INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Adaptive refinement and related topics for finite element analysis of linear elliptic problems

Kyriacou, Stelios Kyprou, M.S.
Rice University, 1991

Order Number 1345320
RICE UNIVERSITY

Adaptive Refinement and Related Topics for Finite Element Analysis of Linear Elliptic Problems

by

Stelios Kyriacou

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
Master of Science

APPROVED, THESIS COMMITTEE:

Dr. Akin, John E., Chairman
Professor
Mechanical Engineering and Materials Science

Dr. Warren, Joe David
Assistant Professor
Computer Science

Dr. Meade, Andrew J.
Assistant Professor
Mechanical Engineering and Materials Science

Houston, Texas
April, 1991
Adaptive Refinement and Related Topics for Finite Element Analysis of Linear Elliptic Problems

Stelios Kyriacou

Abstract

An overview of the methods for finite element adaptive refinement and error estimation for linear elliptic problems is performed. An error estimator, based on nodal averaging of a gradient measure, suitable for linear, steady state, 2D and 3D stress and thermal analysis, is created and interfaced with the FE program FINITE/GP. An adaptive h-refinement method incorporating the above error estimator is implemented for two sample problems and results are presented which indicate that the method works. Conclusions are drawn concerning the usefulness of the method. A separate chapter gives an overview as well as experience gained with solvers for sparse matrices.
Acknowledgments

I would like to express my thanks to my advisor Dr. John Ed Akin who's guidance has always been so helpful, and who has been very encouraging and understanding to my research. I would also like to thank Dr. Joe D. Warren for his help and advice as well as Dr. Andrew J. Meade for serving on my thesis committee. Special thanks go to Dr. Doug Moore for providing valuable advice and support.

I would also like to thank my sponsors, the National Science Foundation, for their support of my Master's Degree as well as the Agency for International Development (AID) who's generous aid has brought me to this position.

Thanks go to Rita Alevriadou, Stuart Strong, Greg Sievert, and Dan Badders for useful comments and proofreading of sections of this thesis.

Lastly I would like to thank my family, my friends and many others who's help has been so supportive.
Table of Contents

Abstract ii
Acknowledgments iii
List of Illustrations viii
List of Tables x
Nomenclature xi

1 Introduction 1
1.1 The Finite Element Method in Engineering and Science 1
1.2 Error Estimates, and Adaptive Refinement 2
1.3 Chapters Arrangement 2

2 Adaptive Mesh Refinement and Error Estimates 4
2.1 Introduction 4
2.1.1 Types of Norms Used in Error Estimates for FEM 5
2.2 Adaptive Mesh Refinement Methods 6
2.2.1 H-refinement 6
2.2.2 P-refinement 6
2.2.3 R-refinement 8
2.3 Error Estimators 8
2.4 Hierarchical P-refinement 9
2.4.1 The Hierarchical Concept 9
2.4.2 Advantages of Hierarchical P-refinement 9
2.4.3 Hierarchical Error Estimator 14
2.4.4 Adaptive Subdivision Procedure 14
2.5 Discussion 14

3 Development of an H-refinement Error Estimator 18
3.1 Introduction 18
3.2 Derivation of Error Measures 18
5 Solvers for Sparse Symmetric Positive Definite Matrices  49
  5.1 Introduction ................................................. 49
  5.2 Hardware and Software used ................................. 50
  5.3 Ordering of the Matrix ...................................... 51
     5.3.1 Fill-in .................................................. 51
     5.3.2 Ordering for Reducing Fill-in .......................... 51
  5.4 Direct Sparse Solvers and Ordering Algorithms .......... 53
  5.5 Implementation and Testing of Solvers .................... 56
     5.5.1 Trying the Bandwidth Solver ............................ 56
     5.5.2 Implementing the Yale Sparse Solver ................... 58
     5.5.3 Implementing the Skyline Solver ....................... 59
  5.6 Discussion .................................................. 60
     5.6.1 The Bone Problem Results .............................. 61

Bibliography .................................................. 62

A Patch Test for the Error Estimator  65
  A.1 Input File .................................................. 65
  A.2 Parts of Output File ....................................... 66

B Program “ref.c” ............................................. 68

C Sample Input Files to program “ref”  80
  C.1 Sample Input File “meshin1” .............................. 80
  C.2 Sample Input File “error-file1” ........................... 80

D Sample Output Files from program “ref”  81
  D.1 Sample Output File “meshin2” ............................. 81
  D.2 Sample Output File “error-file2” .......................... 81
  D.3 Sample Output File “babu2.dat” ............................ 81

E Subroutine “error.f” .............................. 84

F Subroutine “movie.f” .............................. 95
G  Subroutine “caeds.f”  

H  Subroutines “preasb.f”, “intbin.f”, and “int_sort.f”  

I  Subroutine “newdat.f”  

J  Subroutines “shp16q.f” and “der16q.f”  

K  Subroutine “eltype.f”
Illustrations

2.1 Adaptive refinement methods ........................................ 7
2.2 Hierarchical Elements ................................................. 10
2.3 Better conditioning through the use of hierarchical elements ........ 12
2.4 Convergence Rates for various refinement methods .................... 17

3.1 Exact (1), discontinuous (2), and continuous (3) stress functions for a one-dimensional problem for a linear (above) and a quadratic (below) approximation of the displacement $u$. ........................................ 19

4.1 Babuska Problem Definition: side OA has temperature $T = 0$, sides AB, BD, DE, and EO have prescribed fluxes .................. 31
4.2 L-shaped Problem Definition ........................................... 31
4.3 Element Boundary Continuity ......................................... 33
4.4 Definitions associated with the refinement process ....................... 34
4.5 Plot of percentage error vs number of nodes in a log-log scale for the adaptive and uniform refinement of the Babuska problem .......... 41
4.6 Plot of percentage error vs number of nodes in a log-log scale for the adaptive and uniform refinement of the L-shaped problem ........ 42
4.7 Successive stages of adaptive refinement for the L-shaped problem, with error contours drawn ........................................ 43
4.8 Successive stages of uniform refinement for the L-shaped problem, with error contours drawn ........................................ 44

5.1 Effect of various kinds of ordering on fill-in (figure obtained from George and Liu [6]) ................................................. 52
5.2 Bandwidth Storage: Drawing on the left depicts the outer non-zeros in the symmetric matrix and on the right shows the part of the matrix stored in the computer memory ........................................ 54
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>A numbering system for reducing the bandwidth</td>
<td>54</td>
</tr>
<tr>
<td>5.4</td>
<td>Skyline Storage (as used in FINITE/GP)</td>
<td>55</td>
</tr>
<tr>
<td>5.5</td>
<td>The bone problem geometry</td>
<td>57</td>
</tr>
</tbody>
</table>
Tables

3.1 Zienkiewicz correction factors ........................................... 22

5.1 Comparison of the three solvers shows the skyline solver as the best performer ......................................................... 60
Nomenclature

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{H} ) or ( \mathbf{N} )</td>
<td>Element Shape Function Vector</td>
</tr>
<tr>
<td>( \mathbf{B} )</td>
<td>Matrix with the gradients of the element shape functions.</td>
</tr>
<tr>
<td>( \mathbf{D} )</td>
<td>Material information matrix</td>
</tr>
<tr>
<td>( \mathbf{K} )</td>
<td>Stiffness matrix</td>
</tr>
<tr>
<td>( u )</td>
<td>The unknown function</td>
</tr>
<tr>
<td>( f ) or ( \mathbf{F} )</td>
<td>The forcing function</td>
</tr>
<tr>
<td>( e )</td>
<td>Error</td>
</tr>
<tr>
<td>( N ) or ( n )</td>
<td>Number of unknowns (in the FORTRAN subroutines it denotes the number of nodes per element)</td>
</tr>
<tr>
<td>( \mathbf{n} )</td>
<td>Normal unit vector</td>
</tr>
<tr>
<td>( \eta )</td>
<td>Relative error indicator</td>
</tr>
<tr>
<td>( r )</td>
<td>Residual (also represents the radius in polar coordinates)</td>
</tr>
<tr>
<td>( \varepsilon )</td>
<td>Relative error estimator or strain function</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>Stress (or even gradient in a few cases)</td>
</tr>
<tr>
<td>( \bar{\sigma}^* )</td>
<td>Nodal averages of the gradient (evaluated at each node)</td>
</tr>
<tr>
<td>( \sigma^* )</td>
<td>Function based on a FEM approximation of the nodal average values ( \bar{\sigma}^* )</td>
</tr>
<tr>
<td>( q )</td>
<td>Quadrature point</td>
</tr>
<tr>
<td>( nq )</td>
<td>Number of quadrature points</td>
</tr>
<tr>
<td>( ns )</td>
<td>Number of strains per node</td>
</tr>
<tr>
<td>( \mathcal{L} )</td>
<td>Linear differential operator</td>
</tr>
<tr>
<td>( T )</td>
<td>Temperature</td>
</tr>
<tr>
<td>( t )</td>
<td>Gradient (or flux)</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>Finite Element Domain</td>
</tr>
<tr>
<td>( \partial^0 \Omega )</td>
<td>The boundary of the domain with essential boundary conditions imposed</td>
</tr>
<tr>
<td>( \partial^1 \Omega )</td>
<td>The boundary of the domain with natural</td>
</tr>
</tbody>
</table>
boundary conditions imposed

\( r, \theta \)  
Polar coordinates

\( n_{space} \)  
Dimension of the space

\( p \)  
Polynomial degree of the element

\( \lambda \)  
A measure of the smoothness of the exact solution

\( C \)  
Constant

\( k \)  
Constant

\( \beta \)  
Constant

\( \gamma \)  
Constant

\( \delta \)  
Constant

\( A \)  
Area

\( n_q \)  
Number of integration points

\( n_s \)  
Number of stress components at each point

\( W_q \)  
Quadrature weight

\( J \)  
Jacobian function

\( \text{dof} \)  
Degree(s) of freedom

\( \text{FE} \)  
Finite Element

\( \text{FEM} \)  
Finite Element Method

\( \text{FEA} \)  
Finite Element Analysis

\( \text{RMS} \)  
Root mean square

\( \text{MW}(s) \)  
Mega Word(s) (each word represents a number in the computer memory)

\( \text{MB}(s) \)  
Mega Byte(s) (each byte represents eight bits)

\( 1D, 2D, 3D \)  
One-, two-, and three-dimensional respectively

**Mathematical Notation**

\([\bullet]\)  
Square brackets denote a matrix

\( \textbf{A} \)  
Bold font also denotes a matrix

\( \textbf{A}^T \)  
Superscript "\( T \)" denotes the transpose of \( \textbf{A} \)

\( \textbf{A}^{-1} \)  
Superscript "\( -1 \)" denotes the inverse of \( \textbf{A} \)

\( \{\bullet\} \)  
Curly brackets denote a vector

\( A_{ij} \)  
Subscripted lower case letters represent components of a tensor

\( e_u \)  
Subscript denotes the quantity whose error \( e \) is evaluated for
\[ \| \cdot \|_E \quad \text{Double vertical lines denote a norm} \]
\[ \text{Subscript denotes the type of norm} \]
\[ (E \text{ stands for the energy norm}) \]
\[ \bar{u} \quad \text{Bar on top means the nodal vector for } u, \text{ as found by FEM} \]
\[ \hat{u} \quad \text{Hat on top gives the FE approximation of function } u \]
Chapter 1

Introduction

1.1 The Finite Element Method in Engineering and Science

Most of the today’s engineering analysis deals with problems involving differential equations too difficult to solve analytically. Currently the most widely used method of solving these problems, especially the ones with irregular geometries and complex boundary conditions, is the Finite Element Method (FEM), which over time has become an indispensable tool for today’s engineer. The following types of problems have been tackled usually quite effectively by commercial FE codes:

1. Steady state thermal and stress analysis.
2. Incompressible fluid flow.
3. Electromagnetic analysis.
4. Seepage through porous media
5. Lubrication and Real flow analysis
6. Non-linear elastic analysis and Plasticity
7. Transient thermal and stress analysis.
8. Frequency, and modal analysis.

This report deals with error estimators and methods for adaptive refinement for steady state thermal and stress problems as well as for other elliptic problems. These are the simplest to handle problems. The more complicated non-linear and transient problems will hopefully be dealt with in future research.
1.2 Error Estimates, and Adaptive Refinement

Two important problems in FEM are:

- It is very unfortunate that although FEM is such a powerful method it may give inaccurate results if not used by a skilled and careful analyst. A very important source of error is the use of a finite number of elements whereas an infinite number of elements would be required for the FEM solution to converge to the true solution. This error is usually referred to as the discretization error.

- Even skilled analysts are faced with the problem of preparing better meshes in case the old ones do not work satisfactorily, a process which consumes a lot of hours. This process is very wasteful for both human and computer resources.

One way to handle these problems is through error estimates and adaptive refinement methods. These two concepts are explained better in the following paragraphs.

The goal of adaptive refinement is to have the computer (instead of the analyst) judge and refine the solution with the least possible human intervention and the least use of computer resources. This can be achieved partly with the use of error estimation techniques based on mathematical and empirical results. Error estimators can be of two forms: i) local (element-wise) error indicators and ii) a global error estimator. The global error estimator is usually an average of the local indicators and it is used as the “judge” of whether the solution is acceptable as far as discretization error is concerned. On the other hand the error indicators are used only if the solution is judged to be unacceptable, to guide the adaptive refinement process.

The adaptive refinement, in contrast to uniform refinement which subdivides uniformly the whole domain, uses these error indicators to refine only the elements with the highest error. It can do the refinement either by reducing the size of these elements or by refining the polynomials (the shape functions) used for approximating the solution (i.e. increasing their order).

1.3 Chapters Arrangement

The contents is arranged as follows: In the second chapter a review of the following is done: 1) two major techniques on error estimation and 2) two popular methods of adaptive refinement. In the third chapter details are given of an implementation of an error estimator suitable for linear, elliptic problems, such as plane stress and
strain, axisymmetric, and general 3D stress and thermal analysis. In the fourth chapter an experiment in h-adaptive refinement (i.e. one based on decreasing the size of elements) is outlined through the use of the error estimator described in chapter three. Chapter five is minimally connected with the rest of the chapters. In this chapter a comparison of various types of linear systems of equations solvers as well as different computer architectures is made. Solvers and computer architecture are very important aspects of FE adaptive refinement since the solution of the system of linear equations is a major expense.
Chapter 2

Adaptive Mesh Refinement and Error Estimates

2.1 Introduction

Reasons for Adaptive Refinement  As explained in the introductory chapter, adaptive refinement has become a vital part of FE analysis. If successfully implemented it can provide some important advantages like:

- More reliable answers.
- Less experience required.
- Facilitated analysis and higher automation.

Adaptive Refinement in Commercial Software  Many FE software packages have realized the possibilities of adaptive refinement and have begun including error estimators and adaptive mesh refinement capabilities in their codes. Betts [9] gives an up-to-date account of the adaptive capabilities of the major FEM commercial codes. Major examples are:

- I-DEAS from SDRC (currently in owlnet – at Rice University–under the IBM name CAEDS).
- I/FEM from integral.
- Ansys from Swanson Analysis Systems.
- MSC/Probe from MacNeal-Schwendler Corporation.

Aspects of Adaptive Refinement  There are four main aspects in adaptive refinement:

1. The error estimator is the criterion for deciding whether refinement is needed or not.
2. Error indicators are the criteria for deciding which elements to refine.

3. Method to be used for refinement: Here the choice is between using h-, p- or even r-refinement methods or a combination of them.

4. Expense of the adaptive refinement: This project will not be concerned with this aspect. It should be part of future research.

2.1.1 Types of Norms Used in Error Estimates for FEM

Pointwise definitions of errors are not very useful because usually an elementwise error or a total (over whole domain) error is required. For this reason various norms are used for the error estimation. The ones most often used are:

1. The energy norm:
   \[ \|e_u\|_E = \sqrt{\int_\Omega e_u \cdot e_u d\Omega} \]  \hfill (2.1)

2. The L2 stress norm:
   \[ \|e_\sigma\|_{L^2} = \sqrt{\int_\Omega e_\sigma \cdot e_\sigma d\Omega} \]  \hfill (2.2)

3. The H1 norm:
   \[ \|e_u\|_{H^1} = \sqrt{\int_\Omega (e_u \cdot e_u + e_u \cdot e_u) d\Omega} \]  \hfill (2.3)

Which one to use depends on the kind of problem being attacked. This is not a trivial case since there are advantages and disadvantages in the use of each one. For instance a singularity may produce large error estimates not only in the area around it but also in distant areas [2, page 120]. If \( L_2 \) displacement (not stress) error estimates are used they tend to propagate the singularity error far away. This means that the mesh will be refined unnecessarily in many places. On the contrary \( H_1 \) as well as energy error estimates keep large errors in the area near the singularity.

Currently the most commonly used error estimate norm is the energy norm since it has a special significance in elastic problems which are the main use of FEA. For reasons of convenience in the write-up of the error estimator for this thesis an \( L_2 \) stress norm will be used.
2.2 Adaptive Mesh Refinement Methods

The methods currently used in adaptive mesh refinement are:

2. P-refinement Method.
3. R-refinement Method.
4. Combined Methods e.g. h-p, r-h, r-p, etc.

Figure 2.1 shows examples of the various methods of adaptive refinement for a very simple problem. The methods are explained in more detail in the following subsections.

2.2.1 H-refinement

This is the conventional method for mesh refinement in which the mesh size is reduced to obtain a more accurate solution. The original mesh, shown in figure 2.1(a) is composed of four quadrilateral elements while figure 2.1(b) shows the h-refined case in which the errored element i.e. the one with the high or unacceptable error is subdivided into four new elements. This is the standard (in contrast to the buffer) method of subdivision, both of whom are explained in the chapter 4. The name of the h-method comes from the use of “h” in the FE literature to denote the size of the mesh. Algor, Ansys, and I/FEM [9] are among the leading FE software employing the h-refinement method.

2.2.2 P-refinement

This method uses higher order functions to achieve a reduction of the error in the “errored” element. The name p-refinement comes from the symbol “p” used by FE analysts to denote the order of the approximating shape functions. Applied Structure and MSC/Probe [9] are currently using the p-refinement method. Currently most p-refinement methods use hierarchical (a concept explained in section 2.4) shape functions to achieve a reduction of the error in the “errored” element as shown in figure 2.1(c). These shape functions seem very promising and a whole section 2.4 is devoted to them.
(a): Original Mesh

(b): Final Mesh, standard h-refinement, the errored element is subdivided into smaller elements.

(c): Final Mesh, p-refinement, the lines drawn correspond to a unit increase of the polynomial order.

(d): Final Mesh, r-refinement, the sides of the elements behave as springs with stiffnesses proportional to the error in the elements defined by them.

Figure 2.1 Adaptive refinement methods
2.2.3 R-refinement

The way r-refinement is done is by moving the nodes towards the areas of high error as shown in figure 2.1(d). In this way the size of the errored elements decreases while at the same time the low error elements increase in size.

The movement of the nodes towards the high error areas is analogous to the following phenomenon in elasticity. Let the nodes become masses and let the sides of the elements become springs with spring constants set so that in equilibrium to provide the original geometry. Then the stiffness in each spring (side) is increased or decreased according to the value of the error of the FE solution in the neighboring elements. Due to this change, the masses (nodes) will move to a new equilibrium configuration. This is the new geometry according to the r-refinement method.

This last refinement method is not important by itself. It is often used in combination with either of the previous two methods. An example is the FEA software package CAEDS [9] that uses a combination of h- and r-adaptive refinement methods.

2.3 Error Estimators

In FEM there are two kinds of error estimates: (a) a priori estimates and (b) a posteriori estimates. A priori estimates according to Noor et al [23] are based on a knowledge of the characteristics of the solution such as smoothness, and provide qualitative information about the asymptotic rates of convergence as the number of elements goes to infinity. They provide a worst case estimate for a particular class of problems. On the contrary the a posteriori estimates which are the main subject of current research as well as of this thesis, are based on information derived during the solution process and provide an approximation of the actual error in the solution.

Two kinds of a posteriori error estimators will be examined:

a) The first error estimator is based on an averaging done on the gradient of the FE solution. It will be discussed in detail, and implemented in chapter 3.

b) The second one is closely related to the hierarchical p-refinement and will be discussed in section 2.4.
2.4 Hierarchical P-refinement

2.4.1 The Hierarchical Concept

The hierarchical shape functions were introduced by Zienkiewicz and co-workers in 1971 and used in an a-priori chosen manner (i.e. in a non-adaptive scheme). Later their basis was expanded by the use of new families of hierarchical elements introduced by Peano and co-workers. Today there seems to be a trend towards using these elements when adaptive refinement is required.

The basic idea is based on the classical Ritz approach in which a simple geometry can be solved utilizing a trigonometric Fourier series expansion. In this kind of expansion original approximating functions can be enriched by the introduction of new higher frequency trigonometric functions to get a better approximation. The important point to notice is that this process takes place without changing the lower frequency functions. This concept is called hierarchical.

The hierarchical shape functions can be used in an h-refinement context but they seem to be much more useful in a p-refinement context [12]. In classical adaptive p-refinement the increase of the order of the refinement meant always removing the original shape functions and introducing new ones of higher order. This is avoided in the hierarchical adaptive p-refinement; higher order shape functions are simply added to the original shape functions. The result is a mixture of old (low order) and new (higher order) shape functions coexisting in the same model. In this way, the original stiffness matrix and the original solution can still be utilized.

The hierarchical concept is shown in the figure 2.2 taken from reference [18]. The right-hand-side drawings show the hierarchical element and three hierarchical shape functions $N_i^h$ (where $h$ stands for hierarchical and $i$ denotes the node number) while the drawings on the left show the corresponding standard element and shape functions $N_i^s$ (where $s$ stands for standard). Notice the first two shape functions: $N_i^s$ on the left is a quadratic function while $N_i^h$ on the right is a linear function since it is the original shape function.

2.4.2 Advantages of Hierarchical P-refinement

As mentioned earlier, in contrast to the original purpose of the hierarchical shape functions for being used as normal a-priori chosen elements they were found to be much more useful in adaptive refinement. According to Zienkiewicz et al. [18] the
(a) Standard   (b) Hierarchical

Fig. 7.25 Standard and hierarchical shape functions corresponding to a Lagrangian, quadratic element

Figure 2.2 Hierarchical Elements
reasons for this are the following advantages of the hierarchical adaptive p-refinement over classical adaptive p-refinement or h-refinement:

- Improved conditioning of the stiffness matrix.
- Utilizing the initial solution when solving for the refined mesh, and permitting a simple iteration to find the new solution.
- Ease of introducing error indicators.

The above advantages are explained in the following subsections.

**Improved Conditioning of the Stiffness Matrix**

According to Zienkiewicz et al [18] the fact that the new hierarchical shape functions act as perturbations to the original solution, rather than substitutes, produces a more diagonal form of the stiffness matrix $K$ which usually ensures its better conditioning as well as a faster rate of iteration convergence. An example of the above assertion is shown graphically in fig. 2.3 taken from the same source. In the figure a single cubic element example is shown in the upper two drawings. The normal shape function is shown in the first drawing and the hierarchical shape function in the second. The condition numbers for the stiffness matrix are shown for both cases and clearly the condition number of the hierarchical element is an order of magnitude better than the standard element's. A similar example with four cubic elements instead of one, is shown in the lower two drawings. Again there is an order of magnitude of improvement in the condition number of the hierarchical shape functions.

**Initial Solution is Utilized in Hierarchical P-Refinement**

One of the advantages of the hierarchical concept in adaptive refinement stated at the beginning of the section is that the previous solution can be utilized. This is achieved as follows:

Assume that a numerical solution $u_1^{(n)}$ (subscript 1 means initial refinement level) to the problem has already been obtained. The original FE equation (assuming $n$ unknowns) would be:

$$K_{(n)}u_1^{(n)} = f^{(n)}$$
Figure 2.3 Better conditioning through the use of hierarchical elements
If \( m \) more shape functions are added hierarchically, the stiffness matrix can be divided into four parts as:

\[
\begin{bmatrix}
K_{(n)} & K_{(n,m)} \\
K_{(m,n)} & K_{(m)}
\end{bmatrix}
\begin{cases}
\{u_2^{(n)}\} \\
\{u^{(m)}\}
\end{cases} = \begin{cases}
f^{(n)} \\
f^{(m)}
\end{cases}
\] (2.4)

Since the new functions are hierarchical, they will not affect the old ones. So \( K_{(n)} \) will be the same.

So the initial solution \( u_1^{(n)} \) could be used (if needed) as an initial guess for \( u_2^{(n)} \) iterative solution with the initial guess for \( u^{(m)} \) obtained through the use of the new additional equations (the lower \( m \) equations). Thus the initial guess \( u^{(m)} \) will be:

\[
u^{(m)} \approx K^{-1}_{(m)} (f^{(m)} - K_{(m,n)} u_1^{(n)})
\]

So it can be seen that the previous solution \( u_1^{(n)} \) is utilized in hierarchical refinement.

**Ease of Introducing Error Measures**

Hierarchical shape functions have the advantage of providing an immediate error estimate. Since the solution error now is:

\[
e = \hat{u}^{(n+m)} - \hat{u}^{(n)}
\]

With

\[
\hat{u}^{(n)} = N^{(n)} u_1^{(n)} \quad (2.5)
\]

\[
\hat{u}^{(n+m)} = N^{(n)} u_2^{(n)} + N^{(m)} u^{(m)} \quad (2.6)
\]

An approximation for the error is:

\[
e \approx N^m u^{(m)} \approx N^m K^{-1}_{(m)} (f^{(m)} - K_{(m,n)} u_1^{(n)})
\]

In hierarchical p-refinement an error indicator can be defined as the error that will be corrected if a new hierarchical dof (degree of freedom) is introduced. The formula for the \( n+1 \) dof error indicator was found to be [18]:

\[
\eta_{n+1}^2 = \frac{\left( \int_{\Omega} N_{n+1} r \, d\Omega \right)^2}{K_{(n+1,n+1)}} \quad (2.7)
\]

Where: \( r \) is the residual i.e. \( r = -Le \), \( e = u - \hat{u} \approx N_{n+1} u_{n+1} \)
2.4.3 Hierarchical Error Estimator

In the previous paragraph an approximate error indicator (eq. 2.7) was found through the introduction of a hierarchical shape function \( N_{n+1} \). The next step is to find a global error estimator. To take care of the fact that the next shape function might be orthogonal to the residual which means that the error indicator will be zero the following inequality is used:

\[
\left( \int_{\Omega} N_{n+1} r \, d\Omega \right)^2 \leq \left( \int_{\Omega} N_{n+1}^2 \, d\Omega \right) \left( \int_{\Omega} r^2 \, d\Omega \right)
\]  
(2.8)

So according to [18] a good error estimator although not rigorously proved is based on the following norm which is a combination of equations 2.7 and 2.8:

\[
\varepsilon_{n+1}^2 \approx \frac{\left( \int_{\Omega} N_{n+1}^2 \, d\Omega \right) \left( \int_{\Omega} r^2 \, d\Omega \right)}{K_{(n+1,n+1)}}
\]  
(2.9)

For each degree of freedom at the next refinement level an error norm can be evaluated with the equation above. The sum of all these norms \( \varepsilon_{n+1} \) is the error estimator.

According to [18] the evaluation of the residuals is not a straight-forward calculation since usually the shape functions have a lower continuity than the differential operator \( L \). Also care has to be taken at the element boundaries where discontinuities exist.

2.4.4 Adaptive Subdivision Procedure

The adaptive process starts with an initial mesh created by a mesh generator probably to fit the boundaries or maybe laid out by the analyst to fit the problem. Then after the first solution is obtained the error estimator should be used to show if there is a need for further refinement. If there is such a need, refinement is done according to the error indicators.

2.5 Discussion

Automation of Refinement The final aim of adaptive refinement seems to be the automation of the whole process. Some commercial software according to Betts [9] have already created a prototype automated adaptive refinement process. Nevertheless according to the same author critics have commented that this may not be desirable since in its current state the Finite Element Analysis can not predict potential problems, e.g. development of cracks etc. So according to these critics the need for an experienced analyst still exists.
Hierarchical Elements Seem to be the Winners  From the preceding sections it can be seen that hierarchical elements and p-refinement are definitely becoming very competitive in the field of error estimation and adaptive refinement. The advantages of the hierarchical p-refinement have been mentioned in section 2.4: easily calculated error estimates, better conditioning, and ability to use the previous solution as well as a simple iterative solution. A disadvantage of hierarchical elements over normal elements is the fact that there is no direct significance of the unknowns. In the direct method, unknowns represent the displacement etc. at the nodal points. In hierarchical elements this correspondence is lost. Fortunately it is easy to use the unknowns in the equation 2.5 to find the displacement at the nodes.

First Steps Toward P-refinement  To help future work the first step towards p-refinement was done by the creation of five new 2-D element shape function subroutines with \( p = 3 \) and \( p = 4 \). In more detail, subroutines were created to handle 16-noded, 17-noded, 9-noded, and 12-noded quadrilateral elements as well as 15-noded triangular elements. These subroutines were patch-tested with FINITE/GP and performed satisfactorily (except the 15-noded one which could not be made to work). Samples of these new functions are shown in appendix J.

Convergence Rates  The theoretical convergence rates for the various types of uniform or adaptive refinement are [19]:

- Uniform h-refinement:

  \[
  \|e\|_E \leq kN^{(-1/n_{\text{space}}) \min(p, \lambda)}
  \]  

  (2.10)

- Adaptive h-refinement:

  When the refinement is done adaptively, the error becomes uniformly distributed in all elements. In this case the mesh is called optimal and the effect of the singularity (\( \lambda \)) is eliminated.

  \[
  \|e\|_E \leq kN^{(-1/n_{\text{space}}) p}
  \]  

  (2.11)

- Uniform p-refinement:

  \[
  \|e\|_E \leq kN^{-\beta}
  \]  

  (2.12)

  The positive constant \( \beta \) depends on the smoothness of the exact solution.
• h-p-refinement:

\[ \| e \|_E \leq k \exp(-\alpha N^\gamma) \]  \hspace{1cm} (2.13)

The positive constants \( \alpha \) and \( \gamma \) depend on the smoothness of the exact solution and the finite element mesh.

These theoretical convergence rates are sometimes very accurate. In fig. 2.4 taken from [12, page 405] convergence rates for various refinement methods are shown for a short cantilever beam example with uniform loading. The singularity of the solution \( \lambda \) is 0.71 because of a 90 degree corner. Since the dimension of the space is two and the elements are linear then the exponent of \( N \) in equation 2.10 is equal to \( 0.71/2 = 0.355 \) which compares very well with the slope 0.356 for the linear uniform refinement case.

**Implementation of an h-refinement scheme**  Although hierarchical p-refinement methods are more promising than h-refinement, in this project an h-refinement method was employed since it was much easier to implement with the current facilities. Elements in FINITE/GP had no more than order two (\( p = 2 \)) shape functions and no infrastructure for adding hierarchical shape functions. So it was decided that an error estimator based on a modification of the Zienkiewicz-Zhu estimator [4] explained in the next chapter is to be created as well as an h-refinement strategy to be implemented. Details of their construction are given in the following two chapters.
$\lambda/2 = 0.356$, theoretical rate of convergence for uniform refinement

$p/2$, maximum rate of convergence

Experimental rates of convergence for short cantilever beam.

Figure 2.4 Convergence Rates for various refinement methods
Chapter 3

Development of an H-refinement Error Estimator

3.1 Introduction

Summary of chapter. In this chapter an error estimator proposed by Zienkiewicz and Zhu [4] is explained. Then, an algorithm based on this error estimator, modified by Professor J. Akin [26] of Rice University, Houston, is used for the programming of the subroutine "error.f" to be used with the finite element solver FINITE/GP. Finally, tests done to prove the correctness of the algorithm are discussed.

A better solution. According to Zienkiewicz and Zhu [4] in problems of linear elasticity and field problems, a $C_0$ continuity is assumed for the solution $\hat{u}$ resulting in a discontinuous gradient and subsequently, discontinuous stress. The discontinuity of these functions is the main idea behind this error estimator. It is quite intuitive that a continuous measure of these functions would be more accurate than a discontinuous one. A continuous measure can be calculated by a projection method (averaging). The situation is shown in figure 3.1 taken from [12]. The figure shows the exact stress ("1"), the continuous stress evaluated through a projection method ("3"), and the discontinuous stress found with the standard FEM method ("2").

3.2 Derivation of Error Measures

3.2.1 Model Problem

The problem to be investigated is a linear two-dimensional elastic problem although the analysis should be accurate for other field problems as well. The analysis should also apply to one- and three-dimensional problems. The differential equation governing the problem is:

$$Lu = f \quad \text{on} \quad \Omega \subset \mathbb{R}^2$$  \hspace{1cm} (3.1)

Where:
Figure 3.1  Exact (1), discontinuous (2), and continuous (3) stress functions for a one-dimensional problem for a linear (above) and a quadratic (below) approximation of the displacement $u$. 
• $L$ is a linear operator of the form $SDS$ with:
  D : Material information matrix and
  S : A lower order differential operator.

• $u$ is the true solution (displacement):

\[
  u = u_0 \quad \text{on } \partial^0 \Omega \\
  \frac{\partial u}{\partial n} = t_1 \quad \text{on } \partial^1 \Omega \\
  \text{with } \partial \Omega = \partial^0 \Omega \cup \partial^1 \Omega
\]

• $f$ in our case is a forcing function

Let the following definitions apply:

• $B$ : Global gradients of shape functions

• $\bar{u}$ : The FEM solution for the displacements at the nodes

• $\hat{u} = H \bar{u}$ : FE approximation of the solution

• $\bar{\sigma}^*$ : Vector with nodal averages of the stress (evaluated at each node)

Then the FE approximation of the stress function will be:

\[
  \hat{\sigma} = \begin{cases} 
    B \bar{u} & \text{heat} \\
    (DB)\bar{u} & \text{stress} 
  \end{cases}
\]

(3.2)

3.2.2 Calculating a Continuous Stress

Zienkiewicz and Zhu [4] suggest a nodal averaging of the stress done with a weighted residual method, using the same approximation for the stress as the one used for the displacement:

\[
  \sigma^* = H \bar{\sigma}^*
\]

(3.3)

\[
  \int_{\Omega} H^T(\sigma^* - \hat{\sigma})d\Omega = 0
\]

(3.4)

which result in:

\[
  \bar{\sigma}^* = A^{-1} \int_{\Omega} H^T DSH \, d\Omega \bar{u}
\]

(3.5)
where

$$A = \int_{\Omega} \mathbf{H}^T \mathbf{H} d\Omega$$  \hspace{1cm} (3.6)

If the inverse of $A$ is found using a diagonalized $A$ (through a "lumbing" procedure) the calculations above are quite simple. Nevertheless, in this thesis, the projection method used is not the one given above. It is simply an averaging of the stresses from the elements around each node.

### 3.2.3 Error Estimator and Indicators

After the evaluation of the continuous stress measure is done, the approximate error can then be evaluated as:

$$e_\sigma = \sigma^* - \hat{\sigma}$$  \hspace{1cm} (3.7)

Pointwise definitions of errors as in equation 3.7 are not very useful as mentioned in chapter 2, and usually an elementwise error or a total (over whole domain) error is required. Now a decision has to be made on the type of the norm to be used for the error estimator. Considering the energy norm and the $L_2$ stress norm the latter is chosen since this is the easiest to calculate in FINITE/GP, the FE program used in this thesis.

The integration required for finding the norm is performed using numerical integration (Gaussian quadratures). Since the stress is a vector function (having an $xx$ component, a $yy$ component, and an $xy$ component) it is necessary to use a dot product (instead of squaring) to calculate the error norm. The formula used for finding the error $L_2$ norm is the following:

$$\|e_\sigma\|_{L_2} = \sqrt{\int e_\sigma \cdot e_\sigma \, d\Omega} = \sqrt{\sum_{q=1}^{nq} \sum_{i=1}^{ns} e_{\sigma_i}^2 \mid J \mid_q W_q}$$  \hspace{1cm} (3.8)

Where:

- $nq$  Number of integration points
- $ns$  Number of stress components at each point
- $W_q$  Quadrature weight
- $J$    Jacobian function

This error norm (eq. 3.8) is an absolute measure. A relative measure can be evaluated if the error norm is divided by the sum of the error norm plus the stress $L_2$ norm.
The stress $L_2$ norm is:

$$
\|\sigma\|_{L_2} = \sqrt{\int \sigma \cdot \sigma \, d\Omega} = \sqrt{\sum_{q=1}^{n_q} \sum_{i=1}^{n_3} \sigma_i^2 |J|^q \, W_q}
$$

Finally, the relative error estimate is:

$$
\epsilon = \frac{\|e_\sigma\|_{L_2}}{\|\sigma\|_{L_2} + \|e_\sigma\|_{L_2}}
$$

**Error Indicators.** A point to notice here is that all the above integrations have to be done elementwise. So in the process of getting the element error contributions it is very convenient to create elementwise error indicators. The whole process of calculating the error estimator and the error indicators is explained in section 3.3 where the algorithm for calculating them is explained.

**Zienkiewicz Factors** According to Zienkiewicz and Zhu [4] the error estimates have been found to be underestimating the true error. Different elements have shown different behavior and the error is "corrected" with the use of the factors in table 3.1 taken from [4].

### 3.3 Algorithm for Error.f

Error.f (appendix E) is a subroutine programmed to calculate error measures for the FE solver FINITE/GP. It has been created according to the above guidelines and is

<table>
<thead>
<tr>
<th>Element Type</th>
<th>$C^e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4 (quadrilateral linear)</td>
<td>1.1</td>
</tr>
<tr>
<td>T3 (triangular linear)</td>
<td>1.3</td>
</tr>
<tr>
<td>Q9 (quadrilateral quadratic)</td>
<td>1.6</td>
</tr>
<tr>
<td>T6 (triangular quadratic)</td>
<td>1.4</td>
</tr>
</tbody>
</table>

**Table 3.1** Zienkiewicz correction factors
based on the algorithm below\textsuperscript{1}. Names in square brackets denote the actual names used in the subroutine.

1. Initialize summation variables for summing over all the elements.

2. For each element [ie] (finding the element error indicators):

   a. Extract element degrees of freedom \([d(nelfre)]\) from the solution \([dd(ndfree)]\).

   b. Extract the element nodal averaged gradients (for heat problems) or stresses (for stress problems) \([elave(n,ns)]\) from the global nodal averaged gradients vector \([ave(m,nsp1)]\). This global vector with the nodal averages was evaluated in another subroutine.

   c. Evaluate the integrals for the error norm and the gradient norm by looping over the quadrature points to perform numerical integration. So for each quadrature \([q]\) do:

      i. Read from the storage file values of interpolation functions \([h(n)]\), the values of global gradients (B or DB) of the interpolation functions \([b(ns,nelfre)]\), as well as the values of the product of the Jacobian times the quadrature weight \([detwt]\), (all evaluated at the current quadrature point).

      ii. For every stress component \(j\) (remember that the stress function is a vector function) at the current quadrature point \([j = 1, ns]\):

         A. Calculate the component \(\sigma_j^* = estar(j)\) of the continuous stress by dotting the interpolation function values vector (of size equal to the number of nodes on the element) \([h(n)]\) with the nodal continuous stress values vector \([elave(n,j)]\). Remember that everything is done for the current quadrature point:

         \[
         \sigma_j^* = h.\tilde{\sigma}_j^* 
         \]

         B. Calculate the component \(\delta_j = ehat(j)\) of the discontinuous stress by dotting the global derivatives of the shape functions vector \([B(j,n)]\) with the nodal solutions vector \([d(n)]\):

         \[
         \delta_j = B_j.\tilde{u} 
         \]

\textsuperscript{1}Some names have been used rather freely (a very common programming practice) to mean more than one thing. For example, many times the word "norm" may actually refer to its square; also the words gradient and stress many times are used interchangeably.
C. Calculate the difference $e_{\sigma_j}$ [eel] by subtracting the discontinuous stress vector [ehat(ns)] from the continuous one [estar(ns)]:

$$e_{\sigma_j} = \sigma_j^* - \sigma_j$$

(End of loop over stress components)

iii. Calculate dot product [sum] of the vector with the values of the error at the current point for all stress components $e_{\sigma}$ [eel(ns)] dotted with itself:

$$sum = e_{\sigma}.e_{\sigma}$$

(3.9)

iv. Calculate dot product [sum2] of the vector with the values of all the stress components $e_{\sigma}$ [estar(ns)] at the current point dotted with itself:

$$sum2 = \sigma^*.\sigma^*$$

(3.10)

v. Update numerical integration sum for the current quadrature point:

A. Add to the current sum [eNorm(ie)] the product of $sum_q$ in eq 3.9 times the product ($|J_q| W_q$) [detwt] of the determinant of the Jacobian times the weight:

$$e\text{Norm}(ie) = \sum sum_q |J_q| W_q$$

B. Add to the current sum [gNorm] the product of $sum2$ in eq 3.10 times the product ($|J_q| W_q$) [detwt] of the determinant of the Jacobian times the weight:

$$g\text{Norm} = \sum sum2_q |J_q| W_q$$

Now the numerical summations over the quadrature points of the current element are over.

(d) To get global measures for the error norm [sysErr] and the gradient norm [sysNor] update the sums with the current element error [eNorm(ie)] and gradient norm [gNorm] respectively:

$$sysErr = \sum e\text{norm}(ie)$$

$$sysNor = \sum g\text{norm}$$
(e) The correct value for the absolute value of current element error indicator is:
\[ e\text{Norm}(ie) = \sqrt{e\text{Norm}(ic)} \]

(End of the elementwise calculations to find the element error indicators)

3. Calculate the global error estimator (percentage) [sysPercErr] as the product of 100 times the square root of the ratio of the system gradient error norm measure [sysErr] over its sum with the system gradient norm measure:
\[ \epsilon = 100\sqrt{\text{sysErr}/(\text{sysErr} + \text{sysNor})} \]

4. Apply the empirical correlation factor if asked for. Currently this is disabled until further investigation can be done.

5. Find a RMS element norm to be used later to calculate a relative element error indicator. This is the square root of the ratio of the sum of the system gradient norm plus the system error norm divided by the number of elements:
\[ \text{avEINo} = \sqrt{(\text{sysNor} + \text{SysErr})/m} \]

6. Loop over each element to find relative error indicators:
   
   (a) Find the % relative error indicator for the current element:
   \[ \eta = 100\, e\text{Norm}(ie)/\text{avEINo} \]

   (b) Output all the elements having a % relative error indicator higher than the highest permissible value.

7. Output data required for viewing the error with the graphic post-processors CAEDS and MOVIE.BYU.

8. End of the error subroutine.

3.4 Patch Test of the Error Estimator

After the error subroutine was written, a suitable way of ensuring its validity was necessary. A first approach was the use of the patch test. The patch test, put very
simply, is a small problem set up to have a constant gradient everywhere. The FE code should produce exact results for a patch test [12], so it was expected that the error estimator should have a value of "machine-zero" (i.e. zero according to machine accuracy).

The results were satisfactory. Patch tests were performed for nearly all the types of elements supported by FINITE/GP, and all resulted in error estimators of "machine-zero" (about $10^{-6}$ for single precision) as expected.

A representative example of a patch test is shown in appendix A and describes a heat transfer problem in a square domain. The temperatures are prescribed on the boundaries so as to produce a constant gradient. These essential conditions are the only forcing terms. The temperature at the interior node 5 (figure at the bottom of the data file in the same appendix) is correctly predicted by FINITE/GP as 75 degrees and the error indicators are of the order of $10^{-7}$ (10$^{-5}$ divided by 100 to remove the percentage).

### 3.5 Discussion

Although the error estimator performed satisfactorily against the patch tests, a need for tests against benchmark problems was obvious. The following chapter describes the use of the error estimator within an h-adaptive refinement scheme to solve two problems with a singularity. A good performance of the error estimator was observed.

The error estimator in this thesis is the stress error $L_2$ norm. In stress problems, FEM is based on the minimization of the energy of the solution so an estimator based on the error in the energy norm might be most advantageous. The effect of the use of an energy norm should be investigated in future work.
Chapter 4

Implementation of an H-adaptive Refinement Scheme

4.1 Introduction

In the previous chapter an error estimator suitable for adaptive h-refinement was discussed and the implementation details were given. This error estimator is used in this chapter for the construction of h-refinement schemes (i.e. refinement done by reduction of the mesh size).

Objectives The objectives of this chapter are:

1. Investigation of h-adaptive refinement schemes.

2. Implementation of h-adaptive refinement schemes for the analysis of two sample problems.

3. Comparison of convergence rates of the solution against published results. These rates can be observed as slopes in plots of the error norm vs the number of nodes (fig. 4.5).

The main problem in the implementation of the adaptive scheme was the creation of the FINITE/GP input data files containing the initial and refined meshes. The creation of a general program to do this for any kind of problem would be very complicated and time-consuming. For this reason it was decided that the program supposed to create the input data would be specific to the example problems to be tried. For this reason simple enough example problems were needed. In the next section the selection process for the sample problems is given.
4.2 Choosing Sample Problems

4.2.1 Desirable features

The problem to test the adaptive refinement, would hopefully have the following features:

1. Problem with singularity: Since most of the practical engineering problems are singular to some extent, if the problem has a singularity it will make more of a real life problem and less of a mathematical model with no engineering significance.

2. Exact solution should be available: This was very important since it would facilitate the process of evaluating the accuracy of the numerical solution.

3. Nature of the problem: Only linear elliptic problems would be considered.

4. Simple Geometry: This would facilitate the process of refinement. Also a 2D region would be desirable since a 3D geometry would be too complicated (while a 1D geometry would be too simple).

4.2.2 Candidates for the Problem to Test the Adaptive Refinement

The problems considered as candidates were:

1. A thermal problem given by Myers [22]:
   - Advantages: 1) exact solution (although in the form of an infinite series), 2) linear flux conditions.
   - Disadvantages: no singularity.

2. An L-shaped region analyzed by Zienkiewicz et al [12]:
   - Advantages: 1) linear flux conditions, 2) singular.
   - Disadvantages: exact solution not available to the author.

3. A simple beam supported at one end and with uniform traction at the other end, given by Kelly in [2]:
   - Advantages: 1) linear flux conditions, 2) singular.
Disadvantages: exact solution not available.

4. Laplace equation over a rectangular domain given by Babuska et al [15]:

- Advantages: 1) exact solution, 2) singular.
- Disadvantages: Non-linear flux.

The last problem (referred to as the Babuska problem) was considered to be the most suitable for a first check of the adaptive scheme mainly because of the fact that it had a known solution. The L-shaped problem was also tested since many publications have discussed it.

4.3 Explanation of the Babuska Problem

The geometrical aspects of the problem are shown in figure 4.1. The figure describes a two-dimensional steady state heat transfer problem with one side (side OA) having a prescribed temperature of zero degrees (all units are in SI units). The heat transfer is driven by fluxes specified in all the other sides of the domain. The conductivity of the material is one, and internal heat generation is zero. The differential system of equations governing the problem is:

\[ L(T) = f \quad \text{on} \quad \Omega \subset R^2 \]  \hspace{1cm} (4.1)

and boundary conditions

\[ T = 0 \quad \text{on} \quad \partial^0 \Omega \]
\[ \frac{\partial T}{\partial n} = t \quad \text{on} \quad \partial^1 \Omega \]

where

- \( L \): the negative of the Laplace operator.
- \( T \): temperature.
- \( f = 0 \): heat generation term.
- \( \Omega, \partial^0 \Omega, \) and \( \partial^1 \Omega \) (shown in fig. 4.1): domain, essential, and Neumann boundaries respectively.
• \( t \): The temperature gradient defined to satisfy the (known in advance) solution:

\[
T = C \cdot r^{1/2} \cdot \sin(\theta/2)
\]  

(4.2)

with

• \( C = 0.0700754 \)

• \( r \) and \( \theta \) are the polar coordinates with origin at \( O \) (fig. 4.1).

For finding \( t \) the familiar polar coordinate formulae were found useful:

\[
|t| = \begin{bmatrix}
\frac{\partial T}{\partial x} = \frac{\partial T}{\partial r} \cdot \cos(\theta) - \frac{\partial T}{\partial \theta} \cdot \sin(\theta) \\
\frac{\partial T}{\partial y} = \frac{\partial T}{\partial r} \cdot \sin(\theta) + \frac{\partial T}{\partial \theta} \cdot \cos(\theta)
\end{bmatrix}
\]

With:

\[
\frac{\partial T}{\partial r} = 0.5 \cdot C \cdot r^{-0.5} \cdot \sin(\theta/2)
\]

\[
\frac{\partial T}{\partial \theta} = \frac{1}{r} \cdot 0.5 \cdot C \cdot r^{0.5} \cdot \cos(\theta/2)
\]

4.4 Explanation of the L-shaped Problem

The geometrical aspects of the problem are shown in figure 4.2. The figure describes a two-dimensional static plain stress problem with one side having a unit pressure (in SI units) applied to it. It is supported with roller supports on the upper and right sides. The elastic modulus of the material is unity and the Poisson's ratio is 0.3. Because of the geometry a singularity is developed at the corner which in practical terms means that a stress concentration exists there.

4.5 Refinement Strategies

Every time FINITE/GP is run, the error estimator, attached to FINITE/GP is set up to produce a file with all the errored elements i.e. the elements that have an error of more than an input threshold value. In the current experiment the threshold value is set to 10%. This file will be used in guiding the refinement process.

Two adaptive (as well as a uniform refinement) strategies will be investigated in this project. The adaptive methods are the buffer method which is the main method
Figure 4.1 Babuska Problem Definition: side OA has temperature \( T = 0 \), sides AB, BD, DE, and EO have prescribed fluxes.

Figure 4.2 L-shaped Problem Definition
and the standard method. The standard method is used more for comparison purposes and is implemented less carefully than the buffer method. An explanation of the methods is given below after a small introduction on the problem of continuity across element boundaries and irregular nodes.

4.5.1 Continuity Across Boundaries and Irregular Nodes

In finite element analysis, continuity of the solution across element boundaries is very important. The concept is shown in figure 4.3 and explained below. The figure shows a FE mesh with linear shape functions, an element boundary "A" that is continuous, and an element boundary "B" that is discontinuous because of the existence of node #2. Node #2 is referred to as an irregular node [15]. Also, it can further be characterized as a father-son level node, a term borrowed from tree structure theory, since it is created at the area of two consecutive refinement levels. There exist irregular nodes of higher than a father-son level as well.

More examples of "irregular nodes" are shown in figure 4.4 where the nodes defined with a small circle ("o") are "father-grandson level irregular nodes", and the nodes defined with a cross ("x") are "father-son level irregular nodes". Particularly, node #3 is a "father-son level node", while node #1 is a "father-grandson node" (since the elements above it are two levels of refinement higher than the element below).

Constraining the irregular nodes. The discontinuity at the boundary "B" of the first figure 4.3 can be avoided if the "irregular node" #2 is constrained to take a value given by the linear relation that defines the edge with the two edge nodes #1 and #3. FINITE/GP has been refined so that it is able to handle nodes of the father-son level through the above procedure, and could theoretically also be used for irregular nodes of a higher than father-son level. Unfortunately, for higher levels, FINITE/GP does not produce correct results yet. For this reason, refinement was done carefully to avoid meshes with the higher (than parent-son) level irregular nodes. A mesh which has only "father-son irregular nodes" will be called an acceptable mesh.

4.5.2 Buffer Refinement Method

The buffer method, suggested by Warren and Moore [21], makes the following assumption:

Assumption: The errored elements at each level will always be "sons" of elements that
**Figure 4.3** Element Boundary Continuity

were errored in the previous level of refinement.
So an “acceptable” mesh can be guaranteed if in every refinement stage not only the errored elements are subdivided but also the elements that share one or more nodes with the errored elements are subdivided. So the two main steps in the buffer method are:

1. The errored elements are subdivided into four smaller elements.

2. The elements that share one or more nodes with an errored element are subdivided in the same fashion.

These extra elements that are subdivided in the second step form a buffer which avoids creating irregular nodes of higher (than parent-son) level. Examples of the buffer refinement method can be seen in the color figures following the end of this chapter.

**4.5.3 Standard Refinement Method**

This method was tried just to get the overall idea of how it compares with the buffer method. Since time did not permit a better solution, the errored elements were subdivided only if they produced an “acceptable” mesh as explained in subsection 4.5.2.
Figure 4.4 Definitions associated with the refinement process
If they do not then they are not refined. Although this is not the best approach, alternatives would be too complicated.

4.5.4 Uniform Refinement

As the name implies, uniform refinement is done without accounting for the error estimator. All the elements are subdivided irrespective of how much they contribute to error in the solution.

4.6 Production of the FINITE/GP Input Data File

4.6.1 Selection of an Adaptive Refinement Program

A major problem in adaptive refinement is the input data generation. If done manually, the production of the data files for the original and the refined meshes to be input to FINITE/GP would be nearly impossible to do. So the following choices for creating the data files were explored:

1. Creation of a specific program from scratch:
   - Disadvantages: 1) much programming would be required.

2. Use a Finite Element commercial software, CAEDS or PAFEC:
   - Advantages: powerful pre- and post-processing would be readily available.
   - Disadvantages: Adaptive capabilities were not available.

3. Use of a mesh generator provided by Professor J.E. Akin of Rice University:
   - Advantages: FORTRAN source code available.
   - Disadvantages: learning the program specifics.

4. Use of a program provided by Professor J. Warren of Rice University, which was written to do the buffer method of refinement:
   - Advantages: Source code available and adaptive capabilities already in existence.
The "Ref" Program. The last program was chosen since it already had an adaptive refinement scheme built into it. Modifications were applied to it to make it produce data needed for FINITE/GP in the right format. The final product was named ref (from the word REFine). The modified version of the "ref" program is shown in appendix B.

Input Data to FINITE/GP produced by "ref" The input data file to FINITE/GP has the name "babu1.dat" coming from BABUska, whose problem was used for this experiment, and "1" denotes the first (initial) refinement level, similarly for 2, 3, 4, and 5. As an example, the file "babu2.dat", produced by "ref" at the second level of refinement, is shown in appendix D.3. The following data were needed inside the file [3]:

1. Initial header.
2. Node coordinates.
3. Element Topology.
4. Flux or pressure data.
5. Constraint #1 data i.e. temperatures or displacements specified.
6. Constraint #3 data where the "irregular nodes" are constrained as mentioned in subsection 4.5.1.
7. Material data

4.6.2 Input Files to the Refining Program

The "ref" program was created to produce the FINITE/GP input data files; the "ref" program itself required its own input data files. These files were two quite simple files:

1. meshin: A file containing the list of the coordinates of the lower left hand corners and sizes of all the existing squares. An example of this file is in appendix C.1. The initial "meshin" file was created by hand but subsequent ones were created by the previous run of "ref".

2. error-file: A list of all the errored elements (see appendix C.2). This data file was produced by FINITE/GP.
4.7 Comments on Changes Made to FINITE/GP

4.7.1 Interfacing FINITE/GP with Pre- and Post- Processing Programs

In the development of this experiment it was felt that a means for pre- and post-
processing of the data for FINITE/GP was absolutely needed for efficient development
of the refinement methods. The ideal candidate to give these capabilities would be
easy to interface, have color capabilities and provide for hard copies. The following
programs were considered:

1. MOVIE.BYU: Although this provided for color images and was easy enough
to interface, it had some bugs that made it occasionally difficult to use.

2. CAEDS: It was rather slow to start and it was available only on certain
computers (IBM RTs), but it had color capabilities and very easy and powerful
menu structure.

3. A program provided by Professor Ramaswamy of Rice University, Houston:
Although this seemed to be a good choice for hard copies and contouring, it did
not print the node and element numbers.

4. MATLAB, S, NCAR: These had very limited capabilities for showing ele-
ments and nodes too.

CAEDS was found to be the best overall choice with the only disadvantage that
it had only color hardcopy capabilities which are expensive to create. To bypass this
problem, MOVIE.BYU was also used for black-and-white hardcopies. So interfaces
were written for these two packages. These interfaces called caeds.f and movie.f are
shown in appendices G, and F respectively.

Interfacing with MOVIE.BYU and CAEDS was done for topology, coordinates,
stress, deformation, and error. For the time being, the CAEDS interface works only
for pyramids and linear quadrilaterals.

4.7.2 Capability for Using Different Element Types Installed

It is a common practice in adaptive refinement to use triangular elements (in 2D)
for the transition areas (i.e. the areas between fine and course meshes, shown in
figure 4.4). So the capability for using triangular and quadrilateral elements together
was installed (appendix K). Nevertheless, it was never applied to the experiment of this chapter since it was later realized that the constraint method, explained in subsection 4.5.1, was easier to use (and equally effective) than the use of triangular elements.

### 4.7.3 Other FINITE/GP Changes

**New Debugging Features**

1. The stiffness matrix is output if the right flags are changed.
2. The first two element stiffness matrices are output.
3. All the output goes to fort.7 instead of going to the screen.
4. Flux data are input (in the FINITE/GP input data file) in “free format” now. That means that old input data files may not work now if no separation exists between the numbers.

**Other Modifications**

1. The constraint #3 has been corrected.
2. Degenerate elements (i.e. with zero area) are now taken care of.
3. Before, a constraint had to be shown (coordinates data card) at all the nodes it affected. Now it needs to be shown only at one node. For example a constraint #3 had to be placed at all three nodes it affected, while now it needs to be placed only at one node.

### 4.8 Refinement Procedure

The procedure for the three refinement methods used was very similar. The buffer method procedure is given first with much detail since that was the main method. The differences between the procedure of the other two methods and the procedure of the buffer method are then mentioned.
Buffer Method Procedure. The detailed procedure for the buffer method as applied to the Babuska problem was the following: A data file “meshin1” (meshin1 means meshin level 1) shown in appendix C.1 was created with the help of “ref”. It had a moderately fine, uniform mesh with 32 (8x4) elements. The first color figure at the end of this chapter represents this mesh. This file together with an empty “error-file” (to denote that no elements needed subdivision) were fed to “ref” to produce the input data file to FINITE/GP, “babu1.dat” and the file was input to FINITE/GP for the first run. This run of FINITE/GP was the first level of refinement and it produced the “error-file1” (appendix C.2). The file produced by this run, together with the same “meshin1”, were fed to “ref”. A refined mesh, which we will call the second level “babu2.dat” (appendix D.3), was produced, as well as the next level “meshin2” (appendix D.1). The second color figure at the end of the chapter shows the new refined mesh corresponding to the data file “babu2.dat”. This data file was then fed to FINITE/GP to produce a new “error-file2” (appendix D.2) which, together with the new “meshin2”, were fed to “ref” to produce the third level “babu3.dat”. The procedure for the next two levels of the refinement follows the same pattern.

Standard Method Procedure. The procedure for the standard method of adaptive refinement was the same except for the use of a modified “ref”. The modifications were: 1) the buffer subdivision part was removed and 2) the errored elements that would produce an “unacceptable mesh” were not subdivided any further.

Uniform Method Procedure. Finally the procedure for the uniform case was much simpler. The main difference from the above cases was that a fake “error-file”, containing all the elements, was used instead of the file produced by the run of FINITE/GP.

4.9 Experimental Results

4.9.1 Babuska Problem

As explained in section 4.8, figures taken from the various phases of the buffer method with the software CAEDS on an IBM RT are shown in the pages following this chapter. The figures show the meshes used, and the contours of the error in the solution, as well as the node and element numbering.
Condensed results for all methods of refinement used are shown in fig. 4.5. The figure depicts the relationship of relative percentage error to the number of degrees of freedom in a log-log scale. Three different cases are shown:

1. Uniform Refinement case.


The following points can be easily seen from this graph:

1. The curve for the uniform case is a well defined straight line whose slope is \(-0.26\).

2. The curve for the "buffer adaptive case" as well as for the "standard adaptive case" converges to a line much slower than the uniform one, and it's slope is approximately \(-1.0\).

4.9.2 L-shaped Problem

For this problem, a slight modification of the buffer method was tried by allowing elements that are not active anymore to be subdivided. This happens only when its subdivision will still produce a "regular" mesh. The slope for this method was \(-0.495\). On the other hand the slope of the uniform refinement was \(-0.40\). The plot of the error vs the dof is shown in fig. 4.6. Representative stages of buffer adaptive refinement for this problem are shown in figure 4.7, and for uniform adaptive refinement are show in figure 4.8.

4.10 Discussion

Error Contours. In the figures at the end of the chapter, error contours can be seen as approximately concentric circles having as their center the singularity point O (fig. 4.1). As the refinement increases, the circles move inward (i.e. the error gets smaller).
Figure 4.5  Plot of percentage error vs number of nodes in a log-log scale for the adaptive and uniform refinement of the Babuska problem
Figure 4.6  Plot of percentage error vs number of nodes in a log-log scale for the adaptive and uniform refinement of the L-shaped problem.
Figure 4.7 Successive stages of adaptive refinement for the L-shaped problem, with error contours drawn
Figure 4.8  Successive stages of uniform refinement for the L-shaped problem, with error contours drawn
4.10.1 Convergence Rates

Babuska Problem. According to the literature, figure 4.5 (i.e. the plot of the relative error norm against the number of nodes) should show asymptotically linear curves in a log-log plot. The following is a comparison of the results of the experiment with the theoretical predictions:

1. Uniform Refinement: Eq. 2.10 applies here with:
   - \( n_{space} = 2 \)
   - \( p = 1 \)
   - \( \lambda = 0.5 \) (since \( \lambda \), according to Szabo [2], is the exponent of \( r \) in eq. 4.2).

   That means that the slope of the straight line should have a \(-0.25\) slope which shows an excellent agreement with the slope in the current experiment of \(-0.26\).

2. Adaptive Refinement: Eq. 2.11 applies so a slope of \(-0.5\) should be observed. Here the agreement breaks down since the slope of the “asymptote” to the curve as \( N \) increases is about double \((-1.0)\) the theoretical one \((-0.5)\) Reasons for this discrepancy have not been found yet, and further investigation is necessary.

L-shaped Problem. This problem was better behaved and the buffer adaptive refinement slope \((-0.495)\) was near to the theoretical prediction \((-0.5)\). This fact is very encouraging although further investigation needs to be carried out for finding the reasons that the Babuska problem does not show similar agreement with theory. On the other hand the uniform refinement slope \((-0.40)\) was higher than the theoretical \((-0.27)\) and further investigation is necessary for justification of the results.

4.10.2 Future Work

The following points need further investigations:

- The adaptive program used for this experiment was very specific to the problem being solved due to time limitations. Future work should point towards creating a more general adaptive scheme that will be able to solve any kind of problem at least in the family of linear elliptic problems.
• The assumption made in the buffer method, that errored elements are always "sons" of the previous level errored elements (subsection 4.5.2), needs further consideration. Currently the assumption seems to be only approximately correct and the possibility of changing the program to avoid having to make the assumption may be necessary.

• A major point in adaptive refinement is computational economy. High computational savings could be achieved if an iterative solver was used for the subsequent levels of refinement. In this experiment the existing direct solver was used at all times and the previous solutions were not utilized at all. It would definitely be worth implementing an iterative solver with FINITE/GP and experimenting with the economies of the refinement methods.

• As already mentioned in section 4.6.2, the "meshin" file (input to "ref") is created by the previous run of "ref". This is not a desirable feature since ideally FINITE/GP should have created it.

• The interface of FINITE/GP to the graphics program CAEDS is not completely finished since it accepts only linear quadrilaterals and linear tetrahedra. Changes to FINITE/GP to include the other element types may be quite useful.
LOADCASE: 1
STRESS ERROR DENSITY - MAG MIN: 1.15E-04 MAX: 6.73E-03
Chapter 5

Solvers for Sparse Symmetric Positive Definite Matrices

5.1 Introduction

This chapter was written as a documentation of work done on the Finite Element Solver FINITE/GP to make it able to solve a large system of equations efficiently. The system of equations, used as an example in this chapter, came from the analysis of a femur bone to be used in prosthesis work.

Computational Difficulties with Large Matrices. Solvers of large linear systems of equations usually consume a lot of computer resources. This fact is not surprising since:

- Storage space for general (in contrast to sparse) systems is on an average, proportional to the square of the number of unknowns $N$.

- Cpu time is proportional to the cube of $N$.

For a large number of unknowns, this quadratic and cubic dependence of storage space and run-time to $N$ creates many computational difficulties. These difficulties will be presented through an example in section 5.5.

Large Matrices Common in Engineering. Large systems of equations have become commonplace in engineering and science. Complicated partial differential equations need to be solved over complex domains. Computational methods for solving these problems, like the Finite Element Method, create systems of equations having a large number of unknowns. It is the limitations of large matrices, explained in the previous paragraphs, that have proved to be a limiting factor for much of analysis and research.
Sparse Systems of Equations and Sparse Solvers. Fortunately, the majority of large matrices are sparse. Sparse matrices are those associated with a stiffness matrix $K$ which has many more zeros than non-zero coefficients. This sparseness property of systems of equations has permitted the use of special storage schemes and special solution processes that take advantage of the sparseness and reduce the computational difficulties significantly. An example of a sparse system is discussed in section 5.5.

Cholesky Factorization. The rest of this chapter will deal with large matrices that are symmetric positive definite. This is a good assumption for the following reason. The main interest of this thesis is FE models that can be solved by FINITE/GP and it is a fact that this type of problems are always symmetric positive definite. This allows the use of the Cholesky factorization of the matrix $K$, into the product of a lower triangular matrix $L$ with its transpose $L^T$.

5.2 Hardware and Software used

Computer. The main computer used in this project, provided by the mathematical sciences department at Rice University, is a Convex computer, supporting vectorization. Vectorization was found quite useful with typical improvements in run-time of a factor of two. For this reason it was used in all the compilations of the solvers and the FINITE/GP code.

Fortran Compiler and Optimization Options. The native Fortran compiler of the Convex, called fc, shows a good overall performance. With the default being "no vectorization performed", a flag "-O2" has to be used during compilation for achieving vectorization. In addition, fc has other optimization features from which two have been tried: 1) the loop unrolling and 2) the potentially unsafe optimization. Both of them have shown no significant differences (at least when used with the bone data explained in section 5.5).

Debugger and Profiling Tool. The Convex debugger, called csd, operates similar to the "Berkley Unix" debugger (dbx). The Convex debugger performs much better than dbx in the area of array printing since it supports multicolumn printing and suppression of consecutive zeros. Unfortunately it seems to lack full support of
"dummy array" printing (i.e. the printing of the values of dummy arrays in subroutines). The profiling tool, called prof, can be used for finding the subroutines that use the most cpu time.

5.3 Ordering of the Matrix

5.3.1 Fill-in

A major problem when dealing with sparse matrices is that they lose their sparseness when they are being factorized. This is so since the solution process produces non-zeros at places where zeros existed before. This phenomenon, of zeros in $K$ becoming non-zeros in $L$, is called either fill-in or fill.

5.3.2 Ordering for Reducing Fill-in

A partial "fix" to the fill-in problem is the reordering of the equations. Figure 5.1 obtained from the same source (George and Liu [6]) compares graphically the unordered case along with the two main methods of ordering and the resulting fill-ins. The three different orderings of a symmetric $K$ matrix, having 144 non-zeros, are shown at the left part of the figure while the right-hand-side drawings show the corresponding $L$ matrices:

- The two drawings in the first row have the original (unordered) matrix $K$ on the left and the corresponding $L$ matrix on the right. The number of fill-ins in this $L$ matrix is 225.

- The second row shows the same matrix permuted in such a way to reduce the bandwidth (explained in subsection 5.4). Now, the $L$ matrix has 45 fill-ins, a reduction to the one fifth of the original. Notice that the fill-in remains inside the band, i.e. between the diagonal and the farthest (from the diagonal) non-zeros. That is why this ordering technique has tried to keep the non-zeros as near to the diagonal as possible. Both the bandwidth and the skyline solvers (see next section) require an ordering of this type.

- The third row shows the permutation obtained with a special method to reduce fill-in. This time the number of fill-ins becomes even less, 33. The general sparse solver (see next section) works best with this type of ordering.

So it is easy to realize how important the ordering is on the size of fill-in.
Figure 5.1 Effect of various kinds of ordering on fill-in (figure obtained from George and Liu [6])
5.4 Direct Sparse Solvers and Ordering Algorithms

This section talks about the various types of solvers and ordering algorithms that are suitable for them. The following direct sparse equation solvers are probably the most important:

1. **Bandwidth solver**: This solver is the simplest sparse matrix solver to program and it has the simplest data structure. It basically stores the part of the matrix from the diagonal to the furthest away non-zero (see fig. 5.2 taken from Akin [1, page 136]). It performs very well for small to medium size problems depending on the computer system used and on the bandwidth (see subsection 5.3.2) of the stiffness matrix.

The ordering of the equations depends on the method used for numbering the nodes. An example is shown in figure 5.3 where a good and a bad way of numbering are shown for a simple two-dimensional object. This type of numbering will ensure that the bandwidth remains fairly small and that space and time requirements remain low. In more complicated systems, numbering has to be done automatically through an ordering algorithm. The **Reverse Cuthill-McKee** method [6] is mostly used for this purpose. The method is a modification of the normal Cuthill-McKee reordering and produces a superior profile although the same bandwidth. This fact was first discovered and proved by Alan George [6].

2. **Skyline solver** (known also as profile or envelope solver) with the storage scheme shown in figure 5.4: It is similar to the bandwidth solver, with the only difference being that it takes advantage of the variation in bandwidth of the columns of the matrix to reduce the storage requirements. Some solvers may use rows instead of columns. Although not much more sophisticated, it can be much more efficient than the bandwidth method. This efficiency of the skyline solver will be stressed again later in the results and the discussion at the end of the chapter.

Again, the Reverse Cuthill-McKee is the most widely used method of ordering for this solver.

3. **General sparse solver**: Unlike the skyline and bandwidth solvers, this one records only the non-zero elements and fill-ins of the matrix. For this reason it needs supporting arrays that tell it where each non-zero’s or fill-in’s position
Figure 5.2  Bandwidth Storage: Drawing on the left depicts the outer non-zeros in the symmetric matrix and on the right shows the part of the matrix stored in the computer memory.

Figure 5.3  A numbering system for reducing the bandwidth.
in the stiffness matrix is. Its rather sophisticated data structure is harder to implement than the bandwidth or the skyline data structures. An example of a general sparse solver is the Yale sparse solver discussed in section 5.5.

According to George and Liu [6] there are many approaches to achieving a good reordering for this type of solver. The most important of them are the 1) One-Way Dissection, 2) the Nested Dissection, and 3) the Minimum Degree. The One-Way Dissection, designed also by A. George, is especially useful for the Finite Element Method [6].

4. Frontal solver: This solver is different from the others in that it does not store the whole stiffness matrix but instead it stores only equations that are currently active. Equations that have received all their element and constraint contributions are eliminated to provide storage for equations that will become active later [1]. The method seems to be very effective in reducing space requirements. For this reason it is often used with personal computer (pc) versions of commercial software. Two examples are:

---

1An attempt was made in this project to use this method to get some insight on the level of improvement that can be obtained this way. Unfortunately, the implementation details proved to be more complicated and cumbersome than expected, so the experiment was not completed.
- The frontal solver of the pc version of FINITE/GP.
- The frontal solver for the pc version of the code given by Zienkiewicz [12].

**Iterative Solvers.** Other than these direct solver one can use an *iterative solver*. The main advantages of this solver are: 1) it needs minimal amounts of memory since it does not need factorization that produces fill-in and 2) it may be faster than direct solvers given a good initial solution. The usual problems with this solver are 1) to find a good initial guess to obtain an acceptable convergence rate and 2) the uncertainty whether or not the solution will converge.

### 5.5 Implementation and Testing of Solvers

In this section results from using the build-in bandwidth solver of FINITE/GP and from implementing the skyline and the Yale sparse solvers will be presented.\(^2\) The versions of all the solvers used are for symmetric positive definite matrices since the FE code currently used in this project, FINITE/GP, deals only with this type of matrices.

**Bone Problem** A difficult problem tackled in this project was the modeling and analysis of the femur bone for subsequent use in artificial prosthesis work. The initial mesh had 2,400 nodes, and since this is a three dimensional problem having three unknown displacements per node, there was a total of 7,200 unknowns. This mesh came from a general mesh-generator provided by Professors D. Moore and J. Warren of Rice University, Houston. A plot of the bone geometry, provided by Professors D. Moore and J. Warren of Rice University is shown in figure 5.5.

#### 5.5.1 Trying the Bandwidth Solver

Once again, FINITE/GP was going to be used for the analysis of the bone. The analysis of the bone problem was tried, with the build-in bandwidth solver and without any ordering done. The attempt resulted in an abnormal stop. The bandwidth was found from this incomplete run to be nearly equal to the number of unknowns (6,855 compared to a number of equations equal to 7,182). In its current state the problem was unsolvable since it required the huge amount of 49 MWs (Mega Words)

---

\(^2\)Other solvers have not been tried because of time limitations.
Figure 5.5 The bone problem geometry
of memory something equivalent to 196 (4×49) MBs (Mega Bytes) for 4-byte words (one real word is the space required by the computer to store a real number). This was the reason for the abnormal stopping of the run.

The Ordering Program "Reduce". The need for a renumbering scheme was apparent. Reduce, a program, written by Professor J. Akin of Rice University, to perform the Reverse Cuthill-McKee ordering method, was used for this purpose. To make it interface correctly with FINITE/GP a small subroutine (appendix I) was written to format the output for FINITE/GP. With the use of this program the bandwidth was reduced ten-fold to about 700. This was a great improvement and the problem could be handled a little easier now, since it required the amount of 5 million real words.

Excessive time was needed. Workstations like sun-3's, and a Celerity computer (the fastest computer in the Mechanical Engineering Department of Rice University) were sometimes slowed to a halt due to the excessive memory and run-time required. Even with this reduced bandwidth the time required on the Celerity was more than 6 hours, a period too long for experimentation. An attempt to run the program on Cray X-MP in Austin failed since the operating system of Cray prohibited a program from using more than 4 million real words. Fortunately a rather new computer at Rice, a Convex, was made available and was used for the rest of the computations. A few details about the computer are given in section 5.2.

Results of the Bandwidth Solver on the Convex Computer. A first try of the bone data on the Convex machine was made. Without any optimization features used it took as much as six hours (360 minutes) for execution. A large improvement was obtained by the use of the vectorization option of the fc compiler. With this option the run-time was reduced to 150 minutes, less than half the original time. Since this was still a long time for someone to wait for answers, it was decided that an investigation of two other sparse solvers might be useful.

5.5.2 Implementing the Yale Sparse Solver

The next solver to be tried was a general sparse solver from the Yale Sparse Package. This solver was chosen because of convenience in obtaining it and previous experience
with it. A new data structure was needed with this solver, and for its creation, three small subroutines were written: preasb.f, intbin.f, and int_sort.f (appendix H).

After being debugged against a small dummy main, the new subroutine and the solver were linked to FINITE/GP and tried out against small data files to be compared with previous solutions. The solutions were correct, so the large bone data file was fed to the program.

Ideally an ordering like the One Way Dissection should be applied to this solver to get the best results out of it. Unfortunately this has not been possible due to implementation difficulties. Nevertheless, the bandwidth reduction ordering (the Reverse Cuthill-McKee) used before was comparatively effective even for this solver type. An indication of this was that when the original bone data without the reordering were passed to the Yale sparse solver the memory requirement was 12 MWs compared with 6 MWs for the Reverse Cuthill-McKee ordering.

Results. The results were very satisfactory as far as run-time was concerned since a three-fold decrease in time was observed. Instead of a run-time of 150 minutes the solution was obtained in 45 minutes. Unfortunately the results were not so good in memory requirements since a slight increase in memory took place. The memory requirements for the Yale sparse solver were about 6 million words, higher than the bandwidth solver requirements of 5 million words.

5.5.3 Implementing the Skyline Solver

The skyline solver is based on a solver by Hinton and Owen [24], modified by Professor J. E. Akin of Rice University. Since it is structured similar to the bandwidth solver and some of the data structures were already provided by existing subroutines, its implementation was easy.

Results. The results were again very satisfactory as far as run-time was concerned and not so satisfactory as far as space was concerned. The new run-time for the bone data was only 11 minutes, a very pleasant surprise. Compared to 150 minutes for the bandwidth solver, it is a twelve-fold decrease. Even compared to the Yale sparse solver (with a run-time of 45 minutes), it was still far superior. Unfortunately, the reduction in space requirements was much smaller, only 30 percent (from 5 MWs for the bandwidth down to 3.5 MWs for the skyline).
5.6 Discussion

Comparison of Solvers Tried. The results obtained show clearly the winning solver to be the profile solver. Its performance was clearly quite superior than the other two solvers (see table 5.1).³

Solvers Recommended. This chapter has been devoted completely to direct solvers which are dominating the Finite Element arena. Iterative solvers (subsection 5.4) should also be considered in cases of adaptive refinement where a first guess is readily available from the previous solution. It seems that a combination of a skyline solver with an iterative solver should be used in adaptive schemes. The skyline solver would solve the initial mesh and then the iterative solver would solve the refined meshes.

Reordering to be incorporated in the Finite Element Code Currently FINITE/GP does not provide for any means of reordering. From the above discussion it seems that the addition of a reordering ability would improve the ability of the code to handle complicated problems with large stiffness matrices. Future work should be done in this direction.

Total number of non-zeros in stiffness matrix An intermediate result of the Yale sparse solver implementation was the number of non-zeros in the stiffness matrix $K$. This was found to be about 103,000 for the bone data which had a 7,200x7,200

<table>
<thead>
<tr>
<th></th>
<th>Skyline</th>
<th>Yale</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time (min)</td>
<td>11</td>
<td>45</td>
<td>150</td>
</tr>
<tr>
<td>Memory (MW)</td>
<td>3.5</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.1 Comparison of the three solvers shows the skyline solver as the best performer

³The results for the Yale solver can not be considered very accurate since a good ordering for the Yale solver was not available, a fact that may have harmed its performance.
stiffness matrix. This indicates a "non-zero to zero ratio" of 1 to 350 or alternatively 20 non-zeros for each matrix row. Compare this result with the bandwidth of 700 that was obtained after reordering of the nodes. This means that even the bandwidth method was storing mostly zeros in a ratio of 35 zeros to 1 non-zero.

5.6.1 The Bone Problem Results

The bone problem was explained in section 5.5. It was a very refined model with 7200 elements in it. Less refined meshes were not very desirable since the model creation method is a delicate one and it might otherwise create an inaccurate shape. The objective in this problem was to find the stress distribution on the bone and to estimate the accuracy of the solution.

The problem was run on the Convex with simple loading conditions. It produced very high error estimates in many elements, indicating that much higher refinement was needed. Unfortunately, further refinement seemed difficult, given that, even with the current mesh size, many computational problems existed already, including excessive memory space requirements and long run times. For this reason no further refinement was performed. Future research should consider using quadratic instead of linear elements, since quadratic elements have a much higher accuracy than linear elements.
Bibliography


Appendix A

Patch Test for the Error Estimator

A.1 Input File

4.dat: Heat Transfer Patch Test 2D Linear Quadrilaterals

0004  9 0004 0004 0001

1  1  0.  0.
2  1  5.0  0.0
3  1 10.0  0.0
4  1 10.0  5.0
5  0  5.0  5.0
6  1  0.0  5.0
7  1  0.0 10.0
8  1  5.0 10.0
9  1 10.0 10.0

00001000001000010000200000500006

2  1  2  3  4  5
3  1  5  4  9  8
4  1  6  5  8  7

0000100001  50.0
2  1  50.0
3  1  50.0
4  1  75.0
6  1  75.0
7  1 100.0
8  1 100.0
9  1 100.0

00001 1.0  1.0

COMMENTS
This is a patch test with all the external nodes having their temperature prescribed through essential boundary conditions in such a way that a linear variation of the temperature should result. The only node that is not assigned a temperature is node 5. The FE analysis should calculate its temperature to be 75 degrees or else something is wrong.

Temperature

7________8________9 <-----10 meters------100 degrees K
|    |    |   |
| 4  |  3  |

6|________5|_______|4 <-----5 meters------75 degrees K
|    |    |
| 1  |  2  |

|_______|_______| <-----0 meters------ 50 degrees K
| 1    |  2    |  3   |

0 meters 5 meters 10 meters Dimensions

A.2 Parts of Output File

1******************************************************************************************************
4.dat: Heat Transfer Patch Test 2D Linear Quadrilaterals
******************************************************************************************************

1
FINITE VERSION 7
GENERAL PURPOSE FINITE ELEMENT ANALYSIS
PROGRAMMED BY:
PROF. J. E. AKIN, P.E.
RICE UNIVERSITY
DEPT. OF MECHANICAL ENGINEERING & MATERIALS SCIENCE
P. O. BOX 1892
HOUSTON, TX 77251
......

1*** OUTPUT OF RESULTS ***

NODE, 2 COORDINATES, 1 PARAMETERS.

1 0.000000E+00 0.000000E+00 5.000000E+01
2 5.000000E+00 0.000000E+00 5.000000E+01
3 1.000000E+01 0.000000E+00 5.000000E+01
4 1.000000E+01 5.000000E+00 7.500000E+01
5 5.000000E+00 5.000000E+00 7.500000E+01<---- THIS VALUE IS | 75 AS EXPECTED |
6 0.000000E+00 5.000000E+00 7.500000E+01
7 0.000000E+00 1.000000E+01 1.000000E+02
8 5.000000E+00 1.000000E+01 1.000000E+02
9 1.000000E+01 1.000000E+01 1.000000E+02

------------------------------------------------------------------

sysPercErr, cutoff, sysErr, sysno, avElNo:
---------- ---- ---------- ------ ----
1.26159E-05 1.66667 6.30796E-06 50.0000 16.6667

------------------------------------------------------------------
The % error estim. (enorm) for el # 1 is 7.15256E-06
enorm(ie) = 1.19209E-06
The % error estim. (enorm) for el # 2 is 7.15256E-06
enorm(ie) = 1.19209E-06
The % error estim. (enorm) for el # 3 is 2.57889E-05
enorm(ie) = 4.29815E-06
The % error estim. (enorm) for el # 4 is 2.57889E-05
enorm(ie) = 4.29815E-06
\/
NORMAL ENDING OF FINITE/GP PROGRAM.

---------

| ABOUT 10E-7 |
| IF MULTIPLIED |
| BY 100 FOR THE |
| PERCENT |

------------------------
Appendix B

Program "ref.c"

#include <stdio.h>
#include <math.h>

/************************************
 *
 * main.c :
 *
 * PURPOSE: 2D version of Buffer Method for adaptive mesh
 * generation.
 * Input is a quadtree and collection of elements in the
 * the quadtree that need to be split. It outputs data files
 * to be used for FINITE/GP FEM software package.
 * For now it is set up to create the complete data file for
 * the Babuska problem
 *
 * The nodes should fit in an integer grid of size SQUARESIZE
 * The square domain has to be SQUARESIZE x SQUARESIZE to make
 * conversion of the integer to the real domain to work
 *
 * WARNING: Element being split must be a deepest level and
 * not be adjacent to any element at a shallower level
 * Deepest level square can not be less than size = 2
 *
 * BUGS: The arrays getx and gety start form 0 instead of 1 ??
 * The errored elements are not checked if they
 * exist or not
 *
 * This program is a modification of a program by Professor
 * J. Warren of Rice University

68
#define SQUARESIZE 128 /* 2 to the max depth */
/*Below: It has to be 3* squaresize for padding purposes*/
#define MAXSIZE 3*128
#define DUMP 1      /* Not really needed now*/
#define DEBUG 1
#define XZERO -1.0  /* defines the physical space minimum x */
#define YZERO 0.0   /* defines the physical space minimum y */
#define XRANGE 2.0  /* defines the physical space range in x dir */
#define YRANGE 1.0  /* defines the physical space range in y dir */
#define MATERIAL 1  /* material number for the babu.dat file*/
#define PI 3.14159265358979
#define TRUE 1
#define FALSE 0

/* BELOW: array: size of element with i,j as lower left coords */
static int array[MAXSIZE][MAXSIZE];
static int label[MAXSIZE][MAXSIZE]; /* Vertex (node) number */
static int core[SQUARESIZE][SQUARESIZE]; /* Level of refinement*/

FILE *fopen(), *fpnode, *fpelem,*fpedge, *fpnewin,
     *fpredispl, *ferror, *fpmeshin, *fpflux,*fpconstr;

*****************************************************************************

*  
*  grad :  
*  
*  PURPOSE: produces the flux in the x and y direction at point  
* (x,y) for the Babuska problem given the point's coordinates  
*  
*****************************************************************************

void grad ( x, y,fluxX,fluxY)
double x,y,*fluxX,*fluxY;
    /* (Above: Did not need to use pointers) */
{
    double k,r,th,dTr,dTth,dTx,dTy;
    k=0.0700754;
    r=sqrt(x*x+y*y);
    /* To find "th" I need to check three cases */
    if (x==0)
        th=PI/2;
    if (x > 0)
        th= atan(y/x);
    if (x < 0)
        th= atan(y/x)+PI;

    /* Find the gradient components in the "r" and "th" direction, */
    /* used for finding the components in the x and y directions */
    dTr= k*.5/sqrt(r)*sin(th/2);
    dTth= k*.5/sqrt(r)*cos(th/2);
    dTx= dTr*cos(th) - dTth *sin(th);
    dTy= dTr*sin(th) + dTth *cos(th);
    /*I will take care of the flux sign later , */
    /* now positive means towards the left flux*/
    *fluxX=dTx;
    *fluxY=dTy;

    if (DEBUG)
    {
        printf("from grad: %10.8f %10.8f %10.8f %10.8f\n",
            x,y,*fluxX,*fluxY);

        printf("%f %f %f %f\n",k,r,th,dTr,dTth);
        printf("%f %f %f %f\n",x,y,*fluxX,*fluxY);
    }
}
main()
{
    int level, i, j, VertNum, ElemNum, SplitNum, split, x, y, s, t;
    /*Below: Gives the x integer coordinate of each element */
    int getx[SQUARESIZE*SQUARESIZE];
    /*Below: Gives the y integer coordinate of each element */
    int gety[SQUARESIZE*SQUARESIZE];
    int elem, constraint;
    double rx, ry, xslope, yslope, right, up, ry1, ry2, fluxX1, fluxX2, fluxY1,
           fluxY2, rx1, rx2;
    int nodesNum, nseg;

    fpnode = fopen("nodes", "w");
    fpelem = fopen("elements", "w");
    fpedge = fopen("edges", "w");
    fpnewin = fopen("newinput", "w");
    fpindispl = fopen("indispl", "w");
    fpererror = fopen("error", "r");
    fpmeshin = fopen("meshin", "r");
    fpflux = fopen("flux", "w");
    fpconstr = fopen("constr", "w");

    /* INITIALIZE */

    for (i=0; i<MAXSIZE;++i)
        for (j=0; j<MAXSIZE;++j)
            array[i][j] = -1;

    for (i=0; i<MAXSIZE;++i)
        for (j=0; j<MAXSIZE;++j)
            label[i][j] = -1;

    for (i=0; i<SQUARESIZE;++i)
for (j=0; j<SQUARESIZE; j++)
    core[i][j] = 0;

/* Initialize so that if -1 the element argument of getx
does not exist */
for (i=0; i<SQUARESIZE; i++)
    for (j=0; j<SQUARESIZE; j++)
        getx[i*SQUARESIZE+j] = -1;

/* INPUT */

/* Input number of elements */
 fscanf(fpmeshin, "%d ", &ElemNum);

for (i=0; i<ElemNum; i++)
{
    /* Square with lower left hand corner at (x,y) of size s */
    fscanf(fpmeshin, "%d %d %d ", &elem, &x, &y, &s);
    getx[elem] = x;
    gety[elem] = y;
    array[SQUARESIZE+x][SQUARESIZE+y] = s;
}

/* SPLIT ACTIVE ELEMENTS*/

while (fscanf(fperror, "%d ",&elem) != EOF)
{
    /* Split square with lower left hand corner (x,y) */

    x = getx[elem];
    y = gety[elem];

    /* The following 5 lines are commented since they don't seem*/
    /* to work
    if (x==1)
{
    printf("ERROR: element to be subdivided does not exist");
    exit(1);
}
*/

s = array[SQUARESIZE+x][SQUARESIZE+y];
array[SQUARESIZE+x][SQUARESIZE+y] = s/2;
array[SQUARESIZE+x+s/2][SQUARESIZE+y] = s/2;
array[SQUARESIZE+x][SQUARESIZE+y+s/2] = s/2;
array[SQUARESIZE+x+s/2][SQUARESIZE+y+s/2] = s/2;
core[x][y] = 1;
core[x+s/2][y] = 1;
core[x][y+s/2] = 1;
core[x+s/2][y+s/2] = 1;
ElemNum +=3;
}

/* SPLIT BUFFER ELEMENTS*/

/* "s" below is the smallest subdivision so it should be fine */
for (i=SQUARESIZE;i<2*SQUARESIZE;i+=s)
    for (j=SQUARESIZE;j<2*SQUARESIZE;j+=s)
        if ( (core[i-s-SQUARESIZE][j-s-SQUARESIZE] == 1) ||
            (core[i-s-SQUARESIZE][j+SQUARESIZE] == 1) ||
            (core[i-SQUARESIZE][j-s-SQUARESIZE] == 1) ||
            (core[i-SQUARESIZE][j+SQUARESIZE] == 1) ||
            (core[i+s-SQUARESIZE][j-s-SQUARESIZE] == 1) ||
            (core[i+s-SQUARESIZE][j+SQUARESIZE] == 1) ||
            (core[i+s-SQUARESIZE][j-s-SQUARESIZE] == 1) )
        {
            if ( array[i][j] > s )
            {
                fprintf(stderr,}
"Input Subdivision not consistent with the Bufferat %d %d\n"
, i,j);
exit(0);
}
else if ( array[i][j] == s)
{
    array[i][j] = s/2;
    array[i+s/2][j] = s/2;
    array[i][j+s/2] = s/2;
    array[i+s/2][j+s/2] = s/2;
    ElemNum +=3;
}

/*@ TRASFORM INTEGER INTO REAL DOMAIN */
/*@ AND START TO CREATE THE FINITE/GP DATA FILE */

xslope = XRANGE/SQUARESIZE;
/*@Below: Divide squaresize by two because in x
direction we have two squares ...*/
yslope = YRANGE/(SQUARESIZE/2);
constraint=0;
fprintf(fpnode,"Babuska, Adaptive Refinement\\n");
fprintf(fpnode,"%5d%5d%5d%5d%5d\\n",4,nodesNum,ElemNum,4,1,nseg);

/*@ REAL COORDINATES DATA OUTPUT*/

VertNum=1;
for (i=SQUARESIZE;i<2*SQUARESIZE;i+=s/2)
    for (j=SQUARESIZE;j<2*SQUARESIZE;j+=s/2)
        if (array[i][j] > -1)
        {
            t = array[i][j];
            if (label[i][j] <0)
            {

label[i][j] = VertNum;
rx = xslope*(i-SQUARESIZE)+XZERO;
ry = yslope*(j-SQUARESIZE)+YZERO;
fprintf(fpnode,"%10d%10d%10.4f%10.4f\n", VertNum,constraint,rx,ry);
    VertNum++;
}

if (label[i+t][j] <0)
{
    label[i+t][j] = VertNum;
    rx = xslope*(i+t-SQUARESIZE)+XZERO;
    ry = yslope*(j-SQUARESIZE)+YZERO;
    fprintf(fpnode,"%10d%10d%10.4f%10.4f\n", VertNum,constraint,rx,ry);
    VertNum++;
}

if (label[i+t][j+t] <0)
{
    label[i+t][j+t] = VertNum;
    rx = xslope*(i+t-SQUARESIZE)+XZERO;
    ry = yslope*(j+t-SQUARESIZE)+YZERO;
    fprintf(fpnode,"%10d%10d%10.4f%10.4f\n", VertNum,constraint,rx,ry);
    VertNum++;
}

if (label[i][j+t] <0)
{
    label[i][j+t] = VertNum;
    rx = xslope*(i-SQUARESIZE)+XZERO;
    ry = yslope*(j+t-SQUARESIZE)+YZERO;
    fprintf(fpnode,"%10d%10d%10.4f%10.4f\n", VertNum,constraint,rx,ry);
    VertNum++;
}
```c
/* CREATE THE NEXT LEVEL INPUT TO THIS PROGRAM */
/* AT THE SAME TIME CREATE THE TOPOLOGY FOR FINITE/GP */

fprintf(fpnewin,"%d \n", ElemNum);
ElemNum=1;
for (i=SQUARESIZE;i<2*SQUARESIZE;i+=s/2)
    for (j=SQUARESIZE;j<2*SQUARESIZE;j+=s/2)
        if (array[i][j] > -1)
{
    t=array[i][j];
fprintf(fpnode,"%5d%5d%5d%5d%5d%5d\n", ElemNum,MATERIAL,
label[i][j],label[i+t][j],
label[i+t][j+t],label[i][j+t]);
fprintf(fpnewin,"%d %d %d %d \n", ElemNum,i-SQUARESIZE,
j-SQUARESIZE,t);
ElemNum++;
if (DUMP) fprintf(fpdisp1,"%d %d %d i\n", i, j,
    i+t, j);
if (DEBUG && (t/2)>0) printf("label[i+t/2][j]= %d \n ",label[i+t/2][j] ) );
/* If t/2=0 that means that t=1 i.e. lowest level
, so no more subdiv */
if (label[i+t/2][j] > -1 && (t/2)>0 )
{
    fprintf(fpedge,
"%5d%5d%5d%5d%5d%5d%10.4f%10.4f%10.4f%10.4f\n"
 , label[i][j], 1, label[i+t][j],
 1, label[i+t/2][j], 1, 1.0, 1.0, -2.0, 0.0);
}
if (DUMP) fprintf(fpdisp1,"%d %d %d i\n", i+t, j,
i+t, j+t);
if (label[i+t][j+t/2] > -1 && (t/2)>0)
{
    fprintf(fpedge,
```
"%5d%5d%5d%5d%5d%5d%10.4f%10.4f%10.4f%10.4f\n"
  , label[i+t][j], 1, label[i+t][j+t],
  1, label[i+t][j+t/2], 1, 1.0, 1.0, -2.0, 0.0);

} if (DUMP) fprintf(fpindisl1,"%d %d %d 1\n", i+t,
   j+t, i, j+t);
if (label[i+t/2][j+t] > -1 && (t/2)>0)
{
  fprintf(fpedge,
  "%5d%5d%5d%5d%5d%5d%10.4f%10.4f%10.4f%10.4f\n"
  ,label[i+t][j+t],1,label[i][j+t],1,
  label[i+t/2][j+t],1, 1.0, 1.0, -2.0, 0.0);
}
if (DUMP) fprintf(fpindisl1,"%d %d %d 1\n", i, j+t,
   i, j);
if (label[i][j+t/2] > -1 && (t/2)>0)
{
  fprintf(fpedge,
  "%5d%5d%5d%5d%5d%5d%10.4f%10.4f%10.4f%10.4f\n"
  ,label[i][j+t],1,label[i][j],1,
  label[i][j+t/2], 1, 1.0, 1.0, -2.0, 0.0);
}
}

/* CREATE FLUX AND TEMPERATURE DATA FOR FINITE/GP */

if (DEBUG)printf("s/2=%d",s/2);
for (i=SQUARESIZE;i<2*SQUARESIZE;i+=s/2)
  for (j=SQUARESIZE;j<2*SQUARESIZE;j+=s/2)
    if (array[i][j] > -1)
{
  if (DEBUG)printf("i=%d  j=%d",i,j);
  t = array[i][j];
  right=i+t;
  up = j+t;
if (right == 2*SQUARESIZE) /* side bc flux comp*/
{
    rx = xslope*(i+t-SQUARESIZE)+XZERO;
    ry1 = yslope*(j-SQUARESIZE)+YZERO;
    ry2 = yslope*(j+t-SQUARESIZE)+YZERO;
    grad(rx,ry1,&fluxX1,&fluxY1);
    grad(rx,ry2,&fluxX2,&fluxY2);
    fprintf(fpflux,"%5d%5d%5d\n", 1, label[i+t][j], label[i+t][j+t]);
    fprintf(fpflux," %15.7e %15.7e %15.7e %15.7e\n", fluxX1,0.0,fluxX2,0.0);
}

if (up == (SQUARESIZE*3)/2) /* This is for side dc*/
{
    ry = yslope*(j+t-SQUARESIZE)+YZERO;
    rx1 = xslope*(i-SQUARESIZE)+XZERO;
    rx2 = xslope*(i+t-SQUARESIZE)+XZERO;
    grad(rx1,ry,&fluxX1,&fluxY1);
    grad(rx2,ry,&fluxX2,&fluxY2);
    fprintf(fpflux,"%5d%5d%5d\n", 1, label[i][j+t], label[i+t][j+t]);
    fprintf(fpflux," %15.7e %15.7e %15.7e %15.7e\n", fluxY1,0.0,fluxY2,0.0);
}

if (i == SQUARESIZE) /* This is for side ed*/
{
    rx = xslope*(i-SQUARESIZE)+XZERO;
    ry1 = yslope*(j-SQUARESIZE)+YZERO;
    ry2 = yslope*(j+t-SQUARESIZE)+YZERO;
    grad(rx,ry1,&fluxX1,&fluxY1);
    grad(rx,ry2,&fluxX2,&fluxY2);
    fprintf(fpflux,"%5d%5d%5d\n", 1, label[i][j], label[i][j+t]);
fprintf(fpflux, " %15.7e %15.7e %15.7e %15.7e\n", fluxX1,0.0,-fluxX2,0.0);

/*Above: Flux is pos to the right; That means that I
need the negative of flux to make it positive
going in */
}

/* This is for side ab constraint no 1*/
if (j == SQUARESIZE && i >= ((SQUARESIZE*3)/2))
{
    ry = yslope*(j-SQUARESIZE)+YZERO;
    rx1 = xslope*(i-SQUARESIZE)+XZERO;
    rx2 = xslope*(i+t-SQUARESIZE)+XZERO;
    if (i==((SQUARESIZE*3)/2))
        fprintf(fpconstr,"%5d%5d%10.3f\n", label[i][j],1,
0.0);
        fprintf(fpconstr,"%5d%5d%10.3f\n", label[i+t][j],1,
0.0);
}

/* FINITO LA MUSIKA PASATO LA FIESTA */

fprintf(stderr, "all done\n");
}        /* main */
Appendix C

Sample Input Files to program "ref"

C.1 Sample Input File "meshin1"

32
1 0 0 16
2 0 16 16

.........
......... Some element data deleted to save space
.........
31 112 32 16
32 112 48 16

C.2 Sample Input File "error-file1"

9
10
13
14
17
18
21
22
Appendix D

Sample Output Files from program “ref”

D.1 Sample Output File “meshin2”

86
1 0 0 16
2 0 16 16

........

........ (Some data has been deleted to save space)

........

85 112 32 16
86 112 48 16

D.2 Sample Output File “error-file2”

25
31
32
38
39
44
45
51
52

D.3 Sample Output File “babu2.dat”

Babuska, Adaptive Refinement

4 108 86 4 1 16

1 1 -1.0000 0.0000
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>-0.7500</td>
<td>0.0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some coordinate data removed to save space

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>0</td>
<td>1.0000</td>
</tr>
<tr>
<td>108</td>
<td>0</td>
<td>1.0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Some topology data removed to save space

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>83</td>
<td>1</td>
<td>98</td>
<td>104</td>
<td>105</td>
<td>100</td>
</tr>
<tr>
<td>84</td>
<td>1</td>
<td>100</td>
<td>105</td>
<td>106</td>
<td>102</td>
</tr>
<tr>
<td>85</td>
<td>1</td>
<td>102</td>
<td>106</td>
<td>107</td>
<td>96</td>
</tr>
<tr>
<td>86</td>
<td>1</td>
<td>96</td>
<td>107</td>
<td>108</td>
<td>97</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5037700e-02</td>
<td>0.0000000e+00</td>
<td>3.4252098e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4252098e-02</td>
<td>0.0000000e+00</td>
<td>3.2250173e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2250173e-02</td>
<td>0.0000000e+00</td>
<td>2.9730474e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1275031e-02</td>
<td>0.0000000e+00</td>
<td>1.4015080e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.9730474e-02</td>
<td>0.0000000e+00</td>
<td>2.7220333e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4015080e-02</td>
<td>0.0000000e+00</td>
<td>1.7420947e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7420947e-02</td>
<td>0.0000000e+00</td>
<td>2.1238286e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>37</td>
<td>52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1238286e-02</td>
<td>0.0000000e+00</td>
<td>2.4775395e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>52</td>
<td>67</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4775395e-02</td>
<td>0.0000000e+00</td>
<td>2.7201496e-02</td>
<td>0.0000000e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>67</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.7201496e-02  0.00000000e+00  2.8187685e-02  0.00000000e+00  1  82  97
2.8187685e-02  0.00000000e+00  2.8030160e-02  0.00000000e+00  1  104  105
0.00000000e+00  0.00000000e+00  -4.2166259e-03  0.00000000e+00  1  105  106
-4.2166259e-03  0.00000000e+00  -7.6132331e-03  0.00000000e+00  1  106  107
-7.6132331e-03  0.00000000e+00  -9.9101581e-03  0.00000000e+00  1  107  108
-9.9101581e-03  0.00000000e+00  -1.1275031e-02  0.00000000e+00  1  97  108
2.8030160e-02  0.00000000e+00  2.7220333e-02  0.00000000e+00  53  1  0.000
59  1  0.000
68  1  0.000
74  1  0.000
83  1  0.000
89  1  0.000
98  1  0.000
104  1  0.000
2  1  3  1  13  1  1.0000  1.0000  -2.0000  0.0000
3  1  5  1  16  1  1.0000  1.0000  -2.0000  0.0000
5  1  7  1  19  1  1.0000  1.0000  -2.0000  0.0000
7  1  21  1  20  1  1.0000  1.0000  -2.0000  0.0000
21  1  36  1  35  1  1.0000  1.0000  -2.0000  0.0000
36  1  51  1  50  1  1.0000  1.0000  -2.0000  0.0000
51  1  66  1  65  1  1.0000  1.0000  -2.0000  0.0000
66  1  81  1  80  1  1.0000  1.0000  -2.0000  0.0000
81  1  96  1  95  1  1.0000  1.0000  -2.0000  0.0000
100  1  98  1  99  1  1.0000  1.0000  -2.0000  0.0000
102  1  100  1  101  1  1.0000  1.0000  -2.0000  0.0000
96  1  102  1  103  1  1.0000  1.0000  -2.0000  0.0000
00001  00000001.  00000001.
Appendix E

Subroutine "error.f"

SUBROUTINE ERROR (ne,n,ng,nelfre,ndfree,
    1     nodes,lnode,index,dd,d,m,ave,nspl,
    2     ns,nn,nm,nq,h,x,job,negne)

******************************************************************************

   c
   c.... PURPOSE:
   c        -------
   c        This subroutine was tailored to be part of FINITE/GP
   c        program. It provides a global error estimator as well as
   c        element-wise error indicators. It creates the error estimate
   c        by using the norm of the difference
   c        between the discontinuous and the continuous gradient
   c        function of the solution.
   c        (For stress analysis, stress measures are used instead
   c        of gradients)
   c
   c.... More information about this subroutine can be found in
   c        its author's Stelios K Kyriacou thesis of Spring 1991
   c        (third Chapter)
   c

******************************************************************************

   c
   c.... VARIABLES DEFINITIONS
   c                -----------
   c.... (Definitions of the arguments to the subroutine are explained
   c        in the main program FINITE/GP)
   c
   c.... elAve(n,ns) = Matrix containing the values of the nodal average
gradients for an element at the nodes.

b(ns,nelfre) = Matrix containing the values of the global
derivative B (or D*B for stress problems) of the
shape functions at a quadrature points

h(n) = Array containing the values of the shape functions
at a quadrature point
eHat(ns) = Array containing the values of the discontinuous
gradient function at a quad. point (etymology: see
thesis)
eStar(ns) = Array containing the values of the continuous gradient
function at a quad. point (etymology: see theory)
eE1(ns) = The vector of the difference between eStar and eHat
(etyymology: Error in EElement)
eNorm(ne) = The L2 stress or gradient error norm for each element
gNorm = Summation variable
sysNor= System norm. The gradient L2 norm as a root of the sum
of the squares of the L2 gradient norms of all elements

k = Dummy variable
ns = Number of strains per node, ex ey gamma
nspl = Number of strains plus 1 for the Von Mises
iflag(ne) = Denotes whether the element error
is bigger (iflag=1) than the "percentCutoff"
or smaller (iflag=0)
i_error = Dummy to be sent to movie
iadd = dummy vector sent to movie.f
errave= (same as above)
sum, sum1, sum2 = Temporary holders for summations
detWt = The product of the determinant of the Jacobian
times the quadrature weight at a quadrature point
percentCutoff = The % error considered the limit for
acceptability
cutoff = The actual (not % relative) cutoff error
sysErr = Used as an intermediate and as the final value for
c the SYSystem (total) ERRor
avElNo = AVerage EElement stress NOrm (taken
by the square root of the mean of the sum of the squared L2
gradient norm and L2 gradient error norm for all the elements)
c.... correction = whether to use Zienkiewicz's correction factors
c or not

c.... ElPercE = Element PERCentage Error
c.... Change_elem = Used only in case of mixed elements
c.... Negne = Same as above
c ZeroNum = Numerical zero

c.... COMMENTS, BUGS OR OTHER PROBLEMS IN FUTURE WORK
-----------------------------------------------
c.... I am using variables like eNorm for different purposes like
c summing variables etc.
c.... I am using a mixture of capital and small letters although
c to the compiler they are all the same. I do it for easier
c identification of the variables
c.... Area calculations were not needed any more and were
c removed Nov/14/90
c.... Using nFake instead of n etc. is confusing and should be
c cleaned up
c
c.... COMMONS, DIMENSION AND PARAMETER STATEMENTS
-----------------------------------------------
c
common /iounit/ ncrd,nprt,nbug,ntap1,ntap2,ntap3,ntap4,
tap5,ntap6

CDP implicit real*8 (a-h,o-z)
dimension dd(ndfree), d(nelfre), nodes(ne,n),
1 lnode(n), index(nelfre)
dimension ave(m,nsp1), h(n)
c.... * Below nmax is 17 for a 3-D quadratic element
c.... * nmax is 7 for a 3-D element(7=6+ 1 for Von Mises)
c.... * nelfremax = 17 * 17 nodes
parameter (nmax = 17 ,nmax = 7, nelfremax=34, nspacemax=3)
parameter (nmax = 10000, nmax=3000, i_error=0)
dimension elAve (nmax,nmax),b(nmax,nelfremax),
1 eHat (nmax), eStar (nmax), eEl (nmax)
dimension eNorm(nmax),iflag(nmax)
dimension iadd(mmax),errave(mmax)
logical change_elem, correction, debug
parameter (ntap40=40, ntap50=50, ntap51=51)
parameter (zeroNum = 1.0e-20, percentCutoff=10.0
1 , debug= .true. )

c

c.... INITIALIZING
---

c.... * Make sure max limits for array dimensions are not violated
if ( (n.gt.nmax).or.(ns.gt.nmax).or.(nelfre.gt.nelfremax)
1 .or.(nspace.gt.nspacemax) .or. (ne.gt.nemax)
2 .or.(m.gt.mmax) )then
   write(*,*), 'FATAL STOP IN ERROR.F',
   STOP
endif

c

c.... * Initialize summation variables
sysNor = 0.0
sysErr = 0.0

c
if ( debug ) write( ntap50,'(a60,/)')
1  "This is the error debug file"

c.... * Rewind the file with data written in isopar
 rewind ntap6

c

c.... LOOP OVER ELEMENTS TO FIND THE ERROR MEASURES
---

do 50 ie=1,ne
   if ( debug ) write( ntap50,'(a25,i5,/)')
1  "We are in element #",ie

c

c.... EXTRACT THE ELEMENT SOLUTION VALUES D(NELFRE)
---
call lnodes (ie,ne,n,nodes,Lnode)

* To accomodate different elements (triangles &
* quadrilaterals) in the same model create fake n and nelfre
* (fake since they apply to the current element only)
nfak=n
nelfak= nelfre
if ( negne.eq.1 .and. n.eq.4 ) then
    CALL ELTYPE ( lnode,nfak,ncfak,mnfak,nqfak,nelfak,
                  change_elem,ng,ie)
endif

c
* Extract the indexes of the DOF's corresponding to the
c * nodes of this element
call indexel (nfak,nelfak,ng,lnode,index)
c
* Extract the current DOF's
call elfre (ndfree,nelfak,d,dd,index)
c
c EXTRACT THE NODAL AVERAGES ELAVE(N,NS)
c
----------------------------------
do 60 j=1,ns
    call elfre (m,nfak,elAve(1,j),ave(1,j),lnode)
 60 continue
c
INITIALIZE SUMMATION VARIABLES
AND READ THE NUMBER OF QUADR. POINTS
----------------------------------
eNorm(ie)=0.0
gNorm=0.0
read(ntap6) nip
c
LOOP OVER QUADRATURE POINTS (NUMERICAL INTEGRATION)
c
* Integrals evaluated in this loop: the gradient
* error norm, and the gradient norm integrals.
do 70 ip=1,nip
    if ( debug ) write( ntap50,'(a25,i5,/)')
1   "We are in quad point ",ip

c

REVIEW THE REQUIRED DATA AT THE CURRENT QUADR.

POINT: H, B, DETWT

-----------------------------------------------

read(ntap6) ( h(i),i=1,nfak)
read(ntap6) (( b(ins,i),ins=1,ns),i=1,nelfak)
read(ntap6) detWt

C

CALC. ONE COMPONENT OF THE ERROR AND GRADIENT

NORMS FOR EVERY STRESS COMPONENT J

-----------------------------------------------

do 80 j = 1,ns
    eStar(j) = 0.0
    eHat(j) = 0.0

c

GET DOT PRODUCT EHAT(j) = B(j) . D FOR HEAT

OR DEB(j) . D FOR STRESS

-----------------------------------------------

do 90 in=1,nelfak
    eHat(j) = eHat(j) + b(j, in) * d(in)
90 continue

c

GET DOT PRODUCT: ESTAR(j) = ELAVE(j) * H

-----------------------------------------------

do 91 in=1,nfak
    eStar(j) = eStar(j) + elAve(in,j) * h(in)
91 continue

c

SUBTRACT EHAT(J) FORM ESTAR(J) TO GET THE ERROR EEJ(J) FOR THE STRESS COMPONENT J

-----------------------------------------------

eEl(j) = eStar(j) - eHat(j)
if ( debug ) write( ntap50,* )"eStar is ",eStar(j)
if ( debug ) write( ntap50,* )"eHat is ",eHat(j)

80 continue

C..... FIND THE DOT PRODUCTS SUM = EEL . EEL AND
C..... SUM2 = ESTAR . ESTAR
C

sum = 0.0
sum2 = 0.0
do 100 j=1,ns
  sum2 = sum2 + eStar(j)*eStar(j)
  sum = sum + eEl(j)*eEl(j)
100 continue

C..... NUMERICAL INTEGRATION SUMMATIONS OVER THE
C ELEMENT (IE) FOR THE ERROR
C NORM ( E NORM(IE) ) AND GRADIENT NORM (G NORM)
C

eNorm(ie) = eNorm(ie) + sum *detWt
  gNorm = gNorm + sum2 * detWt
70 continue

C
C SUMMATION OF ELEMENT ERROR AND GRADIENT NORMS
C TO GET GLOBAL MEASURES SYSERR AND SYSNOR
C

sysErr = sysErr + eNorm(ie)
sysNor = sysNor + gNorm

C
C FINALLY THE VALUE OF ERROR NORM ENORM(IE) FOR
C THE CURRENT ELEMENT (IE) IS THE SQUARE
C ROOT OF THE PREVIOUS VALUE:
C

  eNorm(ie) = sqrt(eNorm(ie))
50 continue

C
c * To detect debugging errors:
if ((sysNor + sysErr).ne.0.0) then

PERCENTAGE SYSTEM ERROR IS 100 TIMES THE SQUARE ROOT
OF THE RATIO OF SYSTEM ERROR OVER THE SUM OF
SYSTEM NORM PLUS SYSTEM ERROR

sysPercErr = sqrt( sysErr / (sysNor + sysErr) )*100
else
  GOTO 140
endif

c.... EMPIRICAL CORRELATION (CORRECTION) FACTOR ( AS SUGGESTED
C BY ZIENKIEWICZ) IF REQUIRED

  correction = .false.
  if ( correction .eq. .true.) then
  c.... * This is 1.1 for bilinear, 1.3 for lin. triangles,
c.... * 1.6 for biquadratic and 1.4 for quad. triangles
    if (nfak.eq.4 .and. ng.eq.2 )then
      factor=1.1
    else if (nfak.eq.3)then
      factor=1.3
    else if (nfak.eq.8 .and.ng.eq.2 )then
      factor=1.6
    else if (nfak.eq.6 .and.ng.eq.2 )then
      factor=1.4
    else
      factor=1
  endif
  sysPercErr = factor * sysPercErr
endif

c.... FINDING AN RMS VALUE FOR THE ELEMENT
C GRADIENT NORM (AVELNO)
AND A CUTOFF VALUE FOR THE TRUE ERROR

-------------------------------

avElNo = sqrt( (sysNor + sysErr)/m )
cutoff = (percentCutoff / 100.0) * avElNo

FINALLY THE VALUE OF SYSTEM ERROR NORM SYSERR AND OF
THE GRADIENT NORM SYSNOR IS THE SQUARE
ROOT OF THE PREVIOUS VALUES

------------------------------

sysErr = sqrt(sysErr)
sysNor = sqrt(sysNor)
if ( debug ) then
   write(nbug,*)"sysPercErr, cutoff, sysErr,",
   " sysno, avElNo:"
   write(nbug,*)sysPercErr,cutoff,sysErr, sysNor,avElNo
endif

c.
c.... PART OF CAEDS INTERFACE IF REQUIRED

------------------------------

iCaedsErr = 1
if (iCaedsErr.eq.1) then
   write(10,"(i6)" ) -1
   write(10,"(i6)" ) 56
   write(10,"(CH'NONE')")
   write(10,"(CH'NONE')")
   write(10,"(CH'ERROR')")
   write(10,"(CH'NONE')")
   write(10,"(CH'NONE')")
   write(10,"(CH6i10')") 1,1,1,24,2,1
   write(10,"(CH3i10')") 1,1,1
   write(10,"(Ch13.5')") 0.0
endif

c.
c    * Debugging Statements

140 if (sysNor.lt.zeroNum) then
write(npnt,*)'SysNor is equal to zero (gradient is ",
1    "everywhere zero)'
write(npnt,*)'Error has to be zero everywhere or ",
1    "something is wrong'
STOP
endif
c
c.... LOOP OVER EACH ELEMENT FOR SOME FINAL

c ELEMENTWISE CALCULATIONS

c ------------------------------------------
do 150 ie=1,ne
c
c.... FIND THE RELATIVE % ERROR NORM FOR EACH ELEMENT

c ------------------------------------------
elPerce = (eNorm(ie)/avElNo)*100
c.... * Setting the flag iflag = 1 if high error in the element
if (eNorm(ie).gt. cutoff ) then
    write(npnt,*)"The element ",ie," needs ",
1       "refinement. % error est. is ", elPerce
    iflag(ie) = 1
else
    write(npnt,*)'The % error estim. (eNorm) ',
1       'for el ",ie," is ', elPerce
    iflag(ie) = 0
endif
c
c.... PART OF CAEDS ERROR INTERFACE IF

c REQUIRED - ONLY PYRAMIDS
C ARE IMPLEMENTED FOR THE TIME BEING ????:
c ------------------------------------------
if (iCaedsErr.eq.1) WRITE(i0,("(i10,/,e13.5)"
1        ie,1,eNorm(ie)
c
if ( ntap40.ne.0 ) write(ntap40,*)eNorm(ie)
if ( debug .and. nbug.ne.0 )
write(nbug,*),'eNorm(ie) = ',eNorm(ie)

IF REQUIRED PRODUCE A FILE WITH
THE HIGH ERROR ELEMENTS

                    ---
if (ntap51.ne.0 .and. iflag(ie).eq.1)
  write(ntap51,*), ie
  continue

if (iCaedsErr.eq.1) WRITE(10,'(i10)') -1

CALL MOVIE TO CREATE A FILE FOR VIEWING THE
ELEMENTS USING MOVIE.BYU

                    ---
CALL MOVIE (n, m, ne, x, nodes, dd ,eNorm,lnode,iadd,
errave,ave,nspl,job,i_error,nspace)

RETURN
END
Appendix F

Subroutine "movie.f"

SUBROUTINE MOVIE (n,m,ne, x, nodes ,dd ,enorm,lnode,iadd, 1  
erave,ave,nspl,job,i_case,nspace)

C******************************************************************************
C  PURPOSE:
C  To create data files to be fed to MOVIE.BYU for pre- and post-processing. The files are:
C  1) fort.60 (temperature or other primary unknown),
C  2) fort.61 (error),
C  3) fort.62 (coordinates and topology), and
C  4) fort.63 (Von mises)
C  It currently works only for 2D meshes
C  See the 87 v6.0 (unbound) MOVIE.BYU manual pages 139 & 123 for explanations of the format for the data files
C******************************************************************************
C
C  COMMENTS
C
C  n*ne= nedge = total number of edges (typically the number of nodes per element times the number of elements)
C  i= np = the number of parts (often 1 for fem)
C  m = the number of nodes or joints (nj)
C  ne = the number of elements
C  nspl = the parts array (for each part, the first number is the first element in the part, and the second number is the last element in the part)
C  x = the coordinates of the nodes (if 2d, write in 0 for third coordinate direction)
C  nodes = the connectivity of the elements

95
important: connectivity can apparently be of
any length, but the last number for each
element must be negative.

DIMENSION STATEMENTS ETC.

parameter (i_geom=1, nmax=10000)
dimension npl(2), x(m,nspc),dd(m),nodes(ne,n)
dimension enorm(ne),lnode(n),iadd(m),errave(m),ave(m,nsp1)
dimension n_f_a(nmax)

INITIALIZE

.... Make sure max limits for array dimensions are not violated
if (ne.gt.nmax) then
    write(*,*) 'FATAL STOP IN MOVIE.F'
    STOP
endif
nedge= n*ne
np=1
iunit=62
npl(1)=1
npl(2)=ne

GEOMETRY AND TOPOLOGY FILE

if (i_case.eq.i_geom) then
    do 10 i=1,ne
        .... Take care of more than one elem types
        if ( nodes(i,n).eq.0 ) then
            n_f_a(i)= 3
        else
            n_f_a(i)= n
        endif
        nodes(i,n_f_a(i) ) = - nodes(i,n_f_a(i) )
   10 continue
10        continue

c.....    geometry
        write(iunit,240) np,m,ne,nedge
        write(iunit,240) (npl(i),i=1,2)
        if ( nspace.eq.2 ) then
            write(iunit,220) ( (x(i,j),j=1,2),0.0 ,i=1,m)
        elseif ( nspace.eq.3 ) then
            write(iunit,220) ( ( x(i,j),j=1,nspace ) ,i=1,m)
        endif


c.....    ???? --> either (panel data files - np is positive)


c.....    If it is a quadratic we need the connectivity in a
        slightly different form

        if ( (n.eq.8).or. (n.eq.6) ) then
            write(iunit,230) ( ( nodes(i,j), nodes(i,n/2+j),
                j=1,n/2 ), i=1,ne)
        else if ( (n.eq.16).or. (n.eq.17) ) then
            write(iunit,230) ( ( nodes(i,j), nodes(i,n/4+j),nodes
                + (i,2*n/4+j),nodes(i,3*n/4+j), j=1,n/4 ), i=1,ne)
        else
            write(iunit,230) ( (nodes(i,j), j=1,n_f_a(i) ), i=1,ne)
        endif

        c    returning the array nodes to its original form
        do 11 i=1,ne
            nodes(i,n_f_a(i) ) = - nodes(i,n_f_a(i) )
11       continue

        else

        c    VON-MISES STRESS FILE

        c

        if ( job.lt.4 .or. job.eq.6 .or. job.ge.9 )
1       write(63,90) (ave(i,nspl),i=1,m)
TEMPERATURE OR OTHER PRIMARY UNKNOWN

if (job.eq.4 .or. job.eq.5)
    write(60,90) (dd(i),i=1,m)
90  format(6e12.5)

averaging the error for outputing to movie

c... zeroing the arrays
do 100  i=1,m
    errave(i)=0.0
100  iadd(i)=0

c... loop over elements to add the element error to the nodes
around it
do 200  ie=1,ne
    call lnodes (ie,ne,n,nodes,lnode)
c... loop over nodes of the element
do 300  k=1,n
    j=lnode(k)
    iadd(j)=iadd(j)+1
    errave(j)=errave(j)+enorm(ie)
300  continue
200  continue

do 400  i=1,m
    if (iadd(i).eq.0) goto 400
    errave(i)=errave(i)/float(iadd(i) )
400  continue

c
AVERAGED ERROR

c
write(61,90) (errave(i),i=1,m)
endif

c
220 format (1p6e12.5)
230 format (10i8)
240 format (10i8)
    end
Appendix G

Subroutine "caeds.f"

SUBROUTINE CAEDS (m,n,ne,nspace,x,nodes)

C******************************************************************************
C PURPOSE:
C To create a file fort.10 with coordinate, topology, data that can
C be fed to caeds by FINITE/GP
C It works only for tetrahedra (linear) and
C linear quadrilaterals.
C If it is given the wrong elements it responds
C an error message.
C The same file is appended with other types
C of data for CAEDS from other subroutines, it
C gets error data in error.f, and it gets von-mises
C stress data in post.f and postel.f.
C
C COMMENTS
C
C Complete explanations for the data format of the produced
C file exist on page 36-1 of
C " CAEDS Base for IBM RT User's Guide, Volume 1"
C
C******************************************************************************
C VARIABLES
C
C.... Igdx= (Integer) fe Graphical Descriptor id for element
C with "X" nodes (x=4_3d means 3-dim. 4-noded element)
c.... Idx = (Integer) fe Descriptor id for element with "X" nodes
c
c.... i_color1=1 blue, and i_color2=6 (green) = color of nodes
   and elements respectively

c.... i_phys and i_mat = physical and material properties table

c
   PARAMETERS ETC

c
   parameter ( i_delimiter= -1 , i_zero= 0,i_color1=1,
       i_phys=1, i_mat= 1,i_color2=6, iNodeCode=15,
       iTopolCode=71)

   parameter (Igd4_3d=14, Id4_3d= 111, Igd4=5,Id4=44)
   dimension x(m,nspace),nodes(ne,n)

c
   c.... BEGIN CREATION OF UNIVERSAL FILE

c
   write(10,'(i6)')i_delimiter
   write(10,'(i6)')iNodeCode
   do 10 im=1,m
       write(10,100)im,i_zero,i_zero,i_color1,( x(im,j),j=1,nspace)
   10 continue
   write(10,'(i6)')i_delimiter
   write(10,'(i6)')i_delimiter
   write(10,'(i6)')iTopolCode
   do 20 ie=1,ne
       if (nspace.eq.3) then
           if (n.eq.4) then
               write(10,'(7i10)')ie, Igd4_3d, Id4_3d, i_phys, i_mat,
               i_color2, n
             else
               write(*,*:"ERROR FROM caeds.f : element not",
               "implemented"
               STOP
           endif
         else
if (n.eq.4) then
    write(10,'(7i10)')ie, Igd4, Id4, i_phys, i_mat,
    \quad i_color2, n
$ else
    write(*,*)"ERROR FROM caeds.f : element not",
    \quad " implemented"
$       STOP
       endif
       endif

       write(10,'(8i10)')nodes(ie,1),nodes(ie,2),nodes(ie,3)
       ,nodes(ie,4)
20   continue
        write(10,'(i6)')i_delimiter
100  format(4i10,1p3e13.5)

   end
Appendix H

Subroutines "preasb.f", "intbin.f", and "int_sort.f"

SUBROUTINE PREASB (ja, ia, ncoef, mm, maxNR, maxNode, ne, n, nodes, 
$               lnode, m, max_neigh, ng, ndfree, max_coef)

C************************************************************************
C.... PURPOSE: To be used with finite/GP to create vectors JA
C and IA that are needed for storing the subscripts of the
C non-zero elements of the stiffness matrix
C
C.... VARIABLES:
C
C.... maxNR = MAX Neibghors in Reality in contrast to:
C.... max_neigh = the set parameter for the number of neighboring
C elements
C************************************************************************
C DIMENSION STATEMENTS
C
C  dimension ja(max_coef), ia(ndfree+1), mm(max_neigh,m),
$     nodes(ne,n), lnode(n)
C
C.... CREATE "NEIGHBOURS MATRIX" MM
C
C ASSUME that mm(1,j)=0 for every j in 1 to m
C     do 50 i = 1, ne
C.... ASSUME one kind of element type only
C     CALL LNODES (i, ne, n, nodes, lnode )
C     CALL INT_SORT (n, lnode )
C.... For each lnode(j) (its column in matrix mm) of element i do:
C     do 40 j = 1, n
i_col = lnode(j)
c.... To reduce searching since lnodes and mm are sorted now,
c we will use last_l
    last_l = 1
c.... For each influence node k of el i starting with current
c node i_col do:
        do 30 k = j, n
c
c.... Isize is the true number of non zeros in each row of mm
    isize = mm (1,i_col) + 1

c.... To avoid overflowing of a column into next one check:
    if ( isize .ge. max_neigh ) then
        write(*,*) "ERROR IN PREAD.F: Increase max_neigh"

        "$
        ," parameter in finite.f (God knows to what)"
        STOP
    endif

c.... For mm position 1 in the current column (node) do:
        do 20 l = last_l+1, isize+1

c.... mm is sorted in ascending order so:
c.... First check if the new neighbor already exists in mm: go
c to next neighbor

c.... Then Check whether this is the right position to place it
c in mm:
c.... Place it there and push the rest column one position to
c the right

c.... Then Check if current position is the last in the column
c and place it in next position
    if ( L .gt. isize ) then
        mm (isize+ 1 , i_col ) = lnode (k)
        mm (1,i_col) = mm (1,i_col) + 1
        last_l = L
    elseif ( lnode(k) .eq. mm(1,i_col) ) then
        goto 30
    elseif ( lnode(k) .lt. mm(1,i_col) ) then
        mm(1,i_col)= mm(1,i_col) + 1
isize = mm(1,i_col) + 1
do 10 ii= 0, (isize - l - 1 )
    mm(isize-ii, i_col) = mm(isize- ii -1 , i_col)
10   continue
    mm(L,i_col) = lnode(k)
last_l = L
goto 30
endif
20   continue
30   continue
40   continue
50   continue

c c   FIND THE REAL # OF MAX NEIGHBORS FOR STATISTICAL PURPOSES
maxNR = 0
do 11 iCol = 1,m
   if (mm(1,iCol) .eq. 0 ) then
      mm(1,iCol) = 1
      mm(2,iCol) = iCol
   endif
   iTemp = mm(1,iCol)+1
   if ( iTemp .gt. maxNR ) then
      maxNode = iCol
      maxNR = iTemp
   endif
11   continue

c c.... CREATE JA AND IA FROM "MM"
c

c c.... The following code is complicated by the inclusion of ng (dof
c per node > 1) 
nextRow= 1
ia(nextRow)= 1
indexJa= 1
do 90 i= 1, m
   nextRow= (i-1) * ng + 2
lenRow = mm(i,i) * ng
ia(nextRow) = ia(nextRow-1) + lenRow
if ( ia(nextRow) .gt. max_coef ) then
   write(*,*)"ERROR: need to increase max_coef"
   STOP
endif

   do 70 j = 1, mm(i,i)
      do 60 jj = 1, ng
         ja(indexJa) = (mm(j+1,i) - 1) * ng + jj
         indexJa = indexJa + 1
      continue
   60
   continue

70 continue

   if (ng .gt. 1) then
      do 80 ii = 2, ng
         nextRow = nextRow + 1
         lenRow = lenRow - 1
         ia(nextRow) = ia(nextRow-1) + lenRow
         do 71 jj = 1, lenRow
            ja(indexJa) = ja(indexJa - lenRow)
            indexJa = indexJa + 1
         continue
      71
      continue
   endif

80 continue

ndfree = ia(ndfree + 1) - 1

c...

OUTPUT FOR DEBUGGING

c

if (ioutput .eq. 1) then
   write(*,*)" This is the "mm": 

c... For each node do:
   do 91 i = 1, m
      write(*,*)"#,i",(mm(j,i), j = 1, mm(1,i)+1)
   continue

91
write(*,*) " This is JA and IA : "
do 92 i= 1, ndfree
   write(*,*)"ROW : " , i," AND ia= " , ia(i)
   write(*,*) (ja( ia(i)+ j -1 ) , j=1,( ia(i+1) - ia(i) ) )
92 continue
write(*,*)"LAST IA = NOCOEF+1= " , ia(ndfree+1)
endif
end

The following subroutine was adapted from Mojena and Ageloff [25, page 237].

SUBROUTINE INTBIN ( iPos, iVal, nums, lenTrue )
c******************************************************************************
c PURPOSE:
c IntBin (INTeger BINary) performs binary search for
c 'iVal' of ascending (sorted) array 'nums' of length 'lenTrue'
c Returns the position 'iPos' of the number 'iVal' if found
c in array else returns zero
c******************************************************************************
dimension nums(lenTrue)
len = lenTrue
left= 1
iR= len
c
c.... Loop to find the number
10 continue
   mid = (left + iR ) / 2
   if ( iVal .lt. nums(mid) ) then
      iR = mid -1
   else
      left = mid + 1
   endif
   if ( iVal.ne. nums(mid) .and. left.le.iR )
$ \text{GOTO 10}

c.... End of while loop
    if ( iVal .eq. nums(mid) ) then
        iPos = mid
    else
        iPos = 0
    endif
end

SUBROUTINE INT_SORT (size, array)
c************************************************************************************
c  PURPOSE:
c.... Insertion Sort
c.... Sorts "array" of size "size" in ascending order
c
c************************************************************************************
    integer size
    integer array(size),min, insert
    integer j,k,place

c.... Sort the first two numbers
    min = array(1)
    place = 1
    do 100 k=2,size
        if ( array(k) .lt. min ) then
            min= array(k)
            place = k
        endif
    100 continue
    array(place) = array(1)
    array(1) = min

c
c.... Proceed with the rest of the numbers
    do 300 k=3,size
        insert = array(k)
j=k

c

c.... Check the current number against the last one sorted etc.

200  if (insert .lt. array(j-1) ) then

       array(j) = array(j-1)

       j= j - 1

       goto 200

       endif

       array(j) = insert

300  continue

end
Appendix I

Subroutine “newdat.f”

SUBROUTINE NEWDAT (m, ne, n, nnmax, iopt, ntolpt, nodes, nnow, lwas, lnode, iunit)

*******************************************************************************

PURPOSE:

This subroutine is part of redfin.f program and it generates a new coordinate list and topology file output custom-made for FINITE/GP. It works as follows: First the data to be fed to it have to be in FINITE/GP format. They have to be in a file called bone.unred.dat.2. This name comes from UNREDuced DATa for the BONE problem, data part 2, which was the specific problem that the subroutine was created for although it should work for any other problem. Output can be the following depending on the value of iopt:

(iopt = output options)

0 - new element order with new nodes numbering
1 - new element order with old nodes numbering
2 - old element order with new nodes numbering

The new topology goes into file bone.dat.3 while the new coordinates go into bone.dat.2.

VARIABLES

for most variables definition see subroutine lreseq.f

iunit = unit to write new data (not needed any more?)

110
parameter ( material = 1 )
dimension ntolpt(ne+1), nodes(nnmax), nnow(m),
1
lwas(ne), lnode(m)
c

c.... Change the node numbers in the coordinates list
open(unit=10, file="bone.dat.2")
open(unit=11, file="bone.dat.3")
open(unit=12, file="bone.unred.dat.2")
do 21 i=1,m
    read(12,'(2i10,3f10.6)')ii,ij,x1,x2,x3
    write(10,'(2i10,3f10.6)') NNOW(ii),ij,x1,x2,x3
21    continue

c
c define all arrays
c--> loop over elements
do 10 l=1,ne
    if ( iopt .lt. 2 ) then
        lout = lwas(l)
    else
        lout = 1
    endif

c locate original nodes on element lout
k1 = ntolpt(lout) - 1
ksize = ntolpt(lout+1) - ntolpt(lout)
c--> loop over original nodes of element
do 20 k = 1,ksize
    nout = nodes(k1+k)
    if ( iopt .ne. 1 ) nout = nnow(nout)
    lnode(k) = nout
20    continue

c output the element number and its topology
write ( 11, '(6i5)' ) l, material, lnode
10 continue
close ( 10 )
close ( 11 )
close ( 12 )

RETURN
END
Appendix J

Subroutines “shp16q.f” and “der16q.f”

SUBROUTINE SHP16Q (R, S, H)
C
*****************************************************************************
C  SHAPE FUNCTIONS FOR SERENDIPITY QUAD WITH 16 NODES
C
*****************************************************************************
PARAMETER ( PT667 = 0.666666666666667 )
DIMENSION H(16)
C
R, S = LOCAL COORDS OF PT  4--15--11---7---3
C
H = ELEM SHAPE FUNCTIONS I I
C
8  14
C
LNODE = ELEM INCIDENCES LIST I S I
C
ELEMENT SKETCH TO RIGHT 12 +R 10
C
I I
C
10(-1,-1) 30(+1,+1) 16  6
C
I I
C
1---5---9---13---2

RR = R*R
SS = S*S
RS = R*S
RP = 1. + R
RM = 1. - R
SP = 1. + S
SM = 1. - S
RSM = R*SM
RSP = R*SP
SSM = S*SM
H(5) = -PT667*RSM*RM*RP*(1.-2.*R)
H(9) = 0.5*RM*RP*(1.-4.*RR)*SM
H(13) = PT667*RSM*RM*RP*(1.+2.*R)
H(2)  = RP*SM*(R*(4.*RR-1.) - S*(4.*SS-1.)-3.)/12.
H(6)  = -PT667*S*RP*SM*SP*(1.-2.*S)
H(10) = 0.5*SM*SP*(1.-4.*SS)*RP
H(14) = PT667*S*RP*SM*SP*(1.+2.*S)
H(3)  = RP*SP*(R*(4.*RR-1.) + S*(4.*SS-1.)-3.)/12.
H(7)  = PT667*RSP*RM*RP*(1.+2.*R)
H(11) = 0.5* RM*RP*(1.-4.*RR)*SP
H(15) = -PT667*RSP*RM*RP*(1.-2.*R)
H(4)  = RM*SP*(-R*(4.*RR-1.) + S*(4.*SS-1.)-3.)/12.
H(8)  = PT667*SSM*SP*(1.+2.*S)*RM
H(12) = 0.5*SM*SP*(1.-4.*SS)*RM
H(16) = -PT667*S*RM*SM*SP*(1.-2.*S)
RETURN
END

SUBROUTINE DER16Q (R,S,DH)

C*******************************************************************************
C LOCAL DERIVATIVES FOR A SERENDIPITY 16 NODE QUAD
C*******************************************************************************
PARAMETER ( PT667 = 0.6666666666666667 ,
1 PT4DIV3=1.3333333333333333)
C SEE SHP16Q FOR NODE LOCATIONS
DIMENSION  DH(2,16)
RR = R*R
SS = S*S
RS = R*S
RP = 1. + R
RM = 1. - R
SP = 1. + S
SM = 1. - S
P1M2R = 1. - R - R
P1P2R = 1. + R + R
P1M2S = 1. - S - S
P1P2S = 1. + S + S
P4RRM1 = 4.*RR - 1.
P4SSM1 = 4.*SS - 1.

DH(1,1) = (-SM*(-R*P4RRM1-S*P4SSM1-3.)*R*RSM*(-12.*RR+1.))/12.
DH(1,5) = -PT667*SM*RM*RP*P1M2R + PT667*R*SM*RP*P1M2R
1 - PT667*R*SM*RM*RP*P1M2R + PT4DIV3*R*SM*RM*RP
DH(1,9) = 0.5*RP*P4RRM1*SM - 0.5*RM*P4RRM1*SM
1 - 4.*R*SM*RM*RP

DH(1,13) = PT667*SM*RM*RP*P1P2R - PT667*R*SM*RP*P1P2R
1 + PT667*R*SM*RM*RP*P1P2R + PT4DIV3*R*SM*RM*RP

DH(1,12) = (SM*(R*P4RRM1-S*P4SSM1-3.)*R*RP*SM*(-12.*RR -1.))/12.

DH(1,6) = -PT667*S*SM*SP*P1M2S
DH(1,10) = -0.5*SM*SP*P4SSM1

DH(1,14) = PT667*S*SM*SP*P1P2S

DH(1,3) = (SP*(R*P4RRM1+S*P4SSM1-3.)*R*SP*SM*(12.*RR -1.))/12.

DH(1,7) = PT667*SP*RM*RP*P1P2R - PT667*R*SP*RP*P1P2R
1 + PT667*R*SP*RM*RP*P1P2R + PT4DIV3*R*SP*RM*RP

DH(1,11) = 0.5*RP*P4RRM1*SP - 0.5*RM*P4RRM1*SP
1 - 4.*R*SP*SM*RP

DH(1,15) = -PT667*SP*RM*RP*P1M2R + PT667*R*SP*RP*P1M2R
1 - PT667*R*SP*RM*RP*P1M2R + PT4DIV3*R*SP*RM*RP

DH(1,4) = (-SP*(-R*P4RRM1+S*P4SSM1-3.)*R*SM*(-12.*RR+1.))/12.

DH(1,8) = -PT667*S*SM*SP*P1P2S

DH(1,12) = 0.5*SM*SP*P4SSM1

DH(1,16) = PT667*S*SM*SP*P1M2S

DH(2,1) = (-RM*(-R*P4RRM1-S*P4SSM1-3.)*R*SM*(-12.*SS+1.))/12.

DH(2,5) = PT667*R*RM*RP*P1M2R

DH(2,9) = 0.5*RM*RP*P4RRM1

DH(2,13) = -PT667*R*RM*RP*P1P2R

DH(2,2) = (-RP*(-R*P4RRM1-S*P4SSM1-3.)*R*SM*(-12.*SS+1.))/12.

DH(2,6) = -PT667*RP*SM*SP*P1M2S + PT667*S*RP*SP*P1M2S
1 - PT667*S*RP*SM*P1M2S + PT4DIV3*S*RP*SM*SP

DH(2,10) = 0.5*SP*P4SSM1*RP - 0.5*SM*P4SSM1*RP
1 - 4.*S*RP*SM*SP

DH(2,14) = PT667*RP*SM*SP*P1P2S - PT667*S*RP*SP*P1P2S
1 + PT667*S*RP*SM*P1P2S + PT4DIV3*S*RP*SM*SP
DH(2,3) = \((RP\times(R\times P4RRM1+S\times P4SSM1-3.) + R\times P\times (12.\times SS-1.))/12.\) 
DH(2,7) = \(PT667\times R\times RM\times RP\times P1P2R\) 
DH(2,11) = \(-0.5\times RM\times RP\times P4RRM1\) 
DH(2,15) = \(-PT667\times R\times RM\times RP\times P1M2R\) 
DH(2,4) = \((RM\times(-R\times P4RRM1+S\times P4SSM1-3.) + R\times SP\times (12.\times SS-1.))/12.\) 
DH(2,8) = \(PT667\times SM\times SP\times P1P2S\times RM - PT667\times S\times SP\times P1P2S\times RM\) 
\(+ PT667\times S\times SM\times P1P2S\times RM + PT4DIV3\times S\times SM\times SP\times RM\) 
DH(2,12) = \(0.5\times SP\times P4SSM1\times RM - 0.5\times SM\times P4SSM1\times RM\) 
\(- 4.\times S\times SM\times SP\times RM\) 
DH(2,16) = \(-PT667\times RM\times SM\times SP\times P1M2S + PT667\times S\times RM\times SP\times P1M2S\) 
\(- PT667\times S\times RM\times SM\times P1M2S + PT4DIV3\times S\times SM\times SP\times RM\) 
RETURN 
END
Appendix K

Subroutine "eltype.f"

SUBROUTINE ELTYPE ( lnode,n,nc,nm,nqp,nelfak,change_elem,ng,ie)

c************************************************************************************
c PURPOSE:
c This is a "temporary" bypass of the inability of FINITE/GP
c to handle more than one type of elements in the same
c problem.
c Currently it only accomodates linear triangles & linear
c quadrilaterals
c************************************************************************************

common /elt/ nspace

c.... Outputs the new n, nc, nm, misfix(?) (nqp), nelfak (?nelfre) &
c.... change_elem
save triangle
logical triangle, triang_old, change_elem
dimension lnode(n)

c.... Do element type check only if triangle or rectangle

c.... TEMPORARY FIX
if ( nspace .eq. 3 ) RETURN
if (n.eq.4 .or. n.eq.3) then
  triang_old= triangle
  if (lnode(4).eq.0) then
    triangle = .true.
  else
    triangle= .false.
  endif

c.... change_elem=true means this element is different from

c previous
change_elem= .not. (triangle.eq.triang_old)
c.... If (not same type as prev. element) or (first element initialization): check type and define props
   if (change_elem.or. (ie.eq.1)) then
      if (triangle) then
         n=3
         nc=3
         nm=n-nc
         nqp=1
         nelfak= ng*n
      else
         Quadrilateral
         n= 4
         nc= 4
         nm= n-nc
         nqp= 2
         nelfak= ng*n
      endif
   endif
endif
endif
END