $\mathbf{b}$ remember that $\mathbf{b}_j = (\phi_j, \mathbf{f})$ and also:

$$\begin{aligned}
\hat{\mathbf{b}}_i &= \left(\hat{\phi}_i, \mathbf{f}\right) \\
&= \left(\sum_j^{N_b} T_{i,j}\phi_j, \mathbf{f}\right) \\
&= \sum_j^{N_b} T_{i,j}\left(\phi_j, \mathbf{f}\right) \\
\text{giving } \hat{\mathbf{b}}_i &= \sum_j^{N_b} T_{i,j}\mathbf{b}_j \\
\Rightarrow \hat{\mathbf{b}} &= T\mathbf{b} \ . \qquad (2.13)
\end{aligned}$$

Using $\mathbf{u} = T^t\hat{\mathbf{u}}$ and $\hat{\mathbf{b}} = T\mathbf{b}$, A^H can be found in terms of A:

$$\begin{aligned}
A\mathbf{u} &= \mathbf{b} \\
AT^t\hat{\mathbf{u}} &= \mathbf{b} \\
TAT^t\hat{\mathbf{u}} &= T\mathbf{b} \\
TAT^t\hat{\mathbf{u}} &= \hat{\mathbf{b}} \\
\text{but } A^H\hat{\mathbf{u}} &= \hat{\mathbf{b}} \\
\text{therefore } A^H &= TAT^t \ . \qquad (2.14)
\end{aligned}$$

This has shown that:

- $A^H = TAT^t$

- $\mathbf{u} = T^t \hat{\mathbf{u}}$
- $\hat{\mathbf{b}} = T\mathbf{b}$.

is true for a tetrahedral element analysed using the basis functions defined in equations (2.4 and 2.6).

To extend T to a global transformation matrix it is also necessary to define global versions of A, $\mathbf{u}$ and $\mathbf{b}$. For the three-dimensional elasticity problem there are three degrees of freedom (p or q) for each node (i, j, l or m). For example, considering each axis of displacement separately, $\mathbf{u}_{i(p)}$ is the global displacement for node i in direction p, and similarly $\mathbf{b}_{j(q)}$ is the global right-hand side entry for node j in direction q. Logically $A_{j(q),i(p)}$ is the A matrix term in row j(direction q) and column i(direction p). This means the expression:

$$\sum_{i}^{N_n} \sum_{p=1}^{3} \left(A_{j(q),i(p)} \mathbf{u}_{i(p)} \right) = \mathbf{b}_{j(q)} \tag{2.15}$$

(where N_n is the number of nodes) is the dot product of row $j(q)$ of A with the vector $\mathbf{u}$ giving the right hand side term $\mathbf{b}_{j(q)}$.

The matrix T also depends on both the node (i, j, l or m) and on the displacement direction (p or q). So the global equivalents of equations (2.11 and 2.13) are:

$$\mathbf{u}_{i(p)} = \sum_{l}^{N_n} T^t_{i(p),l(p)} \hat{\mathbf{u}}_{l(p)} \tag{2.16}$$

$$\hat{\mathbf{b}}_{m(q)} = \sum_{j}^{N_n} T_{m(q),j(q)} \mathbf{b}_{j(q)} \quad . \tag{2.17}$$

Substituting these equations into equation (2.15) leads to the equivalent of the algebra applied to get equation (2.14):

$$\sum_{i}^{N_n} \sum_{p=1}^{3} \left(A_{j(q),i(p)} \sum_{l}^{N_n} T^t_{i(p),l(p)} \hat{\mathbf{u}}_{l(p)} \right) = \mathbf{b}_{j(q)}$$

and multiplying by $T_{m(q),j(q)}$:

$$\begin{aligned}
\sum_{j}^{N_n} T_{m(q),j(q)} \left[\sum_{i}^{N_n} \sum_{p=1}^{3} \left(A_{j(q),i(p)} \sum_{l}^{N_n} T^t_{i(p),l(p)} \hat{\mathbf{u}}_{l(p)} \right) \right] &= \sum_{j}^{N_n} T_{m(q),j(q)} \left[\mathbf{b}_{j(q)} \right] \\
\sum_{j,i,l}^{N_n} \sum_{p=1}^{3} \left(T_{m(q),j(q)} A_{j(q),i(p)} T^t_{i(p),l(p)} \hat{\mathbf{u}}_{l(p)} \right) &= \hat{\mathbf{b}}_{m(q)} \\
\text{or} \quad \sum_{j,i,l}^{N_n} \sum_{p=1}^{3} \left(T_{m(q),j(q)} A_{j(q),i(p)} T^t_{i(p),l(p)} \right) \left(\hat{\mathbf{u}}_{l(p)} \right) &= \hat{\mathbf{b}}_{m(q)} \\
\text{simplifying to} \quad \sum_{l}^{N_n} \sum_{p=1}^{3} A^H_{m(q),l(p)} \hat{\mathbf{u}}_{l(p)} &= \hat{\mathbf{b}}_{m(q)} \qquad (2.18)
\end{aligned}$$

Equation (2.18) would be the complete matrix multiplication if $m(q)$ was also indexed through. Thus, this implies that $A^H = TAT^t$ is true for the global system also. To get the exact form of equation (2.10) some node reordering would also be required to attain an ordering where the midside nodes come first and then the vertex nodes.

For the purposes of this thesis, it was easier to construct the stiffness matrix using the hierarchical basis from the beginning rather than perform the translation later.

Various other new basis functions could be employed [17, 11, 8, 14], and depending on the type of preconditioning used they might prove to be superior to a hierarchical basis if many hierarchical levels are used with a very simple preconditioning method [14]. However, by employing a suitable preconditioning method, it is possible to obtain good results with an hierarchical basis in three dimensions, even when using a large number of levels in the hierarchical basis functions [8, 6].

This work restricts its attention to a two level p-version of the of the hierarchical basis [38]. As has been demonstrated, this can be easily developed for use with a quadratic basis on an unstructured tetrahedral mesh even with possible discontinuous material properties.

Even higher numbers of p-levels could be used in a basis as is considered by Foresti, Hassanzadeh, Murakami and Sonnad in [31], but this line of research is not pursued here. Rather, the focus is on the developing the idea of implementing a simple two-level method similar to the technique used in [29], in the context of a multi-grid-like method.

Chapter 3

Preconditioning Methods

The basic purpose of preconditioning an iterative method is to improve the rate of convergence towards solution. For good convergence rates when using conjugate gradient methods, it is best to have a condition number for the system to be close to 1. Thus the "ideal" preconditioning matrix C for the system $A\mathbf{u} = \mathbf{b}$ would be $C = A^{-1}$. However, this is much too expensive to calculate and store and essentially constitutes a direct solve of the problem.

Choosing the best preconditioning matrix C is a complicated issue. It is desirable that:

- C to be as close to A^{-1} as possible,
- it is possible to calculate C quickly,
- C requires only limited storage space,
- multiplying by C is not too expensive for each iterative step.

Because the first desired property usually conflicts with the others, the decision becomes an optimization question which is highly dependent on the problem. The final goal is to minimize the memory and time required to find the solution.

One of the most robust and widely used methods is incomplete LU (ILU) preconditioning. However, there are many different issues which arise when using ILU preconditioning.

Perhaps the most fundamental issue concerns the method used to determine the final form of the preconditioning matrix. The basic idea is to limit the amount of fill that occurs during an LU factorization. Usually, an ILU will be based on the graph of the matrix (levels of fill) [27, 23], a drop tolerance [33, 40], or both [24].

Another issue which has been demonstrated to have a strong effect on the convergence rate is the ordering of the unknowns [12, 23]. Many ordering techniques cannot be generalized to unstructured graphs. The two common ordering techniques used in this thesis were the Reverse Cuthill-McKee (RCM) and Minimum Degree ordering. RCM ordering was used because it is cheap to compute and has been shown to be a reasonable ordering for ILU techniques [23]. Minimum degree ordering was used for similar reasons except that it is better for direct solves. Better orderings are known but, again, they cannot be generalized to apply to unstructured graphs or are too expensive to compute for this problem [24]. The effect of these and many other ordering techniques on conjugate gradient methods are also discussed in more detail by Duff and Meurant in [26].

Attention must also be given to the fact that FE analysis of elasticity problems typically do not result in M-matrices. Consequently, an incomplete LU factorization (ILU) of a symmetric positive definite matrix which is not an M-matrix may produce negative pivots [37]. For the conjugate gradient method to work, these negative

pivots must be replaced with positive values.

Finally the implications of using a hierarchical basis must be considered. In contrast to applying one ILU method over an entire matrix, it is possible to apply different methods to each "block" of a matrix generated using hierarchical basis functions.

3.1 Notation

A few notation conventions should be set before discussing preconditioning methods specifics.

The notation $ILU(level, \epsilon)$ will be used to denote an incomplete factorization, with level $level$, and drop tolerance ϵ (method specifics in see Section 3.2). For example

$$ILU(\infty, 0.01)$$

would refer to a pure drop tolerance ILU (drop tolerance = 0.01), while

$$ILU(3, 0.0)$$

would refer to a pure level based ($level = 3$) ILU. This convention is also extended to describe the ILU methods used with the hierarchical basis.

$$ILU_m(\infty, 0.01), \quad ILU_v(3, 0.0)$$

refers to a drop tolerance of $\epsilon = 0.01$ being used to precondition the matrix block A_{mm} (midpoint/quadratic region of A) and a $level = 3$ based ILU to precondition the matrix block A_{vv} (vertex/linear region of A).

During factorization in a given pivot sequence, let $A^{(k)}$ be the submatrix in the incomplete factorization which remains after eliminating the first $k-1$ variables (columns $1,..,k-1$ have been eliminated). This implies the following equivalences of notation:

$$\begin{aligned} A^{(1)} &= A \\ \{A^{(k)}\}_{ij} &= a_{ij}^{(k)} \quad . \end{aligned}$$

It should also be mentioned that when applying standard FE methods to the three dimensional elasticity problem, the resulting system of equations is symmetric. Thus, both memory and CPU time can be saved by calculating and storing only the upper half of the matrix. This was done for all tests made for this thesis.

3.2 Method Specifics

For a given ordering of the unknowns, there is a standard definition of a level based ILU [23] based on edge generation during node elimination in a graph. The same is not true for a drop tolerance based ILU. There are several ways to define a dropping criteria for possible fill terms [33, 40, 24]. In this work, if a drop tolerance ILU is selected, and when

$$\begin{aligned} |a_{ij}^{(k)}| &< \epsilon \, a_{ii}^{(k)} \\ &\text{where: } i \geq k \;\; ; \;\; j > i \end{aligned} \tag{3.1}$$

is true then the element $a_{ij}^{(k)}$ is dropped from the ILU, otherwise it is kept. Remember that since A is symmetric, only the upper triangular factor need be considered.

Experiments were made with various criteria, and, in accordance with previous works [33, 1], have shown that the quality of the ILU seems to be fairly insensitive

to the precise form of the drop tolerance. However, it is of course sensitive to the drop parameter ϵ. The criteria (3.1) happens to be easy to implement efficiently in the test code. The precise choice of the best drop tolerance criterion remains an open question. More discussion of this may be found in [33, 40, 1, 24].

As mentioned previously, an incomplete factorization of a symmetric positive definite matrix which is not an M-matrix, may result in a negative diagonal, i.e. $a_{kk}^{(k)} < 0$. This problem is well known, but has not been considered as being too serious in the past. In many codes, if a negative pivot is encountered, the pivot is simply replaced by a suitably chosen positive number, and the factorization proceeds [40, 37]. From experience, this approach works well in many fluid dynamics applications, but is very poor for stress analysis problems, possibly because stiffness matrices may have off-diagonals which are several orders of magnitude larger (in absolute value) than the diagonals.

Another possibility is to add suitable multiples of the absolute value of any dropped fill terms (for either a drop tolerance or level based ILU) to the diagonal of the ILU. It was shown by Jennings and Malik that the resulting ILU matrix is positive definite [33]. However, the addition of the dropped terms to the diagonal may produce a poor ILU, and can result in slow convergence. It is clear that this approach overestimates the amount that must be added to the diagonal in order to preserve positive definiteness. To see this, consider an ILU of a symmetric positive definite, diagonally dominant M-matrix. It is well known that a ILU of such a matrix will be positive definite, without addition of any terms to the diagonal. However, the approach described above [33, 40, 1] would unnecessarily increase the size of the diagonal of the ILU, and hence slow convergence. Nevertheless, this method is completely reliable, in that a successful ILU factorization is always guaranteed. In the following, we will refer to this approach as *J&M Add*.

Assuming that the original diagonals of the stiffness matrix have been scaled to one, then the following implementation of the method [1] is used in this work. If $a_{ij}^{(k)}$ is a term which is dropped from the ILU (for either level or drop tolerance), then,

$$\begin{aligned} a_{ii}^{(k)} &:= a_{ii}^{(k)} + |a_{ij}^{(k)}| \\ a_{jj}^{(k)} &:= a_{jj}^{(k)} + |a_{ij}^{(k)}| \\ & \quad i \geq k \ ; \ j > i \ . \end{aligned} \tag{3.2}$$

Again remember that in equation (3.2) it is assumed that only the upper triangular part of A is being processed.

The negative pivot problem can also be avoided in the following way. The diagonal $a_{kk}^{(k)}$ is monitored. If

$$\begin{aligned} a_{kk}^{(k)} &< \gamma \ R_k \\ & R_k = \max_{m \geq k} \left(|a_{km}^{(k)}| \right) \ . \end{aligned} \tag{3.3}$$

Then the ILU factorization is aborted, and a new ILU is attempted on the perturbed system [39]

$$\forall i \ ; \ \left(a_{ii}^{(1)} \right)' = (1+\alpha) a_{ii}^{(1)} \ . \tag{3.4}$$

If equation (3.3) is violated again, then α is increased, and the ILU is repeated, until equation (3.3) is satisfied for all k. Typically, the following parameters were used:

$$\begin{aligned} \gamma &= 0.0 \\ \alpha &= (p-1) \ \eta \\ \eta &= 10^{-3} \end{aligned} \tag{3.5}$$

where p is the number of the current attempt to ILU factor the matrix. In other words, if $a_{kk}^{(k)} < 0.0$ then the ILU factor is restarted after each diagonal matrix entry is multiplied by $(1 + 0.001)$, $(1 + 0.002)$, and so on.

It is possible to simply precondition the stiffness matrix A resulting from the standard Lagrange quadratic basis, using an ILU factorization. In other words, incompletely factor A in equation (2.5) as

$$\mathbf{P_0} \;=\; LL^t \;\simeq A \;. \tag{3.6}$$

Henceforth, this preconditioning will be referred to as $\mathbf{P_0}$.

If the hierarchical basis is used, then we will consider two approximate block factorizations of the hierarchical stiffness matrix (2.10). The simplest preconditioning [7, 35], which will referred to as $\mathbf{P_1}$, is

$$\mathbf{P_1} \;=\; \begin{pmatrix} L_m L_m^t & 0 \\ 0 & L_v L_v^t \end{pmatrix} \simeq \begin{pmatrix} A_{mm} & A_{mv} \\ A_{vm} & A_{vv} \end{pmatrix} \tag{3.7}$$

where L_m, L_v are (possibly) approximate factors of the block diagonals

$$\begin{aligned} L_m L_m^t &\simeq A_{mm} \\ L_v L_v^t &\simeq A_{vv} \end{aligned} \tag{3.8}$$

A more accurate preconditioning, which will be referred to as $\mathbf{P_2}$ attempts to take into account some of the vertex-midside coupling terms A_{mv}, A_{vm} [7, 5].

$$\mathbf{P_2} \;=\; \begin{pmatrix} L_m L_m^t & 0 \\ A_{vm} & L_v L_v^t \end{pmatrix} \begin{pmatrix} I & (L_m L_m^t)^{-1} A_{mv} \\ 0 & I \end{pmatrix} \simeq \begin{pmatrix} A_{mm} & A_{mv} \\ A_{vm} & A_{vv} \end{pmatrix} \tag{3.9}$$

If h is the mesh size parameter, then the spectral condition number of A_{vv} is $O(h^{-2})$, since A_{vv} is the usual stiffness matrix for a linear basis function discretization of equation (2.1). It can be shown that the condition number of A_{mm}

is independent of h [7, 35]. An intuitive explanation for this is simply that A_{mm} represents the stiffness matrix of a discretization of equation (2.1) with all vertex (v) nodes fixed [29].

Since A_{mm} has condition number independent of h, then, at least in theory, a very simple preconditioning of A_{mm} can be used. Various authors have proposed a $\mathbf{P_1}$ preconditioning of the form [7, 35]

$$\begin{aligned} L_m L_m^t &= diag\,(A_{mm}) \\ L_v L_v^t &= ILU(\infty, 0.0) \end{aligned} \tag{3.10}$$

(i.e. an exact factorization of A_{vv}, and diagonal scaling of A_{mm}). Consider a sequence of discretizations of equation (2.1) with different mesh sizes h, then if x_h is a vector in the finite element solution space V_h, for a given mesh size h, then it can be shown for $\mathbf{P_1}$ (equation (3.10)) that

$$c_1(x_h, \mathbf{P_1}_h x_h) \leq (x_h, A_h^H x_h) \leq c_2(x_h, \mathbf{P_1}_h x_h)\ ;\quad \forall x_h \in V_h\ \ ,\ \ \forall h \tag{3.11}$$

where c_1, c_2 are independent of h [35]. In other words, $\mathbf{P_1}$ and A^H are spectrally equivalent, which implies that the number of iterations required to solve the stiffness matrix using a PCG method is independent of mesh size. It is also known that if $L_v L_v^t$ is an exact factorization of A_{vv} and $L_m L_m^t$ is any approximate factorization which is spectrally equivalent to A_{mm}, then $\mathbf{P_2}$ and A^H are also spectrally equivalent [5].

Of course, this assumes that $L_v L_v^t$ is an exact factorization of A_{vv}. This matrix is about 1/8 the size of A for typical three dimensional triangulations, and is much more sparse than A (linear basis functions as opposed to quadratic). Hence the cost of exactly factoring A_{vv} is small compared to the cost of exactly factoring A. However, it is possible to use more hierarchical levels, and to use a recursive form

of the $\mathbf{P_2}$ preconditioner, to develop a preconditioner which is spectrally equivalent to A, but does not require an exact factorization, except on the coarsest level [8, 6]. Note that in this case the preconditioner [8, 6] is no longer simply based on diagonal scaling of high levels, but uses an accurate approximation to the Schur complement at each level.

However, use of more than two levels requires the definition of "coarse grid" linear basis functions, which is a non-trivial task for unstructured tetrahedral meshes, or the use of higher order elements. Consequently, only two levels will be considered in this thesis. As mentioned above, the computational work for an exact factorization of A_{vv} is small compared to a direct solve of the original problem, and this is a small price to pay for a robust iterative method.

Unfortunately, the situation is not as rosy as it might appear. Although the constants c_1, c_2 are independent of h for suitable L_m, L_v, these constants are functions of Poisson's ratio ν and the element aspect ratio [35]. In fact, if the $\mathbf{P_1}$ preconditioner with L_m defined as in equation (3.10) is used, the condition number of the preconditioned system (c_2/c_1) may be very large for bad ($\ll 1$) element aspect ratios [35]. Consequently, it will often be necessary to use a more accurate ILU factorization of A_{mm}, even though A_{mm} has condition number independent of h. Note that Farhat and Sobh advocate the use of an exact factorization of A_{mm} in [29].

As mentioned previously, the ordering of the unknowns can have a significant effect on the convergence of PCG methods [26, 23]. In this work, a Reverse Cuthill McKee (RCM) ordering will be used [25] for an ILU factored submatrix. The RCM ordering is based on the nodal graph of the finite element mesh. This nodal ordering is then converted to a degree of freedom ordering by ordering all the displacements at each node consecutively. In other words, this is a block RCM ordering, with

all displacements at a single node contained in each block. Recent studies have indicated that it is desirable to have all unknowns of a node ordered consecutively. If an exact factorization of a submatrix is used, then minimum degree ordering [25] is used to minimize work and fill.

Note that these preconditionings all require some operation like

$$A_{pp}^{-1}x \tag{3.12}$$

which is approximated by

$$(L_pL_p^t)^{-1}x \ , \quad p \in \{m, v\} \tag{3.13}$$

where x is some vector. Instead of simply approximating the inverse in equation (3.13) by the ILU $L_pL_p^t$, it is possible to determine the solution to equation (3.12) to any degree of precision by using a PCG method, with $L_pL_p^t$ as a preconditioner to solve equation (3.12). This gives rise to nested levels of a PCG iteration. Each outer PCG iteration, requires several inner PCG iterations to solve equation (3.12).

Chapter 4

Test Problems

The test matrices were formed using a standard Gelerkin FE method approach. Dirichlet boundary conditions (where displacements are specified) were applied by adjusting the values of both the matrix entry a_{ii} and the right-hand vector component $\mathbf{b}_i$. When a_{ii} is very large compared to the rest of the terms in row i then $\mathbf{u}_i \simeq \mathbf{b}_i / a_{ii}$. Thus to satisfy the Dirichlet boundary conditions a_{ii} was multiplied by a large number (typically 10^9) and $\mathbf{b}_i$ had the specified displacement times the large number added to it.

As previously mentioned, the convergence tolerance used for all the tests was

$$\frac{\|r^k\|_2}{\|r^0\|_2} \quad < \quad 10^{-6} \tag{4.1}$$

where $\|r^0\|_2$ is the initial l_2 norm of the residual, and $\|r^k\|_2$ is the norm of the residual after k iterations. After symmetrically scaling the matrix (and consequently the solution vector and right-hand side), a slightly modified zero vector was used as the initial guess. The modified zero vector was adjusted to contain any specified displacements (Dirichlet boundary conditions).

Note that in some cases we will be comparing results for an ILU of the original stiffness matrix A and the hierarchical basis A^H, in which case the residual, and hence the convergence criteria, are not precisely the same. However, we have computed the difference between the usual residual and the hierarchical basis residual (at convergence) in all our runs. The maximum difference in the norms of these two residuals (at convergence) was about 20%, indicating that if a nodal residual was used in the convergence test for the hierarchical basis, or *vice versa*, the number of iterations would change by one or two, which would not have any appreciable impact on our results.

Where storage limitations permit, we have also solved the test problems using the Yale Sparse Matrix Package (YSMP), which is a direct solver [28]. Minimum degree ordering was used for the direct solve.

A summary of test problem data for each problem is given in Table 4.1. For general tetrahedral meshes, it is useful to have a well defined measure of element aspect ratio. A common measure [34] is three times the ratio of the radius of the circumsphere to the radius of the inscribed sphere for each tetrahedra. Table 4.1 shows the element aspect ratio for each test problem. The test problems are described in detail below.

4.1 Test Cases

4.1.1 Cube Problem

This problem consists of a cube-like solid, with physical dimensions $l \times l \times lz$. This solid is first gridded using an $n \times n \times n$ rectangular grid, which results in $(n-1) \times (n-1) \times (n-1)$ bricks. Each brick is then further subdivided into

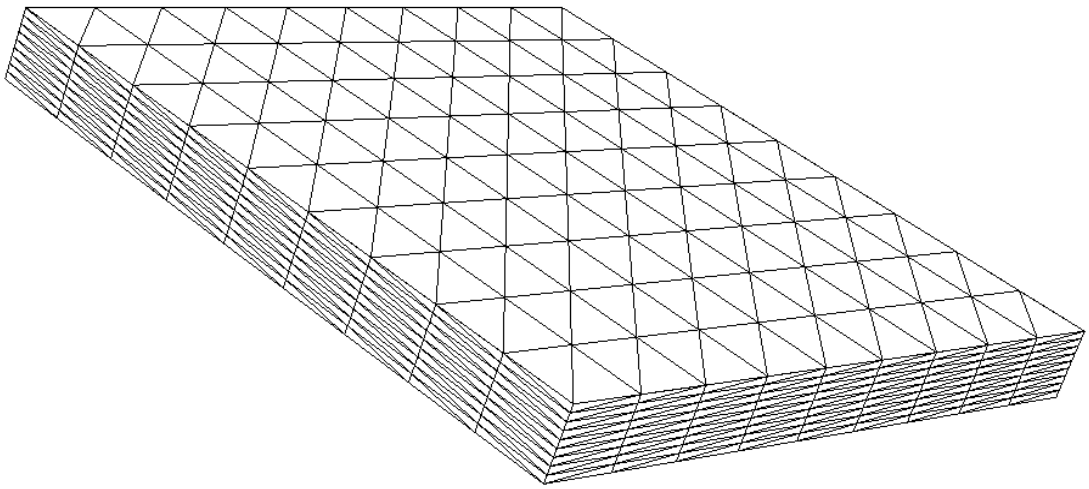


Figure 4.1: Surface Mesh of Cube with $l/lz = 10$ Test Problem

Table 4.1: Summary of Test Problem Data

Problem Descriptions	Degrees of Freedom	Nonzeros in U(A)	Element Aspect Ratios Minimum	Average	Material Properties ν
$4 \times 4 \times 4$ $(l/lz = 1)$	1029	34377	0.717	0.717	0.400
Cube $(l/lz = 10)$			0.190	0.190	
$(l/lz = 100)$			0.020	0.020	
$10 \times 10 \times 10$ $(l/lz = 1)$	20577	816081	0.717	0.717	0.400
Cube $(l/lz = 10)$			0.190	0.190	
$(l/lz = 100)$			0.020	0.020	
Balljoint	21411	847719	1.6×10^{-4}	0.779	0.266
Block Containing Prism	19479	757050	1.2×10^{-4}	0.784	Block 0.300 Prism 0.450
Bushing	49896	2001024	0.451	0.595	0.346
Multi-material Bracket	50637	1998375	1.0×10^{-4}	0.783	Supports 0.450 Base 0.200
Corroded Pipe	73191	2646231	0.184	0.356	0.301

six tetrahedra. A depiction of the surface mesh of a *Cube* with an aspect ratio of $l/lz = 10$ is shown in Figure 4.1.

The solid is constrained at the four bottom corners, and has a specified deflection at one top corner. By varying the thickness lz, the effect of varying element aspect ratio can be observed. This problem was solved using a $4 \times 4 \times 4$ and $10 \times 10 \times 10$ mesh.

4.1.2 Balljoint Problem

The balljoint problem is a quarter model of a balljoint. The ball has a radius of 0.01 m and the shaft extending from it has a radius of 0.008 m and a length of 0.016 m from the center of the ball. The back of the ball has been cut off perpendicular to the axis of the shaft at a distance of 0.006 m from the center of the ball. A depiction of the surface mesh is shown in Figure 4.2. The tetrahedral mesh was

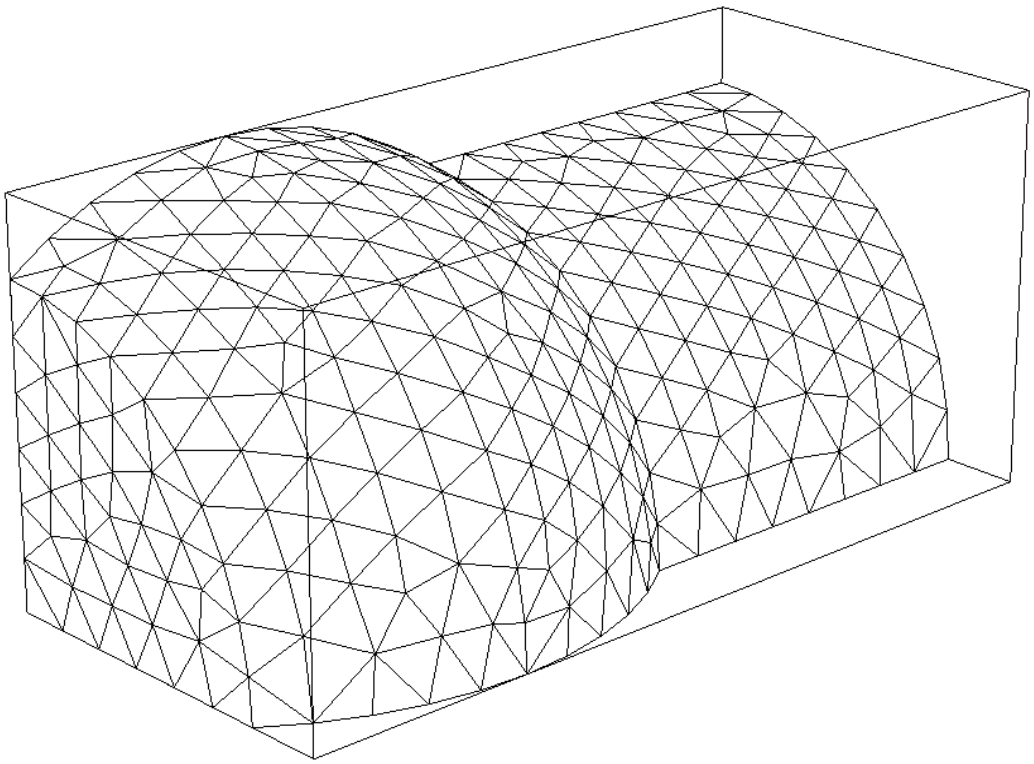


Figure 4.2: Surface Mesh of Balljoint Test Problem

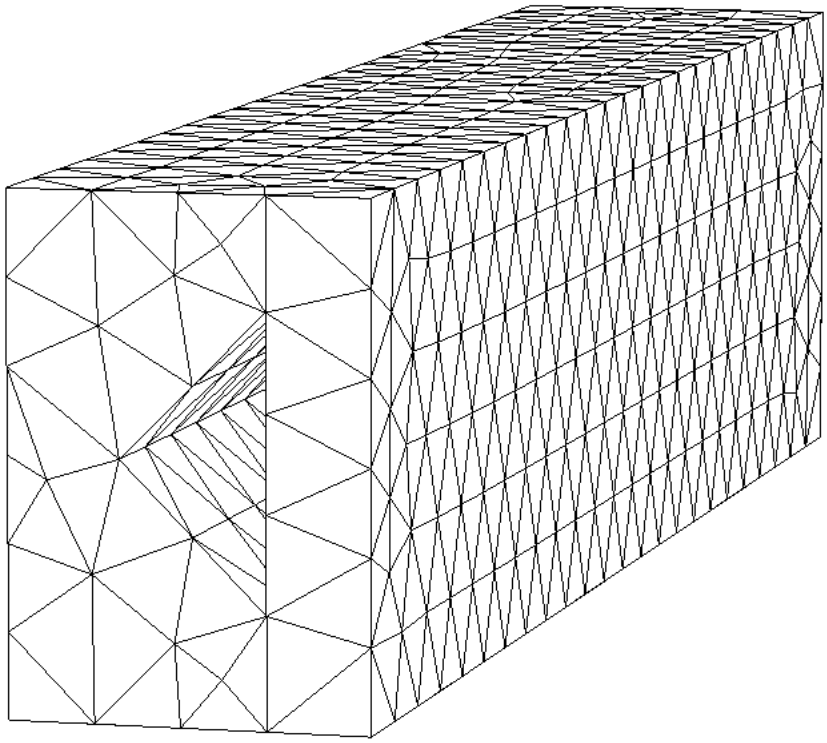


Figure 4.3: Surface Mesh of the Outer Block with Hole for Prism

generated by the I-DEAS VI: Mesh Generation Package [21].

For boundary conditions, the end of the shaft has been deflected along and across the direction of the shaft axis. The ball surface was fixed in a slanted band near where the shaft connects to the ball.

4.1.3 Block Containing Prism Problem

This problem consists of a block made of one material with a prism of another

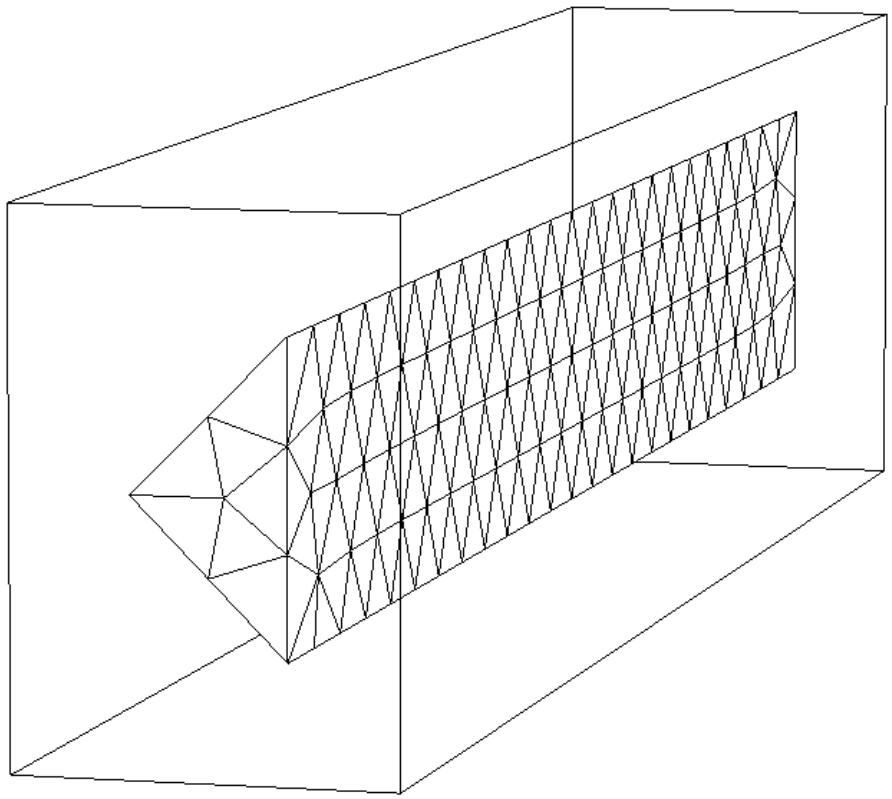


Figure 4.4: Surface Mesh of the Prism

material through the length of the block. The block has a width of 0.07 m, a height of 0.1 m and a length of 0.4 m. The cross section of the triangular prism is defined by the three points (0.02, 0.05), (0.05, 0.02) and (0.05, 0.08). The tetrahedral meshes were generated by the I-DEAS VI: Mesh Generation Package [21]. Refer to Figures 4.3 & 4.4 to see the surface mesh of the two separated components.

Boundary conditions are created by first fixing the position of all the nodes on one of the exposed ends of the prism and then applying a uniform, skewed force over the entire surface area of the other end of the block.

4.1.4 Bushing Problem

The bushing problem is a model of thick washer made from a soft material. The bushing has an outer radius of 0.04 m, an inner radius of 0.01 m, and a thickness of 0.01 m. A view of the surface mesh for the bushing can be found in Figure 4.5.

The boundary conditions are designed to simulate the use of the bushing to protect a block of material from direct contact with a C-clamp being applied right on the edge of the block. Thus the bushing is being bent over the lip of the block by a skew load. The position of all nodes on the bottom surface of the bushing to the right of 0.03 m are fixed and every surface triangle on the top of the bushing is subjected to a skew load.

4.1.5 Multi-material Bracket Problem

The bracket problem is another multi-material test problem. The base and face plates are made of the same material and the supports are made of a different material. The base and face plates are 0.03 m deep/thick, the two triangular plate

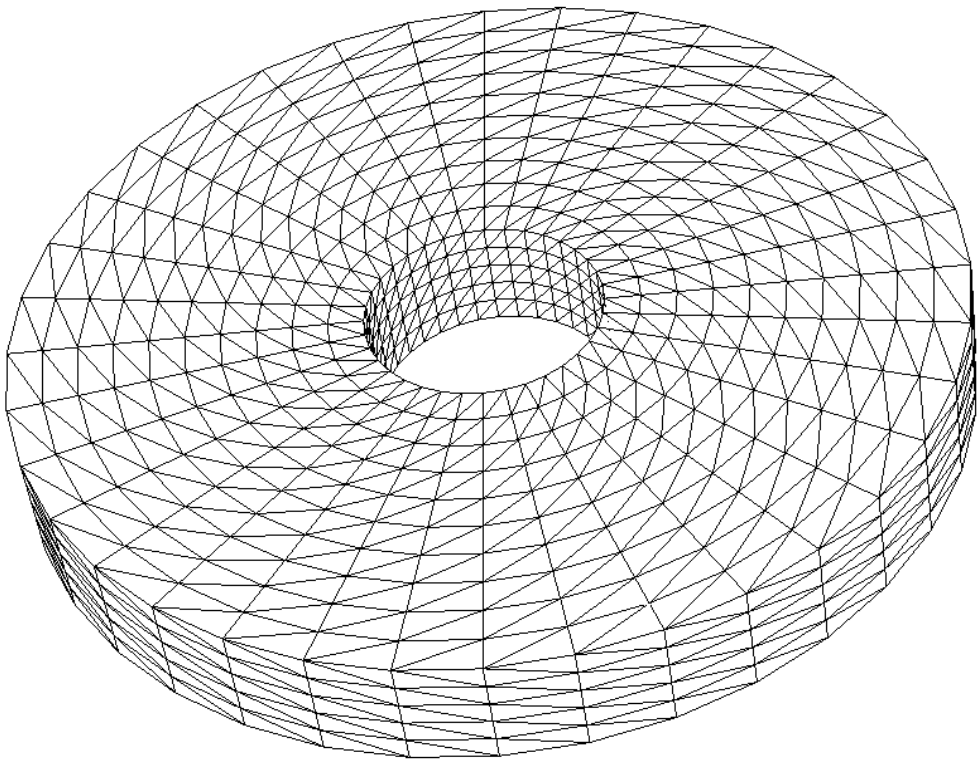


Figure 4.5: Surface Mesh of the Bushing

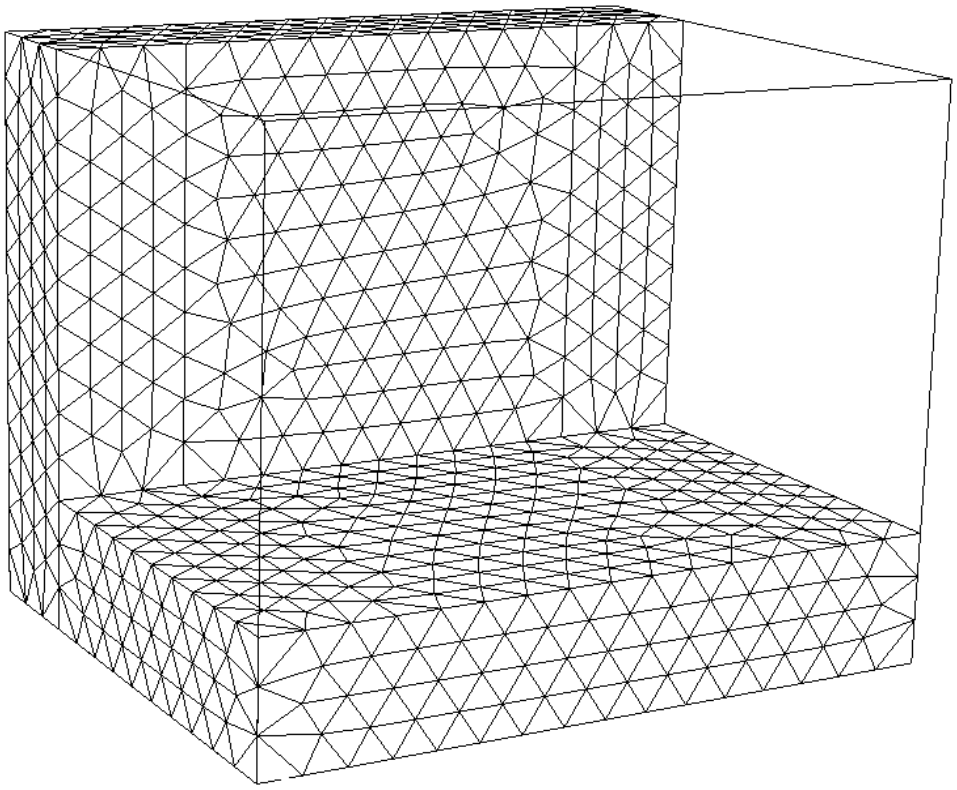


Figure 4.6: Surface Mesh of the Bracket Face and Base Plates

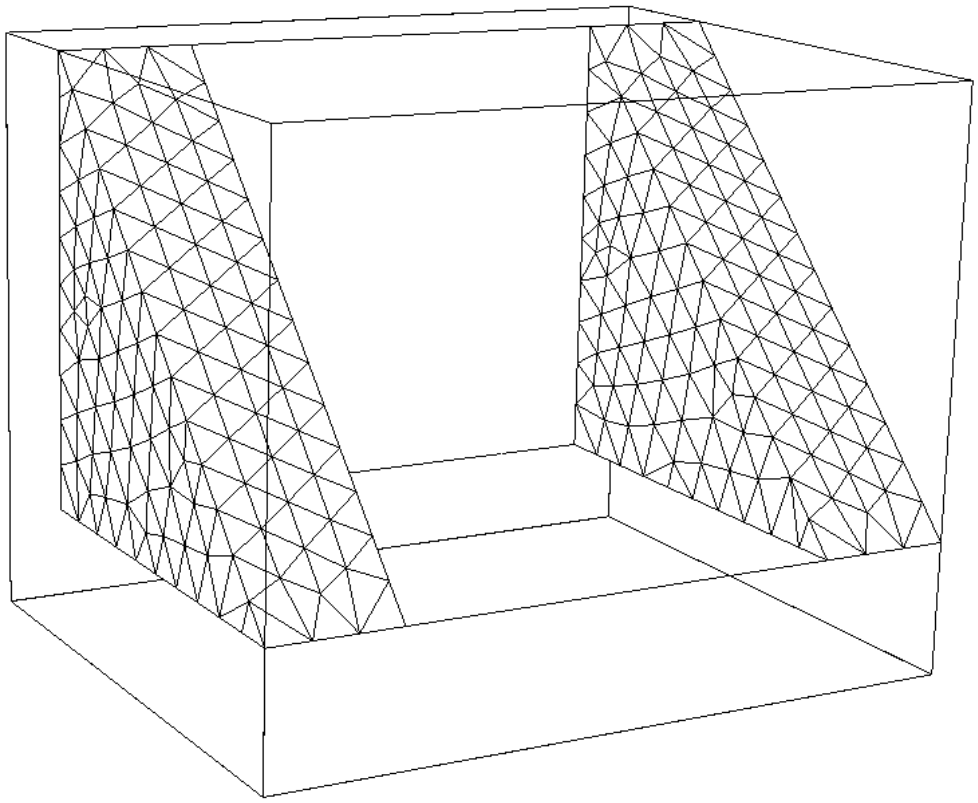


Figure 4.7: Surface Mesh of the Bracket Supports

supports are also 0.03 m thick, and the supports are 0.1 m apart. In total the bracket/brace is 0.13 m deep, 0.13 m high and 0.16 m wide. Figure 4.6 depicts the surface mesh for the base and face plates combined and Figure 4.7 depicts the surface mesh for the supports. The tetrahedral mesh was generated by the I-DEAS VI: Mesh Generation Package [21].

For boundary conditions the base of the plate is fixed, and the face plate nodes are displaced perpendicular to the plate's surface as a function of displacement from the base plate (i.e. $0.001 \times z$ coordinate). This effectively slants the face plate as if heavily loaded.

4.1.6 Corroded Pipe Problem

The mesh for the pipe problem was contributed by B. Chouchaoui [19] who is studying the impact of corrosion on pipe strengths. Only the mesh representing half the arc of the pipe was used. The boundary conditions were a simplified version of the original experimental data. The pipe segment is 350.0 mm long with an inner and outer radius of 155.46 and 161.86 mm respectively. The corrosion pit is centered on the pipe segment and is marked by a finer grid. For our analysis, each hexahedral element was divided into six tetrahedral elements. Figure 4.8 shows the surface mesh from the hexahedral discretization.

For boundary conditions, the pipe experiences tension along its length caused by fixing the nodes of one end and applying a normal force to all surface triangles on the other end. To avoid rotation around the pipe's axis, the positions of the nodes along one lengthwise edge are fixed so no motion is allowed out of the cut plane, while allowing displacements within the cut plane.

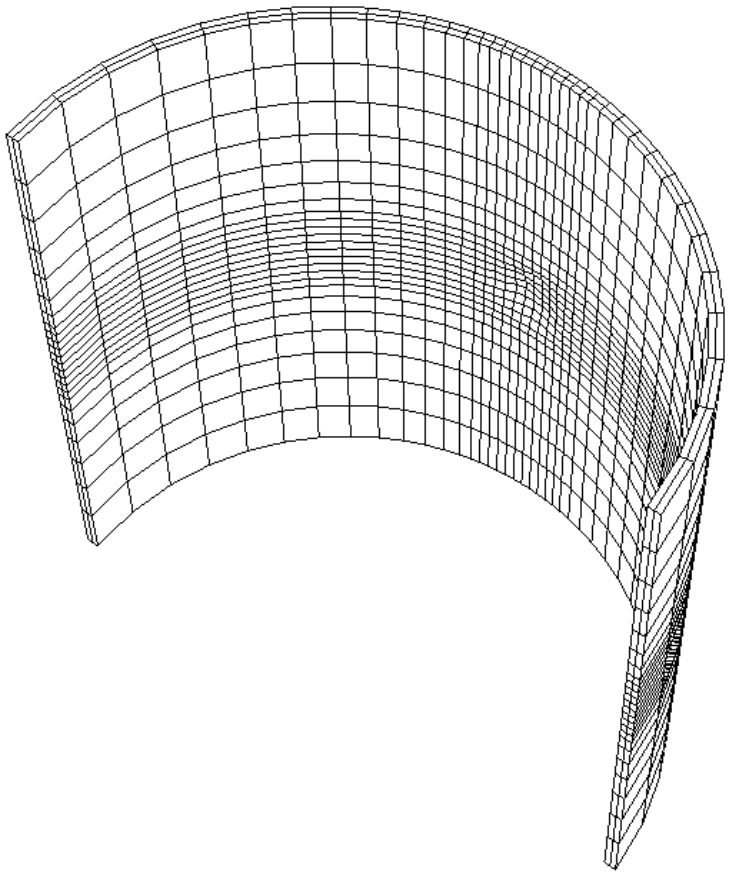


Figure 4.8: Surface Mesh of the Corroded Pipe

Chapter 5

Results

5.1 Preliminary Background

There are too many possible combinations of preconditioning matrices (for the hierarchical system) for every conceivable version to be tested. Therefore, the choice of which later experiments might prove worthwhile to investigate where based on any trends that were observed during the initial testing.

To save space and the reader's patience, detailed results are given only for a representative sample of tests rather than a complete and exhaustive catalogue of the results. Where appropriate, qualitative statements are made regarding any trends observed in test cases not reported in detail herein.

There were a couple of cases where the lack of trends in the results suggests further study needs to be done before any proper conclusions can be made. These cases will also be commented on.

It is worth emphasising that in the following tables the notation $ILU(..,..)$ will refer to an ILU factorization of the original nodal basis, while $ILU_v(..,..)$,

$ILU_m(..,..)$ will refer to the ILU of A_{vv}, and A_{mm} , as in equation (2.10). This was previously mentioned in Section 3.1 .

Unless otherwise indicated, the method of equations (3.3 and 3.4) was used to ensure a positive definite ILU factor. The parameters are the same as those given in equations (3.5). If a negative pivot was encountered on the fifth attempt at ILU factoring ($p = 5$ in equation (3.5)), the matrix solve was aborted.

All results are given in terms of total CPU time and number of iterations. The total CPU time includes all the time required to symbolically and numerically ILU factor the matrix (these steps are combined when a drop tolerance ILU is used), and the iteration cost (forward and backward solves, gradient acceleration work). In addition, the total CPU time includes all the time required for repeated attempts to perform the ILU factorization (equations (3.3 and 3.4)) to ensure positive pivots. The CPU time does not include the time required for construction of the stiffness matrix. CPU times are reported for a SUN 670/MP.

The initial tests were done on the *Cube* problem because of the ease with which both the material properties of the *Cube* and the mesh size being used could be changed. Any discovered trends were further investigated using both the *Cube* problem and the more complex geometry problems. The test results for the *Cube* will thus be presented first followed by the experimental results for problems with more complex geometries.

Table 5.1: P0 Preconditioning for Cube 4x4x4

	CPU seconds (iterations)		
Preconditioning	$l/lz = 1.0$	$l/lz = 10$	$l/lz = 100$
$ILU(1, 0.0)$	8.48 (18)	24.92 (88)	124.26 (649)
J & M add	16.23 (52)	44.64 (217)	*
$ILU(2, 0.0)$	13.17 (14)	19.81 (40)	112.2 (469)
J & M Add	18.34 (27)	33.9 (101)	115.7 (485)
$ILU(\infty, 10^{-4})$	16.58 (6)	14.68 (16)	67.33 (357)
J & M Add	25.62 (23)	35.78 (99)	111.47 (625)
$ILU(\infty, 10^{-5})$	16.51 (3)	15.98 (6)	63.75 (271)
J & M Add	20.59 (8)	29.1 (35)	76.5 (312)

**Did not converge in 1000 iterations*

5.2 Test Results

5.2.1 Cube Results

Some representative results for the *Cube* problem are given in Tables 5.1, 5.2, 5.3 and 5.4 . In both Tables 5.1 and 5.2, the line *J & M Add* refers to the procedure used in [33, 40, 1] (described in Section 3.1) applied to the preconditioning method listed in the line above. Table 5.1 clearly shows that for $\mathbf{P_0}$ preconditioning (equation (3.6)), the number of iterations increases drastically as the *Cube*'s aspect ratio becomes smaller (decreases). Similar behaviour has been noted by Jung, Langer & Semmler and Poole, Knight & Dale [35, 42] and is expected because of the correspondingly poorer condition number of the matrix.

Tables 5.2 and 5.3 indicate that $\mathbf{P_2}$ preconditioning (equation(3.9)) requires on average about one half the iterations as did a similar $\mathbf{P_1}$ preconditioning. However, since each $\mathbf{P_2}$ preconditioning iteration required more than twice the work of a $\mathbf{P_1}$ preconditioning iteration, the $\mathbf{P_2}$ method always required more total CPU time than an equivalent $\mathbf{P_1}$ technique. Consequently, there does not appear to be any advantage to using a $\mathbf{P_2}$ preconditioning, even for problems with poor aspect ratios

$(l/lz \gg 1)$.

Table 5.2 shows that for problems with aspect ratios near unity, a diagonal scaling preconditioning $ILU_m = diag(A_{mm})$ was quite effective. However, the performance of this method degrades severely as l/lz increased. Note that the very accurate ILU factorizations of A_{mm}, $ILU_m(\infty, 0.0)$ and $ILU_m(\infty, 10^{-5})$ are more robust (fewer iterations) and require less CPU time than the less accurate ILU factorizations of A_{mm} as l/lz increases. This is not surprising as it has been noted by Jung, Langer & Semmler [35] that systems of equations based on "elongated" elements require more accurate ILU preconditioners to achieve good convergence rates.

Tables 5.1 and 5.2 indicate that the *J & M Add* technique [33] became slower as the aspect ratio deteriorated (became smaller) for relatively inaccurate ILU factorizations. During inaccurate ILU factorizations, the discarded fill terms are fairly large, so that the amount added to the diagonal is also quite large. This tends to cause an iterative method which uses *J & M Add* to slow down. However, as the ILU becomes more accurate (higher level or smaller drop tolerance), the amount added to the diagonal is smaller, and hence this technique becomes more competitive.

Tables 5.1 and 5.2 also demonstrate a markedly poorer performance of the *J & M Add* technique [33] overall. This trend is not as clear in later test results making a conclusion hard to reach.

Representative computations for the $10 \times 10 \times 10$ *Cube* problem are given in Table 5.4 . The level based ILU $\mathbf{P_0}$ methods are competitive with $\mathbf{P_1}$ preconditionings for l/lz small, but are poor for $l/lz \gg 1$. As for the smaller version of the *Cube* problem, the preconditioners $ILU_m = diag(A_{mm}), ILU_m(0, 0.0)$ provide worse performance as the aspect ratio becomes more extreme ($\ll 1$). This suggests again that more

Table 5.2: P1 Preconditioning for Cube 4x4x4

	CPU seconds (iterations)		
Preconditioning	$l/lz = 1.0$	$l/lz = 10$	$l/lz = 100$
$ILU_v(\infty, 0.0)$ $ILU_m = diag(A_{mm})$	5.78 (58)	38.62 (431)	*
$ILU_v(\infty, 0.0)$ $ILU_m(1, 0.0)$ *J & M Add*	6.9 (38) 7.6 (40)	9.69 (54) 15.2 (99)	47.24 (346) 114.4 (870)
$ILU_v(\infty, 0.0)$ $ILU_m(2, 0.0)$ *J & M Add*	8.6 (36) 9.63 (38)	9.71 (43) 13.2 (62)	52.8 (329) 77.2 (493)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 0.0)$	17.22 (34)	16.25 (37)	29.19 (105)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-4})$ *J & M Add*	12.59 (35) 15.6 (37)	11.199 (40) 14.7 (52)	40.72 (285) 58.0 (421)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-5})$ *J & M Add*	17.22 (34) 20.7 (36)	14.96 (38) 18.77 (43)	27.44 (152) 43.9 (262)

**Failure to converge after 1000 iterations.*

Table 5.3: P2 Preconditioning for Cube 4x4x4

	CPU seconds (iterations)		
Preconditioning	$l/lz = 1.0$	$l/lz = 10$	l/lz = 100
$ILU_v(\infty, 0.0)$ $ILU_m(1, 0.0)$	7.9 (21)	15.02 (46)	71 (251)
$ILU_v(\infty, 0.0)$ $ILU_m(2, 0.0)$	9.7 (20)	10.95 (24)	72.8 (211)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 0.0)$	16.6 (17)	17.46 (19)	32.55 (54)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-4})$	14.3 (19)	12.38 (21)	56.1 (190)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-5})$	19.27 (18)	16.5 (20)	49.8 (135)

Table 5.4: P0, P1 Preconditioning for Cube 10x10x10

vv	CPU seconds (iterations)		
Preconditioning	$l/lz = 1.0$	$l/lz = 10$	l/lz = 100
P0			
$ILU(0, 0.0)$	557 (153)	**	**
$ILU(1, 0.0)$	513 (67)	2033 (394)	***
$ILU(2, 0.0)$	964 (43)	1954 (187)	*
$ILU(\infty, 10^{-1})$	540 (154)	**	**
$ILU(\infty, 10^{-2})$	391 (84)	**	**
$ILU(\infty, 10^{-3})$	521 (44)	1869 (344)	*
$ILU(\infty, 10^{-4})$	539 (154)	1856 (127)	*
$ILU(\infty, 10^{-5})$	539 (154)	3756 (112)	*
P1			
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 0.0)$	***	***	***
$ILU_v(\infty, 0.0)$ $ILU_m = diag(A_{mm})$	421 (58)	1172 (323)	***
$ILU_v(\infty, 0.0)$ $ILU_m(0, 0.0)$	662 (42)	825 (36)	**
$ILU_v(\infty, 0.0)$ $ILU_m(1, 0.0)$	457 (39)	512 (49)	1316 (248)
$ILU_v(\infty, 0.0)$ $ILU_m(2, 0.0)$	584 (39)	563 (36)	1632 (244)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-1})$	298 (41)	**	**
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-2})$	312 (39)	**	**
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-3})$	427 (39)	437 (48)	1404 (315)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-4})$	847 (38)	633 (35)	1378 (240)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-5})$	2182 (38)	1243 (35)	1676 (240)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-6})$	***	2650 (34)	1311 (92)

**Failure to converge after 1000 iterations.*

***Failed due to negative pivot in ILU (five attempts).*

****CPU time limit (4000 seconds) exceeded.*

accurate factorizations of A_{mm} are required as l/lz increases.

Table 5.4 also shows an interesting phenomenon. The very small drop tolerance ILU factorizations of A_{mm} require more CPU time for cubes having aspect ratio near unity compared to problems with poor aspect ratios ($l/lz \gg 1$). When using a given (small) drop tolerance, the larger CPU costs are caused by more fill being generated for $l/lz = 1$ than for $l/lz \gg 1$. For anisotropic M-matrix problems, this effect has been noted previously by D'Azevedo, Forsyth and Tang[24], however, it was noted in [24] that this effect is strongly dependent on the ordering of the unknowns. It would appear that for problems with small ($\ll 1$) aspect ratios, a small drop tolerance ILU is more effective than for problems with aspect ratios ~ 1.

Table 5.5 repeats many of the tests given in Table 5.4, except that the *J & M Add* method was used to prevent negative pivots (instead of adding an *a priori* number to the diagonal (equations (3.3 and 3.4)). Generally, the *J & M Add* method was slower than the method of equations (3.3 and 3.4), even if the ILU was attempted several times before no negative pivots were encountered. However, the poorer performance by the *J & M Add* method was not as consistent as found in the $4 \times 4 \times 4$ *Cube* test cases (Tables 5.1 and 5.2).

In summary, it seems that a robust method for various aspect ratios uses a $\mathbf{P_1}$ preconditioning with

$$ILU_m(1, 0.0) \ , \ ILU_m(2, 0.0)$$
$$ILU_m(\infty, 10^{-3}) \ , \ ILU_m(\infty, 10^{-4})$$

preconditionings for A_{mm}, and an exact factorization of A_{vv}. If it is known that the aspect ratio is near 0.70 (*Cube* $l/lz = 1$) or better, then $\mathbf{P_1}$ with $ILU_m = diag(A_{mm})$ is effective, as are $\mathbf{P_0}$ methods.

Table 5.5: P1 Preconditioning for Cube 10x10x10 with J & M Add

	CPU seconds (iterations)		
Preconditioning	$l/lz = 1.0$	$l/lz = 10$	l/lz = 100
$ILU_v(\infty, 0.0)$ $ILU_m(0, 0.0)$	406 (43)	867 (179)	*
$ILU_v(\infty, 0.0)$ $ILU_m(1, 0.0)$	475 (40)	617 (79)	*
$ILU_v(\infty, 0.0)$ $ILU_m(2, 0.0)$	635 (39)	697 (52)	3476 (626)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-1})$	316 (43)	696 (146)	*
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-2})$	333 (42)	567 (112)	*
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-3})$	440 (39)	524 (68)	3549 (924)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-4})$	1071 (39)	774 (40)	2168 (441)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-5})$	3331 (39)	1763 (35)	1618 (214)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-6})$	***	3734 (35)	1796 (118)

**Failure to converge after 1000 iterations.*
****CPU time limit (4000 seconds) exceeded.*

Further tests were carried out to determine the effect of having a nonconstant Young's modulus (E) within the *Cube*. A problem with $l/lz = 10$, and $\mathbf{P_1}$ preconditioning with $ILU_m(2, 0.0)$ and $ILU_v(\infty, 0.0)$ was used. Even if E had large jump discontinuities of several orders of magnitude, there was very little effect on convergence.

A similar test was carried out to determine the effect of varying Poisson's ratio. ν was kept constant in the *Cube*, but different values of ν were used in different tests. The performance of the iterative methods was highly dependent on ν as expected [35]. For example, the problem with $\nu = 0.49999$ required about 15 times

Table 5.6: P0 Preconditioning for Complex Geometry Problems

	CPU seconds (iterations)				
Preconditioning	Balljoint	Block & Prism	Bushing	Multi-Material Bracket	Corroded Pipe
$ILU(\infty, 10^{-1})$	435 (109)	**	4135 (511)	**	**
J & M Add	830 (219)	896 (281)	7845 (994)	6686 (845)	*
$ILU(\infty, 10^{-2})$	300 (57)	294 (64)	2151 (222)	1979 (190)	**
J & M Add	790 (192)	747 (210)	7406 (853)	5575 (659)	*
$ILU(\infty, 10^{-3})$	448 (33)	395 (32)	2173 (114)	1478 (69)	11684 (789)
J & M Add	892 (120)	894 (146)	6782 (530)	4868 (375)	*
$ILU(\infty, 10^{-4})$	1283 (19)	1018 (16)	4162 (59)	2929 (32)	7178 (319)
J & M Add	1703 (64)	1510 (76)	8064 (281)	6225 (202)	*
$ILU(\infty, 10^{-5})$	4951 (9)	2535 (6)	19516 (48)	13328 (29)	10790 (311)
J & M Add	5299 (33)	3401 (33)	15941 (131)	11995 (92)	13567 (504)
$ILU(0, 0.0)$	**	**	3707 (451)	**	**
J & M Add	805 (220)	819 (253)	*	6818 (851)	*
$ILU(2, 0.0)$	489 (47)	411 (49)	2730 (193)	2077 (125)	**
J & M Add	856 (103)	822 (124)	6106 (482)	5479 (399)	*
$ILU(2, 0.0)$	1041 (28)	801 (27)	4175 (118)	3257 (68)	**
J & M Add	1583 (56)	1245 (61)	7062 (248)	6596 (215)	*

**Failure to converge after 1000 iterations.*
***Failed due to negative pivot in ILU (five attempts).*

the number of iterations as for the same problem with $\nu = 0.40$.

5.2.2 Results from more Geometrically Complex Problems

The test results from the problems having more complex geometries basically confirm the conclusions developed in the previous section. Tables 5.6, 5.7 and 5.8 contain some representative results.

A comparison of Table 5.6 ($\mathbf{P_0}$ preconditioning) and Table 5.7 ($\mathbf{P_1}$ preconditioning with direct solve of A_{vv}) clearly shows that using a hierarchical basis provides

generally better computation times. For one choice of preconditioning parameters the $\mathbf{P_0}$ preconditioning method does give slightly better performance for the two smaller problems, *Balljoint* and *Block & Prism*. However, this performance gain was not repeated for any of the larger test problems. In particular, $\mathbf{P_0}$ was an extremely poor technique for application to the *Corroded Pipe* problem (worse aspect ratio).

The best preconditioners seem to be based on using an exact factorization for A_{vv} and $ILU_m(\infty, 10^{-1})$, $ILU_m(\infty, 10^{-2})$, or $ILU_m(\infty, 10^{-3})$ for A_{mm}. Smaller drop tolerances were required for worse (smaller) aspect ratios (see Table 4.1), as would be expected [35]. As the complex geometry problems do not have as extreme aspect ratios as were used in the *Cube* problems, the recommended drop tolerance is larger. In general, the most robust method in these tests was found to be preconditioning by $ILU_m(\infty, 10^{-3})$ (and exact factorization of A_{vv}).

Unlike the results from the previous section, it is not clear that a level based drop tolerance would provide good results. The test results indicate that a level based preconditioning technique is almost always out-performed by the suggested drop tolerance technique ($\mathbf{P_1}$, $ILU_m(\infty, 10^{-3})$, $ILU_v(\infty, 0.0)$) and that the level based preconditioning provides less consistent results across the tested range of aspect ratios.

Given more information about the equations to be solved, it is possible to improve on these techniques. As discussed earlier, experimentation demonstrates two dominant factors that affect the rate of convergence of the iterative techniques. They are the average aspect ratio of the elements in the mesh and the value of ν. However, the value of ν seems to have a uniform affect on all iterative techniques leaving only the element aspect ratios to be considered.

Table 5.7: P1 Preconditioning with Direct Solve of A_{vv} for Complex Geometry Problems

	CPU seconds (iterations)				
Preconditioning	Balljoint	Block & Prism	Bushing	Multi-Material Bracket	Corroded Pipe
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-1})$	371 (68)	354 (96)	1436 (57)	1355 (87)	**
J & M Add	390 (69)	356 (89)	1706 (70)	1237 (74)	4159 (321)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-2})$	381 (66)	349 (86)	1469 (55)	1236 (71)	7063 (554)
J & M Add	402 (68)	380 (90)	1615 (57)	1255 (72)	2550 (168)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-3})$	496 (66)	465 (85)	1726 (52)	1521 (71)	1889 (105)
J & M Add	542 (67)	497 (88)	1888 (53)	1565 (71)	2068 (111)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-4})$	818 (65)	746 (79)	2656 (52)	2373 (70)	2103 (102)
J & M Add	1017 (66)	936 (86)	3400 (53)	2591 (70)	2268 (106)
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-5})$	1653 (65)	1383 (79)	5024 (52)	4018 (70)	2508 (100)
J & M Add	2320 (65)	2013 (86)	6762 (52)	5119 (71)	2714 (104)
$ILU_v(\infty, 0.0)$ $ILU_m(0, 0.0)$	466 (69)	384 (96)	2099 (58)	**	**
J & M Add	462 (69)	389 (89)	2468 (79)	1593 (75)	5257 (409)
$ILU_v(\infty, 0.0)$ $ILU_m(1, 0.0)$	542 (66)	444 (88)	2211 (54)	1733 (71)	6393 (438)
J & M Add	554 (67)	483 (87)	2271 (53)	1788 (71)	2261 (111)
$ILU_v(\infty, 0.0)$ $ILU_m(2, 0.0)$	740 (65)	597 (81)	2620 (53)	2170 (71)	2574 (112)
J & M Add	799 (66)	705 (86)	2833 (53)	2312 (70)	2678 (109)
$ILU_v(\infty, 0.0)$ $ILU_m = diag(A_{mm})$	403 (98)	422 (153)	1883 (121)	1374 (116)	6900 (731)

***Failed due to negative pivot in ILU (five attempts).*

For the test case meshes generated by the I-DEAS VI: Mesh Generation Package [21], it is noted that the average aspect ratio was always approximately 0.77 . With these aspect ratios, the maximum rate of convergence using a preconditioner with an exact factorization of A_{vv} was attained by using a coarse drop tolerance of $ILU_m(\infty, 10^{-1})$ or, for fast results and minimum space requirements, using a $ILU_m = diag(A_{mm})$. However, even these rapid convergence rates can be improved upon.

Table 5.8 shows results from incomplete factorization of both A_{vv} and A_{mm}. Examination of these results shows that for average aspect ratios of 0.59 or more, very fast convergence can be attained by a $\mathbf{P_1}$ preconditioning with $ILU_v(\infty, 10^{-1})$ or $ILU_v(\infty, 10^{-2})$, and $ILU_m(\infty, 10^{-3})$ or $ILU_m(\infty, 10^{-4})$. This technique was not effective for worse (smaller) aspect ratios.

Extensive testing of the *J & M Add* [33] method was also done. Table 5.7 shows the results for a $\mathbf{P_1}$ preconditioning with direct solve of A_{vv} with and without using the *J & M Add* technique. This technique demonstrates several positive features. For these test cases, it usually only resulted in a small increase in convergence times for the complex geometry problems. For the three cases where attempts to avoid a negative pivot by adding an *a priori* number to the diagonal failed, the *J & M Add* technique did succeed. In several cases, faster convergence times were observed for specific tests of the *Corroded Pipe* and *Multi-Material Bracket* problems. However, the *J & M Add* method was never faster than the recommended $\mathbf{P_1}$ technique (direct solve of A_{vv}, $ILU_v(\infty, 10^{-3})$) alone (Table 5.7).

In general, there is little reason to use the *J & M Add* method for problems with average aspect ratios of ~ 0.59 and better or ~ 0.02 (*Cube* with $l/lz = 100$) and worse. For problems in between these extremes (*Multi-Material Bracket* & *Corroded Pipe*) the results seem to indicate that the method may provide reasonable

Table 5.8: P1 Preconditioning with ILU Factoring of Both Domains for Complex Geometry Problems

	CPU seconds (iterations)			
Preconditioning	Block & Prism	Bushing	Multi-Material Bracket	Corroded Pipe
$ILU_v(\infty, 10^{-1}), ILU_m(\infty, 10^{-1})$	294 (114)	1423 (204)	**	**
$ILU_v(\infty, 10^{-2}), ILU_m(\infty, 10^{-1})$	263 (99)	1686 (222)	**	**
$ILU_v(\infty, 10^{-3}), ILU_m(\infty, 10^{-1})$	277 (96)	876 (107)	723 (87)	**
$ILU_v(\infty, 10^{-4}), ILU_m(\infty, 10^{-1})$	310 (96)	1039 (81)	775 (87)	**
$ILU_v(\infty, 10^{-1}), ILU_m(\infty, 10^{-2})$	315 (112)	1510 (202)	**	**
$ILU_v(\infty, 10^{-2}), ILU_m(\infty, 10^{-2})$	270 (92)	1845 (223)	**	**
$ILU_v(\infty, 10^{-3}), ILU_m(\infty, 10^{-2})$	282 (89)	929 (105)	698 (73)	**
$ILU_v(\infty, 10^{-4}), ILU_m(\infty, 10^{-2})$	305 (86)	1089 (78)	705 (71)	5669 (543)
$ILU_v(\infty, 10^{-1}), ILU_m(\infty, 10^{-3})$	445 (110)	2044 (193)	**	**
$ILU_v(\infty, 10^{-2}), ILU_m(\infty, 10^{-3})$	385 (90)	2695 (219)	**	**
$ILU_v(\infty, 10^{-3}), ILU_m(\infty, 10^{-3})$	398 (88)	1238 (101)	1224 (72)	**
$ILU_v(\infty, 10^{-4}), ILU_m(\infty, 10^{-3})$	414 (83)	1541 (75)	1021 (70)	3226 (247)
$ILU_v(\infty, 10^{-1}), ILU_m(\infty, 10^{-4})$	787 (109)	3692 (191)	**	**
$ILU_v(\infty, 10^{-2}), ILU_m(\infty, 10^{-4})$	703 (90)	5976 (219)	**	**
$ILU_v(\infty, 10^{-3}), ILU_m(\infty, 10^{-4})$	709 (87)	2398 (101)	2410 (72)	**
$ILU_v(\infty, 10^{-4}), ILU_m(\infty, 10^{-4})$	732 (85)	3282 (75)	1776 (71)	3675 (246)
$ILU_v(0, 0.0), ILU_m(0, 0.0)$	**	1404 (211)	**	**
$ILU_v(1, 0.0), ILU_m(0, 0.0)$	273 (101)	1094 (157)	**	**
$ILU_v(2, 0.0), ILU_m(0, 0.0)$	283 (98)	980 (126)	**	**
$ILU_v(0, 0.0), ILU_m(1, 0.0)$	**	1674 (201)	**	**
$ILU_v(1, 0.0), ILU_m(1, 0.0)$	335 (94)	1294 (147)	1112 (112)	**
$ILU_v(2, 0.0), ILU_m(1, 0.0)$	339 (90)	1128 (117)	1007 (86)	5560 (494)
$ILU_v(0, 0.0), ILU_m(2, 0.0)$	**	2380 (200)	**	**
$ILU_v(1, 0.0), ILU_m(2, 0.0)$	495 (92)	1864 (145)	1682 (111)	**
$ILU_v(2, 0.0), ILU_m(2, 0.0)$	505 (90)	1644 (116)	1592 (84)	4750 (330)

***Failed due to negative pivot in ILU (five attempts).*

computation times with more stability than the recommended $\mathbf{P_1}$ technique alone (Table 5.7). No firm conclusion could be drawn from just these test results.

When the *J & M Add* method was combined with a $\mathbf{P_1}$ (Table 5.7) approach with ILU preconditioning for both A_{vv} and A_{mm}, convergence would generally take twice as long or more, depending on the average element aspect ratio. This was also found to be true when the *J & M Add* method was used with a $\mathbf{P_0}$ (Table 5.6) preconditioning technique. There is some evidence the *J & M Add* method improved the robustness of the both the $\mathbf{P_0}$ and $\mathbf{P_1}$ technique, but only for non-recommended ILU's (i.e. $ILU(0, 0.0)$, $ILU(\infty, 10^{-1})$).

Some experiments were also carried out using a iterative method to carry out the operation in equation (3.12), for the midpoint block (A_{mm}), and an ILU preconditioner. This performed very poorly compared with simply using an ILU of A_{mm}, and hence no results are given.

It is worth noting the both the *Block & Prism* and *Multi-Material Bracket* problems contained two different materials (Table 4.1). For both test cases, the physical properties ν and E differed between the component materials. However, there was no indication of any adverse effect on the performance of any of the iterative solving techniques.

In summary, a $\mathbf{P_1}$ preconditioning with an exact factorization of A_{vv} and a drop tolerance of $ILU_m(\infty, 10^{-2})$ or $ILU_m(\infty, 10^{-3})$ appears to be a generally robust method for a range of average element aspect ratios. A smaller drop tolerance should be favoured as the aspect ratio becomes worse. When the element aspect ratio is known to be above 0.58, even better results can be had by using incomplete factorization for both A_{vv} and A_{mm}. In this case having $ILU_v(\infty, 10^{-1})$ or $ILU_v(\infty, 10^{-2})$, and $ILU_m(\infty, 10^{-3})$ or $ILU_m(\infty, 10^{-4})$ is recommended.

Note that the above remarks are concerned solely with average element aspect ratio. The minimum aspect ratios for the *Balljoint* and *Block & Prism* problems were much smaller than the minimum (which was also the average in this case) aspect ratio for the *Cube* problem with $l/lz = 100$, but clearly the *Cube* problem was much more difficult to solve than the former two problems.

5.3 Solution Accuracy

Having compared the CPU costs of the solutions and determined the most robust iterative technique, the actual achieved accuracy of the generated solutions should also be considered.

For the iterative tests, convergence was assumed to have occurred when the initial residual was reduced by a specified amount. However, the residuals during the tests where calculated by using vector terms found during the conjugate gradient process. Thus the residual vector was a sum of vectors rather than being explicitly calculated by $\mathbf{r} = \mathbf{b} - A\mathbf{x}$. It common knowledge that a residual determined during the conjugate gradient step can become a quite poor approximation of the current residual. To avoid misleading results when comparing solutions, a true residual ($\mathbf{r} = \mathbf{b} - A\mathbf{x}$) was often calculated upon convergence. When residuals are compared in the following subsections, this true residual was used.

The same residual was also calculated for the solutions found by using a direct solver for comparison to the true iterative residuals.

5.3.1 Iterative vs. Direct Solve Results

A direct solve was attempted for the $10 \times 10 \times 10$ *Cube* ($l/lz = 10$) and for each of the five complex geometry problems using the Yale Sparse Matrix Package [28]. Minimum degree ordering was used for each of the direct solves. To match the iterative solutions, the direct solutions were also calculated in single precision. Despite only using single precision storage, memory constraints (maximum of 128 Meg available) limited the testing of the direct method to only three of the five complex geometry test problems (Table 5.9).

For the *Balljoint*, *Multi-Material Bracket*, and *Cube* problems, the direct solve cost from six to seventeen times the CPU time to complete compared to the recommended iterative technique (direct solve of A_{vv}, $ILU_v(\infty, 10^{-3})$). The direct solve of the *Block & Prism* required only three times the CPU time compared to the iterative technique because of the small amount of fill which occurred. The remaining two problems (*Bushing* and *Pipe*) required too much memory for a direct solve to be performed.

Table 5.9 shows both the solution times and the amount of storage required for the direct and iterative factored matrices. It can easily be seen that the iterative factored matrix requires only about a fifth of the storage as needed by the direct solver as well as taking less CPU time.

The solutions for each problem from both the direct and the iterative approaches were compared. This comparison was done by calculating:

$$\frac{\|X_{direct} - X_{iterative}\|_\infty}{\|X_{max}\|_\infty}, \frac{\|Y_{direct} - Y_{iterative}\|_\infty}{\|Y_{max}\|_\infty}, \frac{\|Z_{direct} - Z_{iterative}\|_\infty}{\|Z_{max}\|_\infty}$$

where:

- X, Y, and Z are vectors of the calculated x, y, and z displacements for each

Table 5.9: Direct Solve vs. Suggested P1 Technique

	CPU seconds (iterations)					
Preconditioning	$10 \times 10 \times 10$ Cube ($l/lz = 10$)	Balljoint	Block & Prism	Bushing	Multi-Material Bracket	Corroded Pipe
$ILU_v(\infty, 0.0)$ $ILU_m(\infty, 10^{-3})$	437 (48)	496 (66)	465 (85)	1726 (52)	1521 (71)	1889 (105)
Memory	2,886,150	2,946,718	2,494,046	9,095,466	7,656,678	7,750,416
Direct [28]	7771	3091	1361	*	14170	*
Solve Memory	16,122,339	11,151,729	7,350,827	*	34,433,489	*

**Insufficient memory available to complete a direct solve.*

Memory *is the number of single precision words required to store the matrix factors and integer pointers.*

node,

- subscripts *direct* and *iterative* refer to method used to calculate displacement vector, and
- subscript *max* means the maximum displacement from both the iterative and direct solutions.

For all problems, the maximum disagreement between solutions was less than 0.001 (or 0.1%) and was often as small as 0.01%. Thus the iterative solution can be expected to have roughly three or more significant digits of the direct solution, with the applied tolerance (equation (4.1)).

In fact, when the true residuals for both the direct method and the iterative technique, using single precision storage, were compared the residual for the direct method was no better than the true residual of the iterative technique. The direct method's residual was at best almost identical to the iterative residual and at worst

almost 100 times larger.

From these facts it can be concluded that the recommended iterative solver will find at least a comparable solution to the direct method while providing a large CPU cost savings and using less memory.

5.3.2 Single vs. Double Precision

During all the previous tests, only single precision storage was used to store the matrix terms to help reduce memory demands. The iterative tests were considered converged after a reduction of the initial residual by a factor of 10^{-6}. However, some tests were made with higher residual reductions to measure the effectiveness of the 10^{-6} criteria.

Comparison of the true residuals for both a residual reduction of 10^{-6} and 10^{-9} demonstrated that no further accuracy was actually achieved in the hopefully higher precision solution. In other words, when using single precision storage, a residual reduction of 10^{-6} has driven the solution to the limits of the attainable accuracy because of round-off.

However some tests were also made while using double precision storage. These tests show that it is possible to achieve a residual reduction of approximately 10^{-14} before round-off makes further gains negligible. To achieve this residual reduction requires a little less than 2.5 times the CPU time compared to a single precision residual reduction of 10^{-6}.

Chapter 6

Convergence Criteria

6.1 Introduction

The convergence criteria used to determine the results given in the previous chapters was a simple reduction of the initial residual by a user specified factor and also a user set limit for a maximum number of iterations. A residual reduction of 10^{-6} has in general, provided a good solution by driving the iterative scheme to a point where no further improvement could be seen in the solution vector due to numerical round-off. This rather coarse accuracy limit is because of the matrix terms were only stored using single, rather than double precision. More accurate results could be achieved by storing all terms in double precision.

However, it is often the case that a solution with a specific accuracy is required. Error estimates can be evaluated for a solution after it has been calculated. Ohtsubo and Kitamura [41] present an *a posteriori* method for determining and possibly improving the error of a solution to a finite element analysis of a three dimensional elastic problem. The error estimate is calculated by solving the FEA equilibrium

equations for each element using the next higher order basis than was used for the initial stress analysis.

Determining the error of a solution after it is created is not sufficient for current requirements though. Time is money on large computers, and a stopping criteria which terminates the iterative process as soon as a desired accuracy is achieved could save much valuable computation time. But, a premature termination could lead to even worse consequences which may later cost more than just a few more minutes of CPU time.

Thus, a stopping criteria mustn't be too optimistic and give misleading results, nor be so pessimistic that valuable computation time is wasted. A criteria must also be relatively inexpensive to check or the cost of deciding when to halt computations will begin to inhibit the efficiency of the iterative solver. More expensive stopping criteria can be used, but checks should be made less frequently. Instead of checking the criteria after each step, checks could be made every two or more steps.

Arioli, Duff and Ruiz [3] suggest a stopping criteria for iterative solvers based on the quantity ω defined by:

$$\begin{aligned} \omega &= \max_i \frac{|\mathbf{r}(\hat{\mathbf{x}})_i|}{(\mathbf{E}|\hat{\mathbf{x}}| + \mathbf{f})_i} \\ &\quad \text{where: } \mathbf{r}(\hat{\mathbf{x}}) = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}} \ . \end{aligned}$$

Iteration is terminated when ω is reduced to a specified size or has clearly ceased to decrease. However, this technique is very dependent on the choice of $\mathbf{E}$ and $\mathbf{f}$. Several possible assignments are suggested but Arioli, Duff and Ruiz conclude that setting

$$\omega = \frac{\|\mathbf{r}(\hat{\mathbf{x}})_i\|_\infty}{(\|\mathbf{A}\|_\infty \|\hat{\mathbf{x}}\|_\infty + \|\mathbf{b}\|_\infty)}$$

was the best of their presented criteria. Despite being easy to compute, the question of what the termination value of ω should be and how to decide if ω has ceased to decrease remains somewhat open.

Kaasschieter [36] offers a technique based on estimating the smallest eigenvalue (μ_1) of a matrix from the information gained during the application of an iterative conjugate gradient method. The principle advantage of Kaasschieter's method is that the stopping criteria is already expressed in terms of the desired accuracy of the solution. Theoretically, the criteria is designed to halt the iterative procedure when $\|\mathbf{x}-\mathbf{x}_i\|_2/\|\mathbf{x}\|_2 \leq \varepsilon$ (where $\mathbf{x}$ is the exact solution vector and $\mathbf{x}_i$ is the solution vector after the i^{th} step) thus resulting in an solution with an accuracy of ε.

This technique was implemented because of the expected simplicity of adding it to the program's existing PCG algorithms and also because of the direct formulation of the terminating criteria in terms of the required accuracy. Unfortunately, implementation of the algorithm was not simple because the accurate estimation of the 2-norm of a triangular matrix turned out to be non-trivial.

6.2 Theory

Before discussing the necessary theory, a quick note is needed to explain the notation conventions used. All norms in the following theory and application are 2-norms. It should also be mentioned that the vectors $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ are exact solution vectors and $\mathbf{x}_i$, $\mathbf{y}_i$, $\mathbf{z}_i$ are the approximate solution vectors found after the i^{th} iteration.

Kaasschieter's technique is meant for direct application to conjugate gradient code and thus requires modifications before being used with a preconditioned con-

jugate gradient system.

As stated earlier the desired stopping criteria is

$$\|\mathbf{x} - \mathbf{x}_i\|/\|\mathbf{x}\| \quad \leq \quad \varepsilon \tag{6.1}$$

where $\mathbf{x}$ is the exact solution vector and $\mathbf{x}_i$ is the solution vector after the i^{th} step. As the exact solution vector is unknown, the inequality will be replaced by using the following theorem.

Theorem 6.1 *If* $\|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x}_i\|\varepsilon/(1+\varepsilon)$, *then* $\|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x}\|\varepsilon$.

This theorem is proven in Kaasschieter's paper and allows for a more easily developed bound of the solution vector's error.

From Kaasschieter's paper, the conjugate gradient algorithm (after i iterations):

$$\mathbf{x} - \mathbf{x}_i \;=\; \mathbf{A}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_i) \;=\; \mathbf{A}^{-1}\mathbf{r}_i \tag{6.2}$$

and therefore:

$$\|\mathbf{x} - \mathbf{x}_i\| \;=\; \|\mathbf{A}^{-1}\mathbf{r}_i\| \;\leq\; \|\mathbf{r}_i\|/\mu_1 \tag{6.3}$$

where μ_1 is the minimum eigenvalue for $\mathbf{A}$ and is used to approximate $\|\mathbf{A}^{-1}\|$. Thus if $\|\mathbf{r}_i\|/\mu_1$ is made to be less than $\|\mathbf{x}_i\|\varepsilon/(1+\varepsilon)$ by sufficient iterations then the inequality:

$$\|\mathbf{x} - \mathbf{x}_i\| \quad \leq \quad \|\mathbf{r}_i\|/\mu_1 \quad \leq \quad \|\mathbf{x}_i\|\varepsilon/(1+\varepsilon) \tag{6.4}$$

is satisfied and it follows from Theorem 6.1 that the desired accuracy has been achieved.

What is needed is to construct an equivalent of equation (6.3) for the scaled preconditioned conjugate gradient system which can be used to satisfy a similar

inequality as in equation (6.4). To start, this requires the expression and manipulation of the equations for the preconditioned scaled conjugate gradient system, as are found in the existing test code. First, the scaled conjugate gradient system is:

$$\mathbf{DADD}^{-1}\mathbf{x} \;=\; \mathbf{Db} \quad \text{or} \quad \tilde{\mathbf{A}}\mathbf{y} \;=\; \tilde{\mathbf{b}} \tag{6.5}$$

where $\tilde{\mathbf{A}} = \mathbf{DAD}$, $\mathbf{y} = \mathbf{D}^{-1}\mathbf{x}$, and $\tilde{\mathbf{b}} = \mathbf{Db}$. Simple algebra shows that the residual of the new system is:

$$\tilde{\mathbf{r}}_i \;=\; \tilde{\mathbf{b}} - \tilde{\mathbf{A}}\mathbf{y}_i \;=\; \mathbf{D}^{-1}\left(\mathbf{b} - \mathbf{A}\mathbf{x}_i\right) \;=\; \mathbf{D}^{-1}\mathbf{r}_i \;.$$

When preconditioned:

$$\mathbf{L}^{-1}\tilde{\mathbf{A}}\mathbf{L}^{-t}\mathbf{L}^{t}\mathbf{y} \;=\; \mathbf{L}^{-1}\tilde{\mathbf{b}} \quad \text{or} \quad \tilde{\tilde{\mathbf{A}}}\mathbf{z} \;=\; \tilde{\tilde{\mathbf{b}}} \tag{6.6}$$

where $\tilde{\tilde{\mathbf{A}}} = \mathbf{L}^{-1}\tilde{\mathbf{A}}\mathbf{L}^{-t}$, $\mathbf{z} = \mathbf{L}^{t}\mathbf{y}$, $\tilde{\tilde{\mathbf{b}}} = \mathbf{L}^{-1}\tilde{\mathbf{b}}$, and $\mathbf{L}\mathbf{L}^{t} \sim \tilde{\mathbf{A}}$. The next residual $\tilde{\tilde{\mathbf{r}}}_i$ logically follows:

$$\tilde{\tilde{\mathbf{r}}}_i \;=\; \tilde{\tilde{\mathbf{b}}} - \tilde{\tilde{\mathbf{A}}}\mathbf{z}_i \;=\; \mathbf{L}^{-1}\tilde{\mathbf{r}}_i \;. \tag{6.7}$$

Finally, from the definition of $\mathbf{y}$ (equation (6.5)) and $\mathbf{z}$ (equation (6.6)):

$$\mathbf{x} \;=\; \mathbf{Dy} \;=\; \mathbf{DL}^{-t}\mathbf{z} \;. \tag{6.8}$$

Using the above definitions of the preconditioned conjugate gradient system, the equivalent of equation (6.3) can be derived. Combining $\tilde{\tilde{\mathbf{A}}}\mathbf{z} = \tilde{\tilde{\mathbf{b}}}$ and $\tilde{\tilde{\mathbf{A}}}\mathbf{z}_i = \tilde{\tilde{\mathbf{b}}} - \tilde{\tilde{\mathbf{r}}}_i$ yields:

$$\tilde{\tilde{\mathbf{A}}}\left(\mathbf{z} - \mathbf{z}_i\right) \;=\; \tilde{\tilde{\mathbf{r}}}_i \quad \text{or} \quad \left(\mathbf{z} - \mathbf{z}_i\right) \;=\; \tilde{\tilde{\mathbf{A}}}^{-1}\tilde{\tilde{\mathbf{r}}}_i \tag{6.9}$$

and equation (6.8) implies:

$$\left(\mathbf{z} - \mathbf{z}_i\right) \;=\; \mathbf{L}^{t}\mathbf{D}^{-1}\left(\mathbf{x} - \mathbf{x}_i\right) \;. \tag{6.10}$$

From equations (6.9) and (6.10):

$$\begin{aligned}
\mathbf{L}^t\mathbf{D}^{-1}(\mathbf{x}-\mathbf{x}_i) &= \tilde{\tilde{\mathbf{A}}}^{-1}\tilde{\tilde{\mathbf{r}}}_i \\
(\mathbf{x}-\mathbf{x}_i) &= \left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\tilde{\tilde{\mathbf{A}}}^{-1}\tilde{\tilde{\mathbf{r}}}_i \\
\|\mathbf{x}-\mathbf{x}_i\| &\leq \|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|\,\|\tilde{\tilde{\mathbf{A}}}^{-1}\|\,\|\tilde{\tilde{\mathbf{r}}}_i\| \\
\|\mathbf{x}-\mathbf{x}_i\| \,/\, \|\mathbf{x}_i\| &\leq \|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|\,\|\tilde{\tilde{\mathbf{A}}}^{-1}\|\,\|\tilde{\tilde{\mathbf{r}}}_i\| \,/\, \|\mathbf{x}_i\| \\
\|\mathbf{x}-\mathbf{x}_i\| \,/\, \|\mathbf{x}_i\| &\leq \|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|\,\mu_1^{-1}\,\|\mathbf{L}^{-1}\tilde{\mathbf{r}}_i\| \,/\, \|\mathbf{D}\mathbf{y}_i\| \qquad (6.11)
\end{aligned}$$

which is similar to equation (6.3). The reason for this form for the inequality is to make maximum use of values which are already available or easily calculated. μ_1 is calculated by Kaasschieter's algorithm and $\mathbf{y}_i$ and $\|\mathbf{L}^{-1}\tilde{\mathbf{r}}_i\|$ are generated during each iteration by the PCG routines. This leaves multiplying $\mathbf{y}_i$ by the diagonal matrix $\mathbf{D}$ and having some sort of estimate for $\|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|$. Unfortunately, the last term is the most difficult to acquire cheaply.

Direct evaluation of $\|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|$ is possible but would be extremely expensive.

$$\begin{aligned}
\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\left(\mathbf{L}^t\mathbf{D}^{-1}\right)(\mathbf{x}-\mathbf{x}_i) &= (\mathbf{x}-\mathbf{x}_i) \\
\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\mathbf{L}^t(\mathbf{y}-\mathbf{y}_i) &= (\mathbf{x}-\mathbf{x}_i) \\
\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\mathbf{L}^t(\mathbf{y}-\mathbf{y}_i) &= \mathbf{D}(\mathbf{y}-\mathbf{y}_i) \\
\|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|\,\|\mathbf{L}^t(\mathbf{y}-\mathbf{y}_i)\| &\geq \|\mathbf{D}(\mathbf{y}-\mathbf{y}_i)\| \qquad (6.12)
\end{aligned}$$

or

$$\|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\| \geq \frac{\|\mathbf{D}(\mathbf{y}-\mathbf{y}_i)\|}{\|\mathbf{L}^t(\mathbf{y}-\mathbf{y}_i)\|} \qquad (6.13)$$

forms a lower bound estimate for $\|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|$, but $\mathbf{y}$ is not known and so it is approximated with $\mathbf{y}_{i+1}$. This estimate can be calculated easily as the PCG algorithm used supplies the difference of the vectors $(\mathbf{y}_{i+1}-\mathbf{y}_i)$ after each iterative step and $\mathbf{D}$ and $\mathbf{L}^t$ are known.

Thus a stopping criteria has been formed out of equations (6.11) and (6.13):

$$\frac{\|\mathbf{x}-\mathbf{x}_i\|}{\|\mathbf{x}_i\|} \leq \frac{\|\mathbf{D}\left(\mathbf{y}_{i+1}-\mathbf{y}_i\right)\|}{\|\mathbf{L}^t\left(\mathbf{y}_{i+1}-\mathbf{y}_i\right)\|}\,\frac{1}{\mu_1}\,\frac{\|\mathbf{L}^{-1}\tilde{\mathbf{r}}_i\|}{\|\mathbf{D}\mathbf{y}_i\|} \leq \frac{\varepsilon}{(1+\varepsilon)}\,. \tag{6.14}$$

Similar to what was stated for equation (6.4), if the right hand inequality is satisfied then Theorem 6.1 can be used to show that the desired accuracy has been achieved. As $\|\mathbf{D}\left(\mathbf{y}_{i+1}-\mathbf{y}_i\right)\|/\|\mathbf{L}^t\left(\mathbf{y}_{i+1}-\mathbf{y}_i\right)\|$ provides only a lower bound estimate for $\|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|$ it becomes easier than it should be to satisfy the right hand inequality of equation (6.14). To help counter this, the maximum estimate found for $\|\left(\mathbf{L}^t\mathbf{D}^{-1}\right)^{-1}\|$ over all the iterations is used instead of the current estimate for the stopping criteria check:

$$\max_i\left(\frac{\|\mathbf{D}\left(\mathbf{y}_{i+1}-\mathbf{y}_i\right)\|}{\|\mathbf{L}^t\left(\mathbf{y}_{i+1}-\mathbf{y}_i\right)\|}\right)\,\frac{1}{\mu_1}\,\frac{\|\mathbf{L}^{-1}\tilde{\mathbf{r}}_i\|}{\|\mathbf{D}\mathbf{y}_i\|} \leq \frac{\varepsilon}{(1+\varepsilon)}\,. \tag{6.15}$$

Kaasschieter's algorithm for estimating μ_1 is explained in his paper [36] and so it will not be rewritten here. It is based on the relationship between CG methods and the Lanczos procedure. Essentially, the minimum eigenvalue (μ_1) is calculated from a sub-space of the Krylov space ($\mathrm{sp}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, ..., \mathbf{A}^{N-1}\mathbf{r}_0\}$). By using the above derived terminating criteria and the estimated minimum eigenvalue it is hoped an effective stopping criteria can be developed.

It should be noted that the algorithm presented by Kaasschieter is being applied to the scaled and preconditioned system so the resulting minimum eigenvalue estimate is not for the original matrix $\mathbf{A}$, but rather for $\tilde{\tilde{\mathbf{A}}}^{-1}$.

6.3 Results

Once implemented, Kaasschieter's algorithm was initially tested with all conjugate gradient values calculated and stored in double precision, but the $\mathbf{A}$ and factored matrices in single precision. It was soon demonstrated that the solution

Table 6.1: Desired vs. Attained Accuracy

	$10 \times 10 \times 10$ Cube ($l/lz = 1$)	$10 \times 10 \times 10$ Cube ($l/lz = 10$)	Balljoint	Block & Prism
Aspect Ratio	0.717	0.190	0.779	0.784
Desired Accuracy	Attained Accuracy (# of iterations)			
$\varepsilon = 10^{-2}$	4.385×10^{-1} (01)	4.673×10^{-1} (02)	2.762×10^{-3} (08)	5.785×10^{-4} (20)
$\varepsilon = 10^{-3}$	3.674×10^{-1} (02)	2.847×10^{-1} (04)	6.488×10^{-4} (12)	1.841×10^{-4} (25)
$\varepsilon = 10^{-4}$	2.861×10^{-1} (03)	1.252×10^{-2} (14)	2.057×10^{-4} (16)	8.950×10^{-5} (31)
$\varepsilon = 10^{-5}$	8.640×10^{-3} (10)	6.168×10^{-3} (19)	1.752×10^{-4} (19)	1.335×10^{-5} (50)
$\varepsilon = 10^{-6}$	1.242×10^{-3} (15)	1.236×10^{-3} (24)	3.452×10^{-5} (37)	1.906×10^{-6} (79)
$\varepsilon = 10^{-7}$	3.435×10^{-4} (19)	2.612×10^{-4} (31)	4.066×10^{-6} (50)	3.561×10^{-7} (94)
$\varepsilon = 10^{-8}$	6.282×10^{-5} (23)	1.549×10^{-4} (35)	2.066×10^{-6} (59)	8.017×10^{-8} (99)
$\varepsilon = 10^{-9}$	1.638×10^{-5} (27)	3.206×10^{-5} (40)	5.087×10^{-7} (67)	3.523×10^{-8} (103)

All tests done with $ILU_m(\infty, 10^{-3})$ preconditioning in double precision.

vector couldn't be iterated to an arbitrary level of accuracy due to round off errors. Each of the test problem solutions attained a maximum accuracy of roughly 0.1 % ($\|\mathbf{x} - \mathbf{x}_i\|/\|\mathbf{x}\| \leq \varepsilon = 10^{-3}$).

The accuracy of each generated solution vector was determined by comparison to a solution vector generated by a double precision iterative solve which was run until the initial residual was reduced by a factor of 10^{-12}. This solution vector was then considered to be the exact solution.

The tests were then rerun with everything stored in double precision. These results proved to be more indicative of the capabilities of the modified Kaasschieter

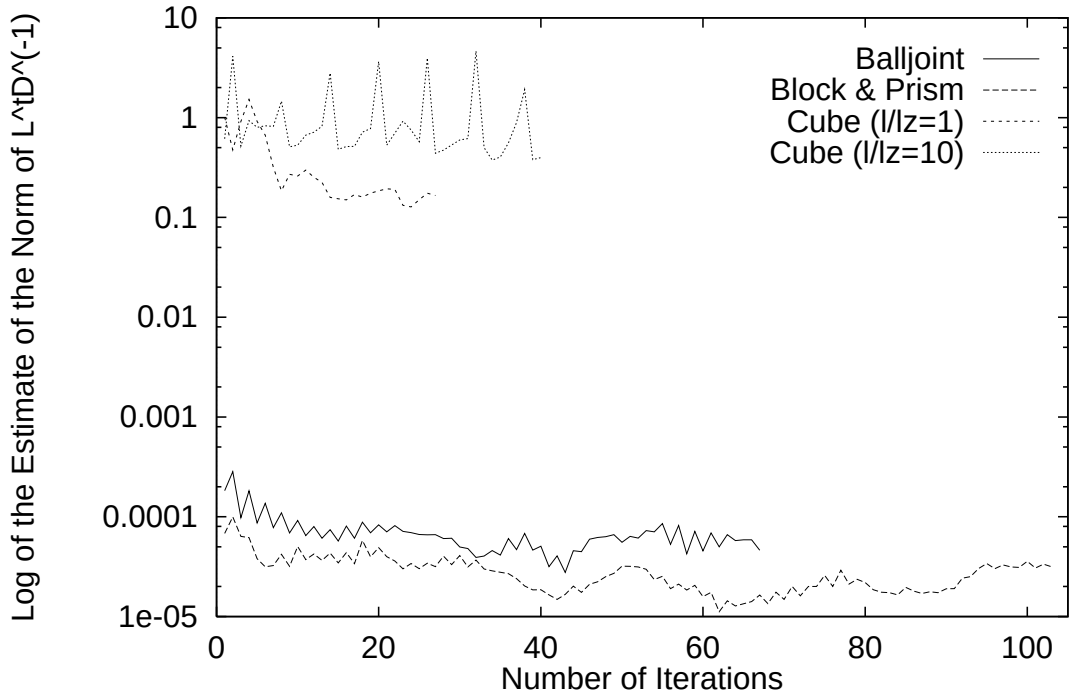


Figure 6.1: Estimate of $\| (\mathbf{L}^t\mathbf{D}^{-1})^{-1} \|$ vs. Number of Iterations

algorithm which were unfortunately not very promising (Table 6.1).

The first thing that can be seen is that the average aspect ratio of the mesh elements seems to have no correlation with the performance of the convergence criteria. There may be some correspondence but it is not obvious from this data. The individual aspect ratios of the elements does have an effect on the estimate of the minimum eigenvalue (μ_1) but this is mentioned later.

For almost all the test cases the stopping criteria terminated the iterations prematurely. The only exceptions are some of the coarser accuracy ($\varepsilon > 10^{-5}$) tests

for the more complex problems (the *Balljoint* and *Block & Prism*). This implies that there are inaccuracies in the inequality being used to determine the stopping point (inequality equation (6.15)). The three terms in the inequality equation which could have have errors are the estimate of $\| (\mathbf{L}^t\mathbf{D}^{-1})^{-1} \|$, the estimate of μ_1, and the calculated $\tilde{\mathbf{r}}_i$.

A plot of the all estimates made of $\| (\mathbf{L}^t\mathbf{D}^{-1})^{-1} \|$ is given in Figure 6.1 when the desired accuracy was specified to be $\varepsilon = 10^{-9}$. Remember that the terminating criteria only used the maximum estimate found up to and including the current iteration and not just the current estimate. The plot suggests that the estimate for the $\| (\mathbf{L}^t\mathbf{D}^{-1})^{-1} \|$ was fairly good as the magnitude of the estimate never significantly changed. Had the magnitudes varied more, it would have indicated that the estimate was poor, and probably untrustworthy.

It is worth noting that the estimate of the $\| (\mathbf{L}^t\mathbf{D}^{-1})^{-1} \|$ for the *Cube* problems is a little worse (more variable) than the estimate found for the more complex problems. Though this would effect the stopping criteria, it alone is not sufficient to explain the poor convergence criteria results found for the *Cube* test cases.

Another difficulty with the modified Kaasschieter algorithms is the poor initial estimate for the minimum eigenvalue which can be seen in Figure 6.2. This poor estimate explains the premature terminations of the two *Cube* test cases when a coarse accuracy was required ($\varepsilon > 10^{-4}$). This is to be expected due to the extremely small sub-space of the Krylov space being used to estimate $\tilde{\tilde{\mathbf{A}}}$'s minimum eigenvalue. This could be easily avoided by specifying a minimum number of iterative steps to be taken.

The initial guess of the solution vector will strongly influence how the estimates of μ_1 develop. For Figure 6.2, the initial guess was the zero vector except for where

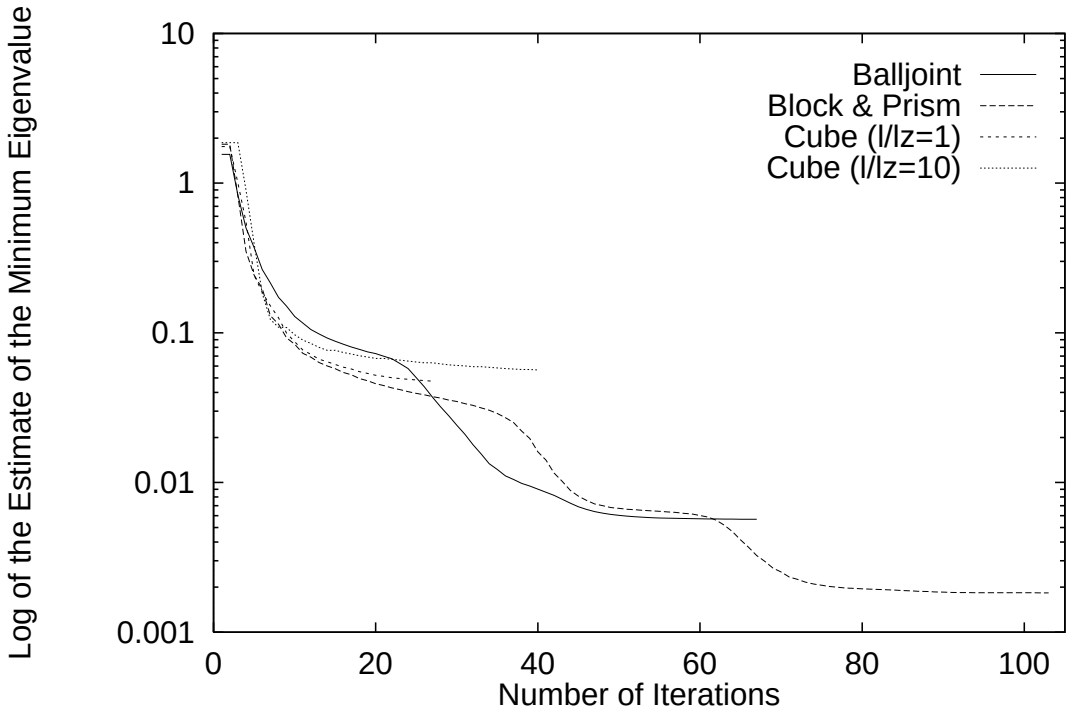


Figure 6.2: Estimate of μ_1 vs. Number of Iterations

a node had a specified displacement. A couple of tests were also made with an initial guess which was just the zero vector. This small change (less than 1% of the initial vector altered) resulted in a pronounced effect on the resulting μ_1 estimates.

A few tests have also shown that when the iterative algorithms are forced to continue beyond the believed points of convergence as reached in Figure 6.2 (and noted in Table 6.1), it is found that the μ_1 estimate does not change significantly anymore. This suggests that Kaasschieter's algorithm does actually converge on the minimum eigenvalues. However the convergence for the more complex problems is much slower than for the *Cube* problems. This is caused by the the meshes containing some elements with extremely poor aspect ratios resulting in smaller minimum eigenvalues.

Despite these observations, that the μ_1 estimate takes longer to converge for the more complex geometry problems and that the μ_1 estimate is sensitive to initial guess of the solution vector, no further conclusions could be made about the performance of the stopping criteria relative to the estimate of μ_1.

Finally, consider the calculated residual $\tilde{\mathbf{r}}_i$. This residual is calculated during each step of the iterative procedure by adding an adjustment vector to the $\tilde{\mathbf{r}}_i$ vector. Thus the residual vector will be highly susceptible to truncation of information as the residual and the adjustment vectors become smaller and smaller in magnitude. It has been noted before that the residual vector calculated in the iterative routines will become inaccurate after a number of iterations. A possible solution would be to re-calculate the true residual periodically during the iterative procedure.

Having an incorrect residual vector accounts for the slowly deteriorating performance of the stopping criteria for the more complex geometry problems as higher and higher accuracies are specified.

6.3.1 Conclusion

Unfortunately, the modified Kaasschieter's approach seems to only work consistently for the *Block & Prism* problem with a mid-range desired accuracy ($10^{-2} < \varepsilon < 10^{-5}$). Somewhat passable performance was also observed for the *Balljoint* problem with the same range of specified accuracies. However, for higher accuracies for the *Block & Prism* and the *Balljoint* problems and for any accuracy for the two *Cube* problems the stopping criteria proved itself to be premature.

The reasons for this seem to depend on the difficulty of finding good estimates of $\| (\mathbf{L}^t\mathbf{D}^{-1})^{-1} \|$ and μ_1, and maintaining an accurate value of $\tilde{\mathbf{r}}_i$. It possible to see specific regions where a poor estimate of one of these values clearly is the cause of the incorrect termination of the iterative procedures but in general it is more likely a combination of these factors which lead to the poor stopping criteria results.

I believed initially that Kaasschieter's method would be easy to modify and implement within my existing code. However, because of the difficulties in getting accurate values for the terms in the stopping criteria inequality, I have come to conclude that this method needs more modification or is simply inappropriate for the problems being tested.

Future research could be done on the importance of the initial guess on the convergence of the μ_1 estimate. A good initial guess for the solution vector will result in rapid convergence to the correct solution but this is not necessarily true for the convergence of the μ_1 estimates.

Other possible avenues of research which might prove fruitful are possible combinations of terminating criteria and the recalculation of the exact residual periodically. The residual used in the iterative CG method is a summation which can be quite sensitive to round-off and truncation errors common to ill-conditioned

matrices.

Finally, any future work should include a few experiments with some of the criteria developed by Ashby, Holst, Manteuffel and Saylor [4]. Their criteria are based on using dynamically calculated estimates of the condition number and inner products already calculated in the conjugate gradient process.

Chapter 7

Conclusions

7.1 Preconditioning

The performance of PCG iterative methods for linear elasticity is highly dependent on the average element aspect ratio. With an average aspect ratio better than 0.70, many preconditioning methods will be successful. However, as the average aspect ratio becomes smaller, the iterative methods encounter difficulty unless powerful preconditioning methods are used. The test problems used in this study had average tetrahedral aspect ratios varying between 0.78 and 0.02.

For unstructured grids with quadratic tetrahedral elements, a robust iterative method for a wide range of aspect ratios uses a hierarchical basis block preconditioner ($\mathbf{P_1}$), and a direct solve of the vertex block ($ILU_v(\infty, 0.0)$), coupled with a drop tolerance incomplete factorization ($ILU_m(\infty, 10^{-3})$) of the midpoint block. This method can be fine-tuned by using a coarser or finer drop tolerance for situations where the average aspect ratio is known/expected to be better or worse than 0.50 respectively. Level based ILU methods were not, in general, consistently

competitive with the drop tolerance ILU techniques (for the full range of the test problems).

With this type of preconditioning, the recommended iterative methods achieved at least comparable if not better solutions while outperforming the direct solution method by a factor of between three and seventeen times in terms of CPU cost. For the larger problems, the direct solutions could not be obtained due to the excessive memory requirements ($>$ 128 Meg) necessary to complete the direct solve. In those cases where direct solution method did succeed, the iterative and direct solutions for the displacements agreed to at least three Figures (a difference of 0.1% or smaller).

The hierarchical basis $\mathbf{P_1}$ block preconditioner with $ILU_v(\infty, 0.0)$ and $ILU_m(\infty, 10^{-3})$ generally outperformed a level based or drop tolerance based incomplete factorization preconditioning ($\mathbf{P_0}$) of the nodal basis stiffness matrix. There also did not appear to be any advantage to using the more accurate block preconditioner ($\mathbf{P_2}$) with the hierarchical basis.

Faster convergence rates can be achieved for meshes with average aspect ratios above 0.58 by using an incomplete factorization for both A_{vv} and A_{mm} regions. $ILU_v(\infty, 10^{-1})$ or $ILU_v(\infty, 10^{-2})$, and $ILU_m(\infty, 10^{-3})$ or $ILU_m(\infty, 10^{-4})$ seem to be the best combinations.

There are two standard methods for ensuring that the incomplete factorization of a symmetric positive definite matrix produces positive diagonal pivots. The *J & M Add* method [1] adds the dropped terms in the incomplete factorization to the diagonal. This method is guaranteed to produce positive pivots, but overestimates the amount necessary to add to the diagonal, and hence may produce a poor preconditioner. The alternative, is to keep adding a small relative amount to the diagonal until the incomplete factorization succeeds (see equation (3.3)) [39]. The

situation regarding these two methods is not altogether clear. For a hierarchical basis block preconditioner ($\mathbf{P_1}$), with a small drop tolerance for the ILU of the midpoint block (ILU_{mm}) the method of equation (3.3) was generally superior to the *J & Add* method. However, if a larger drop tolerance (or low level) is used for the ILU of the midpoint blocks, then there are some cases where the *J & M Add* method does succeed, while the technique of equation (3.3) fails due to an excessive number of attempts to incompletely factor the midpoint block (A_{mm}).

For fully three dimensional objects, available mesh generators appear to be capable of producing tetrahedral meshes with average element aspect ratios greater than 0.50, in which case we can expect good performance from a block hierarchical basis preconditioner compared to a direct solver. However, even for extreme element aspect ratios ($\simeq .02$), the block hierarchical basis preconditioner will also outperform a direct solver when a small drop tolerance is used for the midpoint preconditioner. Such extreme aspect ratios may arise in full three dimensional modelling of plates and shells constructed of composite materials.

7.2 Stopping Criteria

Unfortunately, the modified Kaasschieter's approach seemed to only work somewhat consistently for the *Block & Prism* problem and the *Balljoint* problem with desired accuracies in the range of $10^{-2} < \varepsilon < 10^{-5}$. Partially to blame for this is the modifications made to Kaasschieter's algorithms in order to apply it to preconditioned conjugate gradient (PCG) routines as compared to conjugate gradient (CG) routines. These modifications turned out to be difficult and not very accurate. Thus Kaasschieter's technique [36] is inappropriate for a general use PCG iterative solver without further developments.

The large effect the initial guess has on the convergence of the estimate for the smallest eigenvalue (μ_1) is also worth noting. Unfortunately, the best guess for the convergence of the PCG method may not be the best for the convergence of the μ_1 estimates.

Bibliography

[1] M. A. Ajiz and A. Jennings. A robust incomplete choleski conjugate gradient algorithm. *Int. J. Num. Meth. Eng.*, 20:949–966, 1984.

[2] F. Angeleri, V. Sonnad, and K. Bathe. Studies of finite element procedures - an evaluation of preconditioned iterative solvers. *Comp. Struct.*, 32:671–677, 1989.

[3] M. Ariola, I. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, 13(1):138–144, 1992.

[4] S. F. Ashby, M. J. Holst, T. A. Manteuffel, and P. E. Saylor. The role of the inner product in stopping criteria for congugate gradient iterations. Technical Report UCRL-JC-112586, LLNL, November 1992. Submitted to SIAM J. Mat. Anal. Appl.

[5] O. Axellson and P. S. Vassilevski. A survey of multilevel preconditioned iterative methods. *BIT*, 29:769–793, 1989.

[6] O. Axellson and P. S. Vassilevski. Algebraic multilevel preconditioning methods II. *SIAM J. Num. Anal.*, 27:1569–1590, 1990.

[7] O. Axelsson and I. Gustafsson. Preconditioning and two level multigrid methods of arbitrary degree of approximation. *Math. Comp.*, 40:219–242, 1983.

[8] O. Axelsson and P. S. Vassilevski. Algebraic multilevel preconditioning methods I. *Num. Math.*, 56:157–177, 1989.

[9] R. E. Bank, T. F. Dupont, and H. Yserentant. The hierarchical basis multigrid method. 1987. ZIB Technical report, Berlin.

[10] E. J. Barbero. A 3-d element for laminated composites with 2-d kinematic constraints. *Comp. Struct.*, 45:263–271, 1992.

[11] E. Barragy and G. F. Carey. Preconditioners for high degree elements. *Comp. Meth. Appl. Mech. Eng.*, 93:97–110, 1991.

[12] A. Behie and P. A. Forsyth. Comparison of fast iterative methods for symmetric problems. *IMA J. Num. Anal.*, pages 41–63, 1983.

[13] A. Behie and P. A. Forsyth. Incomplete factorization methods for fully implicit simulation of enhanced oil recovery. *SIAM J. Sci. Stat. Comp.*, 5:543–561, 1984.

[14] J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comp.*, 55:1–22, 1990.

[15] B. Brusson and V. Sonnad. A comparison of direct and iterative methods for sparse unsymmetric systems of linear equations. *Int. J. Num. Meth. Eng.*, 28:801–815, 1989.

[16] V. E. Bulgakov and M. Y. Belyi. On the multi-grid technique for solving three dimensional boundary value engineering problems. *Int. J. Num. Meth. Eng.*, 33:753–764, 1992.

[17] G. F. Carey and E. Barragy. Basis function selection and preconditioning high degree finite element and spectral methods. *BIT*, 29:794–804, 1989.

[18] P. Chin, E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Preconditioned conjugate gradient methods for the incompressible Navier Stokes equations. *Int. J. Num. Meth. Fluids*, 15:273–295, 1992.

[19] B. Chouchaoui. Evaluating the remaining strength of corroded pipes. Ph.D. Thesis, Mechanical Engineering, University of Waterloo. September 1992.

[20] S. D. Connell and D. G. Holmes. A 3d unstructured adaptive multigrid scheme for the euler equations. AIAA 93-3339.

[21] Structural Dynamics Research Corporation. Ideas VI Solid Modeling & Ideas VI Finite Element Modeling Packages, (1992).

[22] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Drop tolerance preconditioning for incompressible viscous flow. *Int. J. Comp. Math.*, 44:301–312, 1992.

[23] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM J. Matrix Anal. Applic.*, 13:944–961, 1992.

[24] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Towards a cost effective ILU preconditioner with high level fill. *BIT*, 32:442–463, 1992.

[25] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices.* Oxford, 1986.

[26] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989.

[27] S. C. Eisenstat, H. C.Elman, and M. H. Schultz. Block preconditioned conjugate gradient like methods for numerical reservoir simulation. *SPE J. Res. Eng.*, 3:307–312, 1988.

[28] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale sparse matrix package I: the symmetric codes. *Int. J. Num. Meth. Eng.*, 18:1145–1151, 1982.

[29] C. Farhat and N. Sobh. A coarse/fine preconditioner for very ill conditioned finite elemnt problems. *Int. J. Num. Meth. Eng.*, 28:1715–1723, 1989.

[30] S. Foresti, G. Brussino, S. Hassanzadeh, and V. Sonnad. Multilevel solution method for the p-version of finite elements. *Comp. Phys. Comm.*, 53:349–355, 1989.

[31] S. Foresti, S. Hassanzadeh, H. Murakami, and V. Sonnad. A comparison of preconditioned iterative methods using rapid operator application against direct solution methods. *Int. J. Num. Meth. Eng.*, 32:1137–1144, 1991.

[32] O. P. Iliev, M. M. Makarov, and P. S. Vassilevski. Performance of certain iterative methods in solving implicit difference schemes for the 2-d Navier-Stokes equations. *Int. J. Num. Meth. Eng.*, 33:1465–1479, 1992.

[33] A. Jennings and G. M. Malik. Partial elimination. *J. Inst. Maths. Appl.*, 20:307–316, 1977.

[34] Barry Joe. Tetrahedral mesh generation. University of Waterloo Scientific Computing Seminar, 1992.

[35] M. Jung, U. Langer, and U.Semmler. Two level hierarchically preconditioned conjugate gradient methods for solving linear elasticity finite element equations. *BIT*, 29:748–768, 1989.

[36] E. F. Kaasschieter. A practical termination criterion for the conjugate gradient method. *BIT*, 28:308–322, 1988.

[37] D. Kershaw. The incomplete choleski conjugate gradient method for the solution of systems of linear equations. *J. Comput. Phys.*, 26:43–65, 1978.

[38] J. Mandel. Two level domain decomposition preconditioning for the p-version finite element method in three dimensions. *Int. J. Num. Meth. Eng.*, 29:1095–1108, 1990.

[39] T. Manteuffel. An incomplete factorization technique for positive definite systems. *Math. Comp.*, 34:473–497, 1980.

[40] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients. *ACM Trans. Math. Soft.*, 6:206–219, 1980.

[41] H. Ohtsubo and M. Kitamura. Element by element a posteriori error estimation of the finite element analysis for three-dimensional elastic problems. *Int. J. Num. Meth. Eng.*, 33:1755–1769, 1992.

[42] E. L. Poole, N. F. Knight, and D. Dale. High performance equation solvers and their impact on finite element analysis. *Int. J. Num. Meth. Eng.*, 33:855–868, 1992.

[43] M. P. Robichaud and P. A. Tanguy. Finite element solution of three dimensional incompressible flow problems by a preconditioned conjugate residual method. *Int. J. Num. Meth. Eng.*, 24:447–457, 1987.

[44] L. H. Tan and K. Bathe. Studies of finite element procedures - the conjugate gradient and gmres methods in ADINA and ADINA-F. *Comp. Struct.*, 40:441–449, 1991.

[45] P. S. Vassilevski. Hybrid V-cycle algebraic multilevel preconditioners. *Math. Comp.*, 58:489–512, 1992.

[46] N. Wiberg and P. Moller. Formulation and solution of hierarchical finite element equations. *Int. J. Num. Meth. Eng.*, 26:1213–1233, 1988.

[47] O. C. Zienkiewicz, J. P. Gago, and D. W. Kelly. The hierarchical basis concept in finite element analysis. *Comp. Struct.*, 16:53–65, 1983.