CLOSED-FORM DEVELOPMENT OF A FAMILY OF HIGHER ORDER TETRAHEDRAL ELEMENTS THROUGH THE FOURTH ORDER

by

SARA ELIZABETH MCCASLIN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2008

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to Dr. Lawrence for always responding to my questions and concerns quickly, and with patience. It has been a pleasure to work with him and learn from him. Dr. Shiakolas has provided warnings regarding problem areas in this research, and has always been willing to answer questions regarding his own work, for which I thank him. I would also like to thank Dr. Nomura for his advice regarding Mathematica, as well as guidance in how to access necessary applications through the UT Arlington network servers. Dr. Wang, Dr. Chan, and Dr. Dennis have provided helpful comments, criticisms, and guidance, which has challenged me and improved the quality of my research.

Dr. Lindsay Wells taught me how to debug programs while I was a senior at the University of Texas at Tyler; this skill has been a vital part of my education, and I wish to thank him.

I owe an incredible debt of gratitude to my mother, Lura McCaslin, who has held me up with her prayers, moral support, and refusal to allow me to feel sorry for myself. She has invested many hours in my education (whether driving me back and forth to class or listening to my endless worrying), and saying "Thank you" will never be enough.

April 18, 2008

ABSTRACT

CLOSED-FORM DEVELOPMENT OF A FAMILY OF HIGHER ORDER TETRAHEDRAL ELEMENTS THROUGH THE FOURTH ORDER

Sara Elizabeth McCaslin, PhD.

The University of Texas at Arlington, 2008

Supervising Professor: Kent Lawrence

This research is concerned with the development and implementation of a family of tetrahedral elements through the fourth order. The straight-sided tetrahedral elements are developed in closed-form. This work investigates the efficiency of closed-form implementation of stiffness matrices and error estimators compared to numerical implementation. An additional objective is the compaction of closed-form source-code files which require as little storage space as possible, a more pronounced requirement at high *p*-levels.

For the straight-sided elements through *p*-level 4, the stiffness matrix, equivalent nodal load vectors, and error estimators (based on nodal averaging) are developed using closed-form equations obtained through the use of a computer algebra

system. The stiffness matrix and error estimators are also implemented using numerical integration so that a timing comparison between the numerical and the closed-form approaches could be performed.

The curved-sided elements, including the stiffness matrix, equivalent nodal load vectors, and error estimators are also implemented using Gaussian quadrature only. A test conducted on a model of all curved-sided elements is used to verify that the elements are working correctly.

Results indicate that the closed-form implementation solutions are comparable to the numerical solutions. For all *p*-levels the closed-form stiffness matrix is more efficient by a factor of at least 4 when compared with numerically integrated elements.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF SYMBOLS

Subscripts

c	curved-sided
e	element
m	magnitude
r	radial
s	straight-sided
v	vector
θ	tangential

Nomenclature

a, b	inner and outer radii of a cylinder, respectively
A	area
A_f	area of a face
$[A]$	matrix of constants used in strain vector representation
$[B]$	strain displacement matrix
C_{ij}	entry in the inverse coefficient matrix
$[C]$	coefficient matrix
$[D]$	elasticity matrix

e	error estimate
e_i	error estimate for element i
E	modulus of elasticity
F	applied force
$[g]$	used in closed-form stiffness development
G	shear modulus
$[G]$	geometry and material dependent matrix
h	height of a beam
I	moment of inertia
$[I]$	identity matrix
$[J]$	Jacobian matrix
k	torsional stiffness constant
$[K]$	stiffness matrix
$[N]$	shape function matrix
$[N']$	shape function matrix
$[P]$	used in closed-form stiffness and error estimation development
$\{r\}$	equivalent nodal load vector
$[R]$	used in closed-form stiffness and error estimation development
L	length of a beam
L_i	volume coordinate
M	moment

p	polynomial order
P_i	i^{th} Legendre polynomial
r	radius
S	surface
T	torque
u	strain energy
u_i	strain energy for element i
u, v, w	displacements
U	strain energy
V	volume
$\vec{V}_{ij}$	vector from node i to node j
$\overrightarrow{V_u}$	normalized unit vector
w_i	weight for Gaussian or Gauss-Lobatto numerical integration
w	width of a beam
x, y, z	global coordinates

Greek Symbols

α	coefficient of thermal expansion
$[\Gamma]$	used in closed-form stiffness development
$\overline{\eta}$	permissible global error
δ	displacement

δu_i	displacement of node *i* in the *u* direction
ΔT	temperature difference
ε_m	mean permissible error
$\{\varepsilon\}$	strain vector
$\{\Phi\}$	applied load vector
η	global error
Ω	domain of integration
φ_i^j	i^{th} shape function for *p*-level *j*
ν	Poisson's ratio
σ	stress
$\{\hat{\sigma}\}$	stress vector obtained from the finite element solution
$\{\sigma^*\}$	smoothed stress vector
$\{\overline{\sigma^*}\}$	nodally averaged stress vector
τ	shear stress
ξ, η, ζ	local coordinates on a standard tetrahedral

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Historical Background

Finite element analysis is a powerful engineering tool that analyzes objects by breaking them into individual elements and nodes; the objects modeled can range from geometrically simple parts to complex systems. The basic principles of physics and engineering are applied to the individual elements and nodes, and the elements are later reassembled to provide a solution for the entire structure.

According to Zienkiewicz [1], the finite element method has two distinct lines of ancestry: discrete engineering systems and mathematical approximations. The earliest discrete approach was known as the stiffness method, and in the early 1900s was successfully used for applications such as bridge construction. In this methodology, the structure is viewed as a system of interconnecting components: displacements at the ends (nodes) of these components (elements) were assumed to be linearly related (by the stiffness of the element) to the forces applied at the ends (nodes). If the sum of the forces at each joint is assumed to be zero, equilibrium is preserved; if the displacements at the nodes are treated as the unknown, then the system of displacements is made continuous. The resulting system of equations can be used to solve for the actual displacements.

Matrices were found an efficient tool for representing the systems of equations. Relaxation methods and iterative processes were developed to speed the solution, but, prior to the advent of computer systems, only a relatively small number of unknowns could be solved for. The construction of bridges, railways, and tall buildings, along with the use of reinforced concrete in such applications, provided a further incentive for engineers to improve their methodologies for efficient solutions [2].

The mathematical approximation used in finite element analysis stems from the idea of a continuum; this approach requires the use of partial differential equations, rather than a set of discrete equations, to represent the physics involved. Mathematical methods developed included the finite-difference method, trial function, minimization of potential energy, the use of weighted residuals, and boundary solution methods.

The establishment of computer technology has allowed for the efficient and accurate solution of extremely complex models. During World War II, "secret" research in the area of relaxation led to the successful solution of 900 simultaneous equations, almost quadrupling what was possible before the war. A large model in 1960 involved two to three thousand variables; the largest model to date, solved using computerized techniques, appears to be a billion plus variables [3].

According to Zienkiewicz [1], ". . . less emphasis is being placed on research leading to more economical and efficient methods on computation." Many users of finite element analysis are accepting methodologies that are inefficient and unrefined simply because computer usage (both in terms of memory and processing power) is far cheaper than in the past. Zienkiewicz goes on to say that " . . . we should strive to show

that more refined calculation is generally preferable to the use of inefficient methods . . ."

It is possible to solve complex problems using the older methods on new computer systems, but there is still a need for more elegant, computationally efficient approaches. It should also be noted that, with improvement in speed and memory capabilities of common desktop computers, as well as the widespread availability of efficient computer algebra systems, there remain many unexplored areas where finite element research can be advanced.

1.2 Research Objectives

This project involves the development and implementation of a family of straight-sided and curved-sided tetrahedral elements through the fourth order. The research places emphasis on the efficiency of closed-form solutions, made possible by modern computer algebra systems, for development of tetrahedral stiffness matrices and nodal averaging error estimators. Research has been pursued in this area for various two dimensional elements as well as for isoparametric tetrahedral elements [4, 5, 6]; this research focuses on straight-sided elements emplyoing the Szabo and Babuska basis hierarchical shape functions [7 ,8] for *p*-levels 1 through 4, straight-sided, and straight-sided elements based on isoparametric shape functions for *p*-level 4 [9]. Both closed-form straight-sided and numerical curved-sided implementations are considered.

A straight-sided element, including its stiffness matrix, equivalent nodal load vectors, and error estimators (based on nodal averaging), is implemented using closed-form equations obtained through use of a computer algebra system, but a numerical

integration implementation is also be developed to allow for a comparison in efficiency between closed-form and numerical approaches. Both versions are implemented in FORTRAN code, tested, and results compared with theoretical values where available.

Since a significant portion of the computation expense involves the formation of the stiffness matrix and error estimator for each element in the model, the major outcome of this research involves a time study performed to compare the execution time expense of the closed-form implementation of element matrices with that of the numerical implementation of element matrices.

In addition, the equivalent curved-sided elements, including stiffness matrix, equivalent nodal load vectors, and error estimators were implemented using Gaussian quadrature. These elements are also implemented in FORTRAN code and tested.

1.3 Literature Review for Tetrahedral Elements

The simplest two-dimensional element is the triangle; its three-dimensional counterpart is the tetrahedron, whose use has become practically unavoidable in finite element modeling because it readily lends itself to the representation of complex geometries. As a more recent example of their use in unusual geometric models, note that Dennis et.al. used tetahedral elements when modeling the human head and neck for research involving cooling of the human brain to prevent a stroke after onset of cerebral ischemia [10]. The first suggestions concerning the use of a tetrahedral element are those of Gallagher et al. [11] and Melosh [12] in the early 1960s. Argyris et. al. [10] developed the TET 20 (quadratic strain) and TEA 8 elements in 1968, both possessing complete polynomials for the displacement fields and satisfying displacement

compatibility. Pawlak et al. [13] developed a four node tetrahedron with three translational and three rotational degrees of freedom in 1991, which proved to be more computationally efficient than the ten node tetrahedron, but not as accurate.

Recent research into the development of improved tetrahedral elements includes a low-order tetrahedral element created in 1994 by Key et. al. [14]: an eight-node tetrahedral comprised of a four-node tetrahedral element enriched with four mid-face nodal points for use in all-tetrahedral modeling involving wave propagation. In 1999, Kong et. al. [15] developed a new fourth-order tetrahedral element with mass lumping for solving the wave equation. Bittencourt [16] developed fully tensorial and modal shape functions for triangles and tetrahedra, which included a tensorial based Gauss-Jacobi integration procedure.

1.4 Literature Review for Shape Functions

Hierarchic families of triangular elements were developed by Katz and Rossow [17] and tetrahedral elements were presented by Babuska, Katz, and Szabo, both in 1979 [7], and revisited later by Szabo and Babuska in 1991 [8]. Main characteristics of the hierarchic shape functions include the property that basis functions of level p are a subset of the basis functions of level $p + 1$, forming a hierarchical family, and that they are composed of complete polynomials.

Hierarchical elements can be used in the *p*-method implementation of finite element analysis, where a required level of accuracy is achieved by retaining the same mesh but increasing the polynomial level used. In 1982, Babuska and Szabo found that, for quasiuniform meshes, the *p*-method cannot have a lower rate of convergence than

the *h*-method, and in some cases the *p*-method can converge twice as fast [18]. An summary of the *p*-method (as well as the *h*-*p* method) was provided by Babuska and Suri in 1990 [19], which concluded that, although the higher polynomial levels are more computationally expensive, the ratio of work to accuracy and engineering accuracy is better for the *p*-method than for the *h*-method.

Major modifications to hierarchical shape function bases would include the Carnevali basis, implementing orthogonal bases (using Gram-Schmidt orthonormalization) with better conditioning, which is of interest because conditioning can be an issue when the condition number of the stiffness matrix increases exponentially with an increase in p-level [20]; this differs from standard hierarchical elements with *p*-level 3 or greater. In 2001, bases were developed with better sparsity and conditioning properties as opposed to ill-conditioning of the Szabo-Babuska basis caused by coupling of the volume (bubble) and face shape functions. The improved properties obtained by modifying the Szabo-Babuska basis by orthogonalization, which also reduces the condition number [21].

The isoparametric tetrahedral element shape functions are well-known for *p*-levels 1 through 3, referred to, respectively, as the constant strain tetrahedron, linear strain tetrahedron, and quadratic strain tetrahedron [22]. The fourth order element, sometimes referred as the cubic strain tetrahedron, is not as well documented, but the equations for the development of its shape functions can be found in the paper by Argyris introducing the TET 20 (linear strain) element [9].

1.5 Literature Review for Stiffness Matrices

In the development of the TET 20 and TEA 8 elements, Argyris et al. [9] obtained the stiffness matrices using matrix transformations of the modal stiffness. Since that time, the use of symbolic processors has allowed for closed-form evaluation of the stiffness matrices of tetrahedral elements [5].

1.5.1 Closed-form Stiffness Matrices

Tinawi [23] used closed-form integration to obtain the stiffness matrix of non-hierarchic triangular elements in 1972; Subramanian and Bose [24, 25] developed stiffness matrices without the use of numerical approximation for the family of plane triangular elements in 1982 and for C_0 continuous tetrahedra in 1983. Closed-form expressions for plane hierarchic triangular elements were later investigated by Rathod et al. , including a recursive method for curved triangular elements [26]. In 1984, Babu and Pinder [27] obtained analytical integration formulae for linear isoparametric quadrilateral finite elements, and demonstrated a savings in computational effort when compared to Gaussian quadrature.

Nambiar [28] and Lawrence et al. [29] showed that the implementation of closed-form expressions resulted in improved speed compared to Gaussian numerical integration for both constant strain and linear strain triangular elements.

With regard to tetrahedral elements, Shiakolas et al. [4, 5, 6] developed closed-form expressions for linear and quadratic strain tetrahedral elements using Mathematica to produce the closed-form equations in FORTRAN format; the research showed

significant time savings when compared to numerical approximation using Gaussian quadrature.

1.5.2 Curved Tetrahedral Elements

For curved tetrahedral elements, matrix development requires some form of numerical integration or processing. Dey et al. [30] have recently been able to improve the efficiency numerical integration of hierarchical curved tetrahedra using integration by table look-up; the non-polynomial portions of the integrand are approximated by polynomials and a table of precomputed values is developed based on the approximating polynomials.

1.6 Literature Review for Error Estimators

Discretization errors are defined as the difference between the exact solution and the numerical solution, and result from the attempt to represent a continuum by a finite number of subdivisions. The two basic types of discretization error estimates are *a priori* and *a posteriori*. *A priori* estimates are useful for obtaining the worst case in a class of solutions of a problem but do not provide information about the actual error. Conversely, *a posteriori* methods use information obtained during the solution process, as well as some *a priori* assumptions concerning the solution.

Various adaptive strategies exist for refining the model based on the error estimation. The *h*-method reduces the size of the elements in a mesh when the local error indicator is above a previously determined error tolerance; the *p*-method increases the local order of approximation; the *r*-extension uses a fixed number of nodes which are redistributed to areas of high error in the mesh [31].

According to Zienkiewicz, there are essentially two procedures available for a posteriori error estimation today: residual error estimators and recovery based error estimators [31]. This research into error estimation was introduced by Babuska and Rheinboldt [32] and considered local residuals of the numerical solution, allowing estimation of the local errors from a norm, such as the energy norm. It allowed for adaptive refinement to be achieved by first locating the elements with the greatest error and subdividing them to improve the accuracy.

The second approach uses a recovery process to more accurately represent the unknowns. Variations in the recovery based error estimations involve the type of procedure used to substitute for the exact solution in the discretization error calculation. The recovery based methods were pioneered by Zienkiewicz and Zhu [33], and the mathematical basis for the Zienkiewicz-Zhu (ZZ) method was explained by Ainsworth et. al. [34]. This method, which is both simple and computationally efficient, is based on using a globally smoothed stress distribution to represent the exact solution. Byrd [35] developed a stress nodal averaging estimator that was implemented in ANSYS [36]. Other variations include the superconvergent patch recovery method (SPR), also developed by Zienkiewicz and Zhu [37], and the recovery by equilibrium in patches as developed by Boroomand and Ziekiewicz [38].

Note that Carstensen and Funken [39] have proven that the ZZ error indicator provides reliable upper and lower bounds for the error and thus has the property of an error estimator.

Closed-form error estimation has not been a major focus of finite element researchers. However, Nambiar in 1989 [28], Lawrence et. al. in 1991 [29], and Shiakolas et. al. in 1992 [4, 6, 40] performed research in this area, and their results have been promising in that they show closed-form implementation is more efficient in terms of speed when compared to numerical evaluation.

1.7 Literature Review for Gaussian Cubature

Fellipa has created a set of Mathematica scripts that implement the most commonly used Gauss rules for finite element applications, and these can be used "as-is" in computer algebra systems or used to generate rule values to be implemented in a program [41]. Cools and Rabinowitz produced a thorough bibliography of monomial cubature rules in 1993 [42], followed by Part 2 of the series in 1999 [43]. In 2003, Cools continued this work with an encyclopedia of cubature formulas available on the World Wide Web, which provides recomputed points and weights to either 16 digits or 32 digits, corrects misprints in the original manuscripts, and also gives access to an extensive bibliography [44].

The theory, development, methodology, and testing involved in this research are presented in the chapters that follow. The element shape functions, and modifications required for curved-sided development, are discussed in Chapter 2. Chapters 3 and 4 present the closed-form development of the stiffness matrices and error estimators, respectively, while briefly outlining the numerical implementations. Derivations of the equivalent nodal load vectors for pressure, shear, and temperature are summarized in Chapter 5. Chapter 6 deals specifically with the compaction algorithm used to reduce

the size of the Mathematica produced source code files, and Chapter 7 discusses the usage of Mathematica in this research. Chapter 8 presents the test problems used, while Computational results for the test problems, timing comparisons, and compaction are discussed in Chapter 9, which ends with conclusions and a discussion of future areas of work.

CHAPTER 2

ELEMENT SHAPE FUNCTIONS

In finite element modeling, tetrahedral elements are the three-dimensional counterpart to the two-dimensional triangle in finite element modeling. The shape of a tetrahedral element readily lends itself for use in both simple and complex models, and its usage has become very common. Tetrahedrons also lend themselves readily to automatic volume meshing.

There are different types of tetrahedral elements in practice; in the context of closed-form solutions, the work of Shiakolas [4, 5, 6] focused on isoparametric tetrahedral elements, but this research focuses on hierarchic tetrahedral elements for straight-sided elements [7, 8], and isoparametric fourth order elements [9] for curved-sided, as well as straight-sided.

Hierarchic elements differ from isoparametric elements in several ways, including their use in the *p*-version of the finite element method. In the *p*-version, the mesh is generally held fixed while the polynomial approximation, represented by degree *p*, is increased; in the *h*-version, for which the isoparametric elements are well adapted, the polynomial approximation is held fixed while the mesh is refined.

Hierarchic elements are also based on a complete set of polynomials, as illustrated in Fig. 2.1 by Pascal's pyramid through *p*-level 4. Unlike isoparametric elements where all nodal variables represent displacement, hierarchic elements include

nodal variables that are based on derivatives of displacement. The external nodal variables are used to enforce global C_0 continuity, while the internal nodal variables complete the polynomial. Note that the shape functions used are defined as integrals of Legendre polynomials, rather than having a Lagrange basis as do the isoparametric elements.

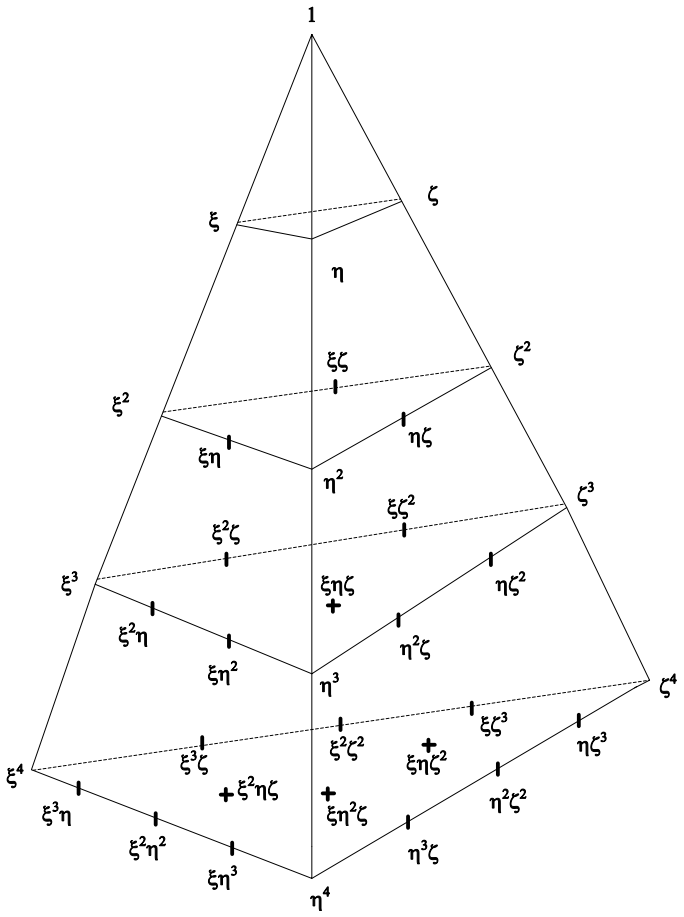


Figure 2.1 Illustration of the complete polynomials found in *p*-levels 1 through 4.

Babuska, Szabo, and Katz [7] introduced a family of hierarchic elements, used in this research, with the property that polynomial p is a subset of polynomial $p + 1$;

thus, the stiffness matrix, equivalent nodal loads, and error estimation terms posses this same property.

This chapter discusses the development of the hierarchic and isoparametric shape functions, their implementations using a computer algebra system (abbreviated CAS) for both straight-sided elements and curved-sided elements, as well as node numbering issues.

2.1 General Introduction

The shape functions for tetrahedral elements are described in terms of local coordinates, called volume or natural coordinates and indicated by *L*. The local coordinates are transformed into global coordinates by the mapping illustrated in Fig. 2.2 below.

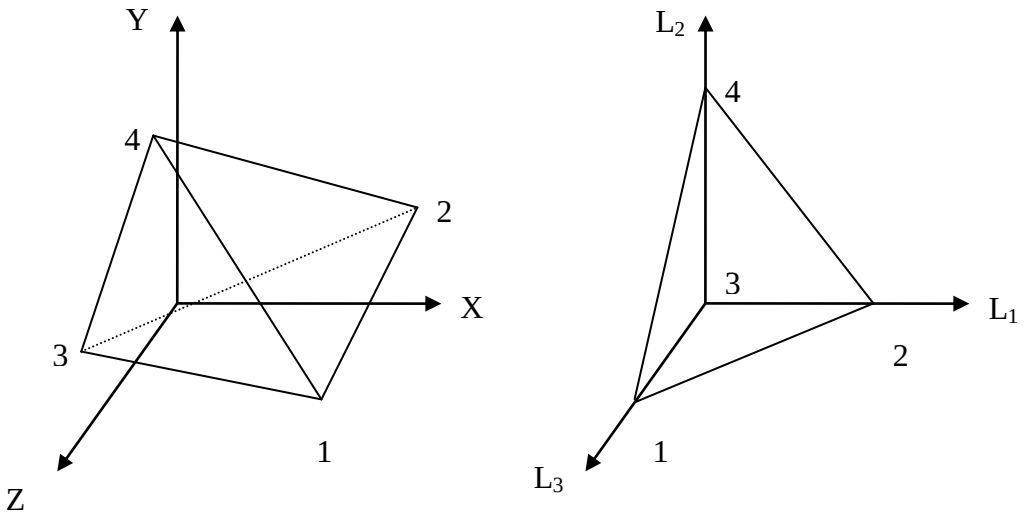


Figure 2.2 Mapping from global to local coordinates.

To map from the local coordinates to global coordinates, the transformation shown in Equation 2.2 is used. Local coordinates L_i (i=1, 2, 3, 4) are mapped to global coordinates of any point within the element represented $\{x, y, z\}$ based on the global coordinates of the vertices of the tetrahedral element $\{x_i, y_i, z_i\}$ (i=1, 2, 3, 4).

$$\begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \end{Bmatrix} \tag{2.2}$$

The local coordinates are also known as volume coordinates, note the physical nature of the volume coordinate illustrated for L_1 shown in Equation 2.3.

$$L_1 = \frac{volP234}{vol1234} \tag{2.3}$$

Volume coordinate L_i represents ratio of tetrahedral volumes based on an arbitrary internal point P inside the tetrahedron. The value of L_i is 1 at vertex i and zero on the opposing face. The sum of the volume coordinates is always 1, which indicates that they are not independent. The relationship in Equation 2.4 is used to eliminate L_4 from Equation 2.2, resulting in Equation 2.5, where $x_{ij} = x_i - x_j$, $y_{ij} = y_i - y_j$, $z_{ij} = z_i - z_j$.

$$L_4 = 1 - L_1 - L_2 - L_3 \tag{2.4}$$

$$\begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} = \begin{bmatrix} x_{14} & x_{24} & x_{34} & x_4 \\ y_{14} & y_{24} & y_{34} & y_4 \\ z_{14} & z_{24} & z_{34} & z_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ 1 \end{Bmatrix} \tag{2.5}$$

2.2 Hierarchical Shape Functions

For the hierarchical-based elements, shape functions as described by Szabo and Babuska [9] were used for *p*-levels 1 through 4. There are four nodal shape functions and three types of modes: edge, face, and internal. The edge modes are always associated with mid-side edge nodes, while the face modes are associated with the center of the face; the internal modes, or bubble nodes, are located at the centroid of the element.

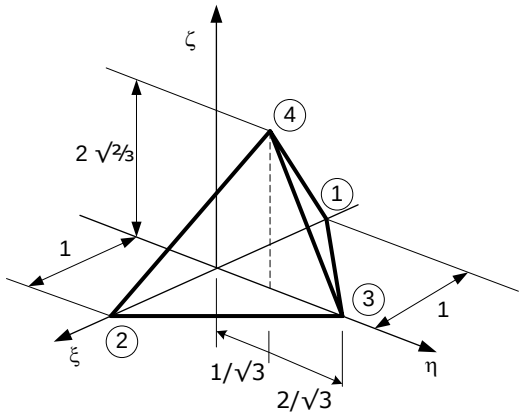


Figure 2.3 Standard tetrahedral element.

If a standard tetrahedral element is defined as shown in Fig. 2.3 above, then the volume coordinates are as follows:

$$L_1 = \frac{1}{2}\left(1 - \xi - \frac{1}{\sqrt{3}}\eta - \frac{1}{\sqrt{6}}\zeta\right) \tag{2.6}$$

$$L_2 = \frac{1}{2}\left(1 + \xi - \frac{1}{\sqrt{3}}\eta - \frac{1}{\sqrt{6}}\zeta\right) \tag{2.7}$$

$$L_3 = \frac{\sqrt{3}}{3}\left(\eta - \frac{1}{\sqrt{8}}\zeta\right) \tag{2.8}$$

$$L_4 = \sqrt{\frac{3}{8}}\zeta \tag{2.9}$$

$$L_1 + L_2 + L_3 + L_4 = 1 \tag{2.10}$$

There are four nodal shape functions, given as follows:

$$\varphi_i^1 = L_i \text{ where i = 1, 2, 3, 4} \tag{2.11}$$

The implementation used by Adjerid, Aiffa, and Flaherty [21] was found to be the clearest explanation of the development of the remaining shape functions, and is reproduced here. After defining the following two formulas, the remaining shape functions can be expressed.

$$P_i = \left[1 \quad x \quad \frac{1}{2}\left(3x^2 - 1\right) \quad \frac{1}{2}\left(5x^3 - 3x\right) \quad \frac{1}{8}\left(35x^4 - 30x^2 + 3\right) \quad \frac{1}{8}\left(63x^5 - 70x^3 + 15x\right)\right] \tag{2.12}$$

$$\mathcal{E}_k(t_1, t_2) = \frac{-8\sqrt{4k+2}}{k(k+1)} P_i'(t_2 - t_1) \tag{2.13}$$

$$\mathcal{F}_{r_1,r_2}(t_1, t_2, t_3) = P_{r_1}(t_2 - t_1) P_{r_2}(2t_3 - 1) \tag{2.14}$$

$$\mathcal{B}_{r_1,r_2,r_3}(t_1, t_2, t_3, t_4) = P_{r_1}(t_2 - t_1) P_{r_2}(2t_3 - 1) P_{r_3}(2t_4 - 1) \tag{2.15}$$

There are 6 (p-1) edge modes on each edge E_j, for j = 1, 2, 3, . . . , 6, located at the midpoint of the edge. These edge modes are given by the equation below. Note that, for implementation in this research, the values of j_1 and j_2 are first calculated and then sorted in ascending order. For example, suppose the results were 4 and 1,

respectively: in actual implementation, they would be sorted so that they would be 1 and 4 for j_1 and j_2.

$$\varphi_i^2 = L_{j1} L_{j2} \mathcal{E}_k\left(L_{j1}, L_{j2}\right) \tag{2.16}$$

where $k = 1, 2, \ldots, p-1$,

$$j_1 = \begin{cases} 1 + j \; mod \; 3, 1 \le j \le 3 \\ 1 + j \; mod \; 4, 4 \le j \le 6 \end{cases}, \text{ and } \; j_2 = \begin{cases} 1 + (j+1) \; mod \; 3, 1 \le j \le 3 \\ 4, 4 \le j \le 6 \end{cases}$$

There are 4 (p-1) (p-2) / 2 face modes, each located in the centroid of face F_j, j = 1, 2, 3, 4. These modes are given in Equation 2.13 below. Note that after j_1, j_2, and j_3 are calculated, this triplet of values is then sorted. For example, suppose the results were 1, 4, and 3, respectively: in actual implementation, they would be sorted so that they would be 1, 3 and 4 for j_1, j_2, and j_3.

$$\varphi_i^3 = L_{j1} L_{j2} L_{j3} \mathcal{F}_{r_1, r_2}\left(L_{j1}, L_{j2}, L_{j3}\right) \tag{2.17}$$

where $j = 1, 2, 3, 4$

$j_1 = 1 + \text{j} \; mod \; 4$, $j_2 = 1 + j_1 \; mod \; 4$, and $j_3 = 1 + j_2 \; mod \; 4$

$k = 3, 4, \ldots, p$ and $r_1 + r_2 = k - 3$

For the region modes (or bubble nodes), there are $(p-1)(p-2)(p-3)/6$ modes located at the centroid of the element. Their existence starts in p-level 4, where there is one such mode. The equation for this bubble mode is given as follows, for p-level 4 only.

$$\varphi_i^4 = L_1 L_2 L_3 L_4 \mathcal{B}_{r_1, r_2, r_3}(L_1, L_2, L_3, L_4) \tag{2.18}$$

where k = 4, 5, . . . , p and $r_1 + r_2 + r_3 = k - 4$

The shape functions through *p*-level 4 were derived and implemented in *Mathematica*© for use in both the closed-form and numerical solution codes. Observation of the output verified the nested nature of these shape functions, i.e., *p*-level 2 contains all the shape functions of *p*-level 1, and *p*-level 3 contains all the shape functions found in *p*-level 2. This helped to serve as verification that the shape function calculations were working as expected.

Only the nodal shape functions represent displacement: all other shape functions (edge, face, and bubble modes) represent derivatives of displacement, rather than displacement. Table 2.1 shows the node numbering, shape function, and displacement representation for each node in p-levels 1 (please reference Figure 2.2 to determine the position of nodes 1 – 4). Tables 2.2, 2.3, and 2.4 show the additional nodes found in each p-level.

The nodal numbering indicated in the tables is illustrated on a tetrahedral element in Figs. 2.4 and 2.5 for *p*-levels 1 through 4. Note that the edge nodes are all located at the middle of the edge, and the face nodes are all located at the middle of the element face.

Table 2.1 Summary of Hierarchical Shape Functions for *P*-level 1.

P-level 1 Shape functions	*Type*	*Position*	*Node #*	*Variable type*
L_1	Vertex	Node 1	1	Displacement
L_2		Node 2	2	
L_3		Node 3	3	
L_4		Node 4	4	

Table 2.2 Summary of Additional Hierarchical Shape Functions for *P*-level 2.

P-level 2 Shape functions	*Type*	*Position*	*Node #*	*Variable type*
$-4\sqrt{6}\,L_2\,L_3$	Edge	Edge 2 3	5	2^{nd} derivative
$-4\sqrt{6}\,L_1L_3$		Edge 1 3	6	
$-4\sqrt{6}\,L_1\,L_2$		Edge 1 2	7	
$-4\sqrt{6}\,L_1\,L_4$		Edge 1 4	8	
$-4\sqrt{6}\,L_2\,L_4$		Edge 2 4	9	
$-4\sqrt{6}\,L_3\,L_4$		Edge 3 4	10	

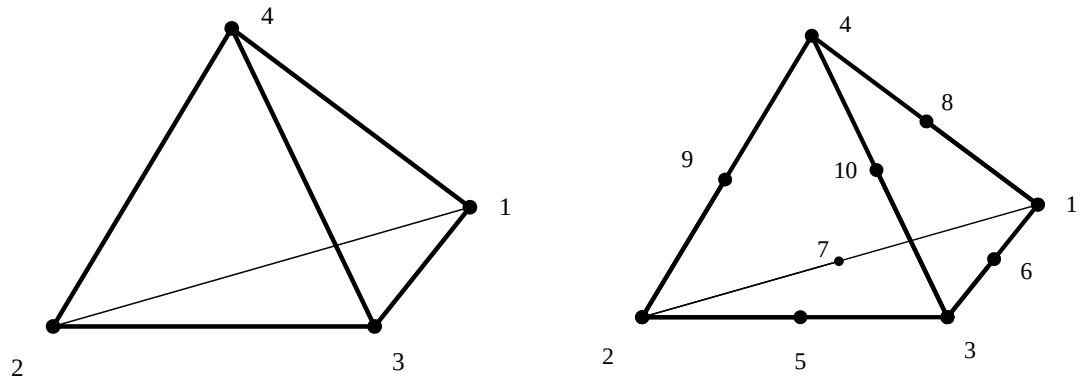


Figure 2.4 *P*-levels 1 and 2 with node ordering.

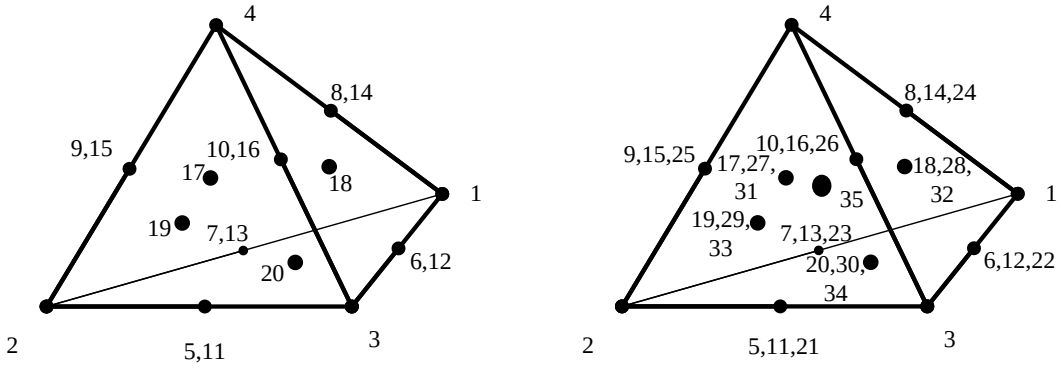


Figure 2.5 *P*-levels 3 and 4 with node ordering.

Table 2.3 Summary of Additional Hierarchical Shape Functions for *P*-level 3.

P-level 3 Shape functions	*Type*	*Position*	*Node #*	*Variable type*
$4\sqrt{10}\ L_2L_3(L_2 - L_3)$	Edge	Edge 2 3	11	3rd derivative
$4\sqrt{10}\ L_1L_3(L_1 - L_3)$		Edge 1 3	12	
$4\sqrt{10}\ L_1L_2(L_1 - L_2)$		Edge 1 2	13	
$4\sqrt{10}\ L_1L_4(L_4 - L_1)$		Edge 1 4	14	
$4\sqrt{10}\ L_2L_4(L_4 - L_2)$		Edge 2 4	15	
$4\sqrt{10}\ L_3L_4(L_4 - L_3)$		Edge 3 4	16	
$L_2\ L_3\ L_4$	Face	Face 2 3 4	17	
$L_1\ L_3\ L_4$		Face 1 3 4	18	
$L_1\ L_2\ L_4$		Face 1 2 4	19	
$L_1\ L_2\ L_3$		Face 1 2 3	20	

Table 2.4 Summary of Additional Hierarchical Shape Functions for *P*-level 4.

P-level 4 Shape functions	*Type*	*Position*	*Node #*	*Variable type*
$-\sqrt{14}/3\ L_2\ L_3\ (15(L_2 - L_3)^2 - 3)$	Edge	Edge 2 3	21	4th derivative
$-\sqrt{14}/3\ L_2\ L_3\ (15(L_2 - L_3)^2 - 3)$		Edge 1 3	22	
$-\sqrt{14}/3\ L_1\ L_2\ (15(L_1 - L_2)^2 - 3)$		Edge 1 2	23	
$-\sqrt{14}/3\ L_1\ L_4\ (15(L_1 - L_4)^2 - 3)$		Edge 1 4	24	
$-\sqrt{14}/3\ L_2\ L_4\ (15(L_2 - L_4)^2 - 3)$		Edge 2 4	25	
$-\sqrt{14}/3\ L_3\ L_4\ (15(L_3 - L_4)^2 - 3)$		Edge 3 4	26	
$L_2\ L_3\ L_4(L_2 - L_3)$	Face	Face 2 3 4	27	
$L_3\ L_4\ L_1(L_1 - L_3)$		Face 1 3 4	28	
$L_4\ L_1\ L_2(L_1 - L_2)$		Face 1 2 4	29	
$L_1\ L_2\ L_3(L_2 - L_1)$		Face 1 2 3	30	
$-L_2\ L_3\ L_4(2L_4 - 1)$		Face 2 3 4	31	
$L_3\ L_4\ L_1(2L_1 - 1)$		Face 1 3 4	32	
$L_4\ L_1\ L_2(2L_4 - 1)$		Face 1 2 4	33	
$L_1\ L_2\ L_3(2L_3 - 1)$		Face 1 2 3	34	
$L_1\ L_2\ L_3\ L_4$	Bubble	Centroid	35	

Shiakolas [4, 5] developed closed-form isoparametric tetrahedrons for *p*-levels 1 through 3. That work is here extended by considering the fourth order isoparametric tetrahedron. The fourth order isoparametric tetrahedral shape function is not commonly documented in the literature, although Argyris et. al. [9] provided an interpolation scheme for direct determination of modal functions. Like the fourth order hierarchical elements, each element has four corner nodes, three nodes per edge, three nodes per face, and one internal node.

For a fourth order element (p = 4), Eqns. 2.19 through 2.23 below can be used to derive all 35 shape functions [9].

$$i + j + g + h = p \tag{2.19}$$

$$\zeta_1^k = {}^{i}/_{p}\,, \zeta_2^k = {}^{j}/_{p}\,, \zeta_3^k = {}^{g}/_{p}\,, \zeta_4^k = {}^{h}/_{p} \tag{2.20}$$

$$\omega^k = f_1^i f_2^j f_3^g f_4^h \tag{2.21}$$

$$f_v^l = c_v^l \prod_{L=0}^{l-1} \left(\zeta_v - \zeta_v^L\right) \tag{2.22}$$

$$\zeta_v^L = {}^{L}/_{p}, c_v^l = {}^{p^l}/_{l!} \tag{2.23}$$

Using the above formulation, a summary of each type of shape functions included in the fourth order is provided below in Eqns. 2.24 – 2.28, while the thirty-five shape functions, their positions in the master element, and nodal assignment is summarized in Table 2.5.

The shape functions for the corner nodes are given by Eq. 2.24.

Table 2.5 Summary of Shape Functions, Positions, and Nodal Assignments for 4th Order Isoparametric Elements

Node	*Shape Function*	*Type*	*Position*
1	(32*(-3/4 + L1)*(-1/2 + L1)*(-1/4 + L1)*L1)/3	L1	Node 1
2	(32*(-3/4 + L2)*(-1/2 + L2)*(-1/4 + L2)*L2)/3	L2	Node 2
3	(32*(-3/4 + L3)*(-1/2 + L3)*(-1/4 + L3)*L3)/3	L3	Node 3
4	(32*(-3/4 + L4)*(-1/2 + L4)*(-1/4 + L4)*L4)/3	L4	Node 4
5	(128*(-1/2 + L1)*(-1/4 + L1)*L1*L2)/3	L1 - L2	Closer to L1
6	(128*L1*(-1/2 + L2)*(-1/4 + L2)*L2)/3	L1 - L2	Closer to L2
7	64*(-1/4 + L1)*L1*(-1/4 + L2)*L2	L1 - L2	Middle
8	(128*(-1/2 + L2)*(-1/4 + L2)*L2*L3)/3	L2 - L3	Closer to L2
9	(128*L2*(-1/2 + L3)*(-1/4 + L3)*L3)/3	L2 - L3	Closer to L3
10	64*(-1/4 + L2)*L2*(-1/4 + L3)*L3	L2 - L3	Middle
11	(128*(-1/2 + L1)*(-1/4 + L1)*L1*L3)/3	L1 - L3	Closer to L1
12	(128*L1*(-1/2 + L3)*(-1/4 + L3)*L3)/3	L1 - L3	Closer to L3
13	64*(-1/4 + L1)*L1*(-1/4 + L3)*L3	L1 - L3	Middle
14	(128*(-1/2 + L3)*(-1/4 + L3)*L3*L4)/3	L3 - L4	Closer to L3
15	(128*L3*(-1/2 + L4)*(-1/4 + L4)*L4)/3	L3 - L4	Closer to L4
16	64*(-1/4 + L3)*L3*(-1/4 + L4)*L4	L3 - L4	Middle
17	(128*L1*(-1/2 + L4)*(-1/4 + L4)*L4)/3	L4 - L1	Closer to L4
18	(128*(-1/2 + L1)*(-1/4 + L1)*L1*L4)/3	L4 - L1	Closer to L1
19	64*(-1/4 + L1)*L1*(-1/4 + L4)*L4	L4 - L1	Middle
20	(128*L2*(-1/2 + L4)*(-1/4 + L4)*L4)/3	L4 - L2	Closer to L4
21	(128*(-1/2 + L2)*(-1/4 + L2)*L2*L4)/3	L4 - L2	Closer to L2
22	64*(-1/4 + L2)*L2*(-1/4 + L4)*L4	L4 - L2	Middle
23	128*(-1/4 + L1)*L1*L2*L3	L1 - L2 - L3	Closer to L1
24	128*L1*(-1/4 + L2)*L2*L3	L1 - L2 - L3	Closer to L2
25	128*L1*L2*(-1/4 + L3)*L3	L1 - L2 - L3	Closer to L3
26	128*(-1/4 + L2)*L2*L3*L4	L2 - L3 - L4	Closer to L2
27	128*L2*(-1/4 + L3)*L3*L4	L2 - L3 - L4	Closer to L3
28	128*L2*L3*(-1/4 + L4)*L4	L2 - L3 - L4	Closer to L4
29	128*(-1/4 + L1)*L1*L2*L4	L1 - L2 - L4	Closer to L1
30	128*L1*(-1/4 + L2)*L2*L4	L1 - L2 - L4	Closer to L2
31	128*L1*L2*(-1/4 + L4)*L4	L1 - L2 - L4	Closer to L4
32	128*(-1/4 + L1)*L1*L3*L4	L1 - L3 - L4	Closer to L1
33	128*L1*(-1/4 + L3)*L3*L4	L1 - L3 - L4	Closer to L3
34	128*L1*L3*(-1/4 + L4)*L4	L1 - L3 - L4	Closer to L4
35	256*L1*L2*L3*L4	L1 - L2 - L3 - L4	Centroid

$$\varphi_i^1 = {}^{32}/_3 \left(L_i - \tfrac{3}{4}\right)\left(L_i - \tfrac{1}{2}\right)\left(L_i - \tfrac{1}{4}\right) L_i \text{ where i = 1, 2, 3, 4} \quad (2.24)$$

Edge modes consist of three nodes per edge: one in the middle, and nodes ¼ distance from each endpoint. Edge mode functions for the nodes ¼ edge length from corner node *i* are shown in Eq. 2.25.

$$\varphi_i^2 = {}^{128}/_3 \left(L_i - \tfrac{1}{2}\right)\left(L_i - \tfrac{1}{4}\right) L_i L_j \quad (2.25)$$

Edge mode functions for mid-edge nodes between corners *i* and *j* are found in Eq. 2.26.

$$\varphi_i^2 = 64 \left(L_i - \tfrac{1}{4}\right)\left(L_j - \tfrac{1}{4}\right) L_i L_j \quad (2.26)$$

There are three nodes per face, with each node near a vertex. Face mode functions nearest vertex *i* on a face composed of *i*, *j*, and *k* are found in Eq. 2.27.

$$\varphi_i^3 = 128 \left(L_i - \tfrac{1}{4}\right) L_i L_j L_k \quad (2.27)$$

There is one internal mode located at the centroid of the element. Its equation is given below.

$$\varphi_i^4 = 256 L_1 L_2 L_3 L_4 \quad (2.28)$$

Table 2.5 summarizes the shape functions, node assignments, and positions for the fourth order isoparametric tetrahedral element.

2.4 Summary of Curved-sided Modifications

For curved-sided elements, the distorted global Cartesian coordinates are mapped to curvilinear coordinates, as shown in Fig. 2.5. These curvilinear coordinates allow the distorted tetrahedral element to be represented by an undistorted parent element. The local curvilinear coordinates as defined as follows in Eqns. 2.29 – 2.32.

With any integration limits involved appropriately modified, the shape functions developed in this section can be modified for curvilinear applications by an appropriate substitution of variables [22].

$$\xi = L_1 \tag{2.29}$$

$$\eta = L_2 \tag{2.30}$$

$$\zeta = L_3 \tag{2.31}$$

$$1- \xi - \eta - \zeta = L_4 \tag{2.32}$$

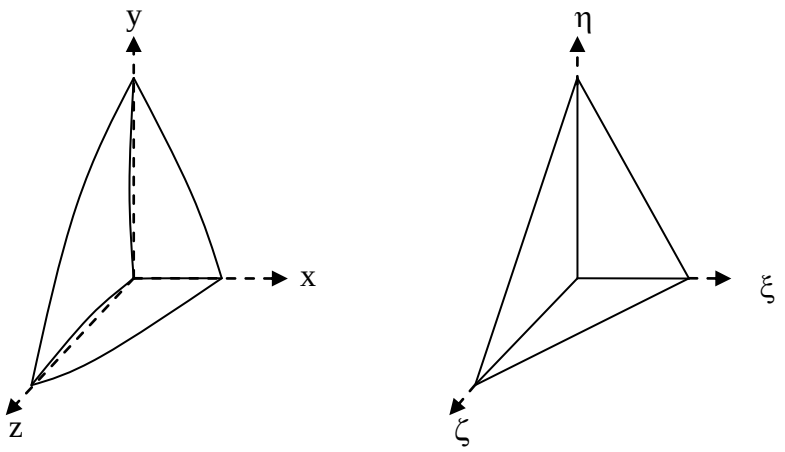


Figure 2.6 Distorted tetrahedral element mapped to curvilinear coordinates.

The process of mathematically converting the distorted element into the curvilinear element is known as “mapping,” and requires conversion from the global

coordinates *x*, *y*, *z* to the local coordinates ξ, η, ζ. This transformation requires the calculation of the Jacobian between the local and global coordinates. If $f(\xi, \eta, \zeta)$ represents a function defined in terms of the local coordinate system, use of the chain rule gives the following.

$$\frac{\partial f}{\partial \xi} = \frac{\partial f}{\partial x} \times \frac{\partial x}{\partial \xi} + \frac{\partial f}{\partial y} \times \frac{\partial y}{\partial \xi} + \frac{\partial f}{\partial z} \times \frac{\partial z}{\partial \xi} \tag{2.23}$$

If f represents nodal displacement *u*, *v*, and *w* (all dependent on the shape functions), then the Jacobian matrix $[J_c]$ of the transformation of function f can be obtained as follows

$$\begin{Bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{Bmatrix} f = [J_c] \begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{Bmatrix} f = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{Bmatrix} f \tag{2.24}$$

Because *x*, *y*, and *z* can be obtained from the shape functions in the form shown below, the partial derivatives found in the Jacobian matrix can be obtained using Eq. 2.26 [22]. Note that $N_i^{'}$ is the shape function, in terms of the local coordinates, associated with the *i*th node, x_i is the *x*-coordinate of the *i*th node, and *n* is the number of nodes per element.

$$x = \sum_{i=1}^{n} N_i^{'} x_i,\ y = \sum_{i=1}^{n} N_i^{'} y_i,\ \ z = \sum_{i=1}^{n} N_i^{'} z_i \tag{2.25}$$

$$\frac{\partial x}{\partial \xi} = \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \xi} x_i \tag{2.26}$$

This allows the Jacobian matrix to take the form shown below [22].

$$[J_c] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \xi} x_i & \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \xi} y_i & \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \xi} z_i \\ \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \eta} x_i & \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \eta} y_i & \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \eta} z_i \\ \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \zeta} x_i & \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \zeta} y_i & \sum_{i=1}^{n} \frac{\partial N_i^{'}}{\partial \zeta} z_i \end{bmatrix}$$

(2.27)

Defining finite element properties, such as stiffness and equivalent nodal loads, can be represented in the form shown below, where the matrix $[H]$ is dependent on the shape functions $[N]$ defined with respect to the global coordinates, and the integration is performed over the volume (in the case of error estimation and stiffness matrices) or over the area (in the case of equivalent nodal loads).

$$\int_V [H]\, dv \tag{2.28}$$

To transform the variables to local coordinates, the determinant of the Jacobian matrix is used as illustrated below for the volume of an element.

$$dx\, dy\, dz = det(J_c)\, d\xi\, d\eta\, d\zeta \tag{2.29}$$

Integrals for tetrahedral elements then take the following form, where the integration is carried out over the parent region, the undistorted form of the element [22].

$$\int_0^1 \int_0^{1-\eta} \int_0^{1-\eta-\zeta} [\bar{H}(\xi, \eta, \zeta)] d\xi\, d\eta\, d\zeta \tag{2.30}$$

When curved elements are used, the model is first built using all straight-sided elements. Nodes located on curved edges or surfaces are moved to the appropriate coordinates after the straight-sided model has been completed. The following methodology for moving the nodes to curved edges and faces follows the

implementation discussed by Muthukrishnan in his dissertation [45]. The coordinates are determined by first assuming that the curved surface can be represented by a conic surface, whose axis lies along the *z*-axis of the model, and whose surface can be defined by two radii and the endpoints of the cone. This is illustrated in Fig. 2.7, and the formula is given in Eq. 2.31 below where the symbols correspond to illustration referenced.

$$R_c = R_2 + \left({}^{z - z_2}/_{z_1 - z_2}\right)(R_1 - R_2) \tag{2.31}$$

After the value of R_c is known, the coordinates of the new node position can be obtained using geometry, as shown in Fig. 2.8. The required equations are shown below, and the variables again correspond to illustration referenced. These relationships are valid for moving both edge nodes and face nodes.

$$R_s = \sqrt{{x_s}^2 + {y_s}^2} \tag{2.32}$$

$$x_c = {}^{R_c}/_{R_s}\, x_s \tag{2.33}$$

$$y_c = {}^{R_c}/_{R_s}\, y_s \tag{2.34}$$

The curved-sided elements were implemented using the fourth order isoparametric implementation only. During the course of research, it was found that the hierarchic shape functions selected do not lend themselves to the type of curved implementation described above.

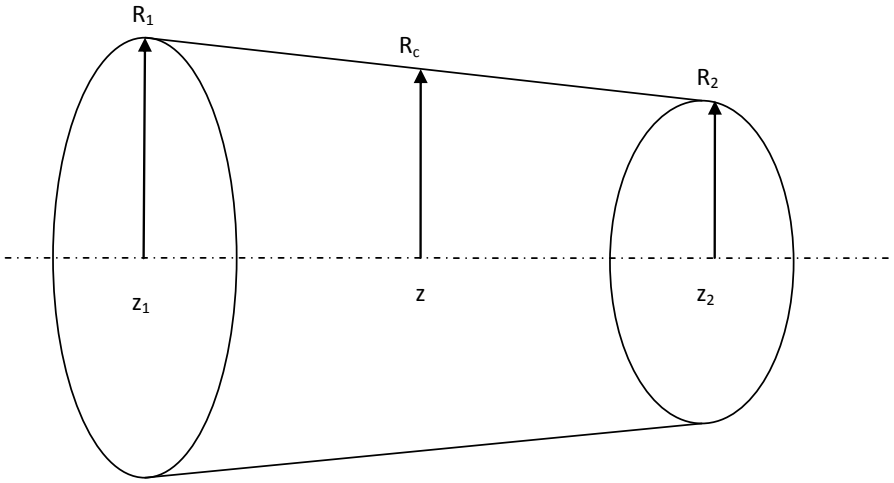


Figure 2.7 Illustration of a conic surface.

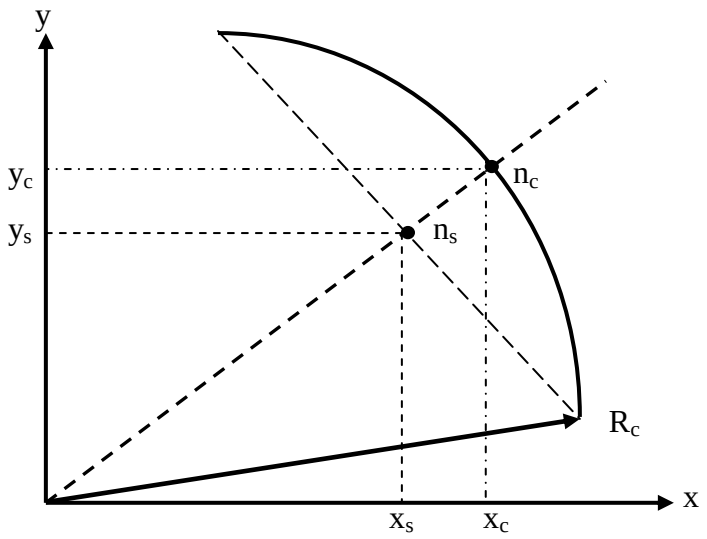


Figure 2.8 Coordinates of a node on a curved surface.

2.5 Local Node Numbering Issues

One of the problematic issues discovered during initial testing involved the sensitivity of the higher order shape functions of *p*-levels 3 and 4 to edge and face

directionality based on local node numbering. It is possible for shape functions on shared edges with opposite directionality to cancel each other out, as illustrated along an edge in Fig. 2.8 (courtesy Dr. Shiakolas in private communication). If the element on the left has edge directionality 2-3, and the element on the right has edge directionality 3-2, the shape functions could cancel each other out and produce erroneous results.

As a further illustration, the face of a tetrahedral element will be considered. Fig. 2.9 shows elements A and B which share a common edge: local nodes 2 and 3 for both elements (A2 and A3, with B2 and B3). The edge orientations are in opposite directions, and, as discussed the effects of shape functions for these edges may cancel each other out during calculations such as stiffness or error estimation.

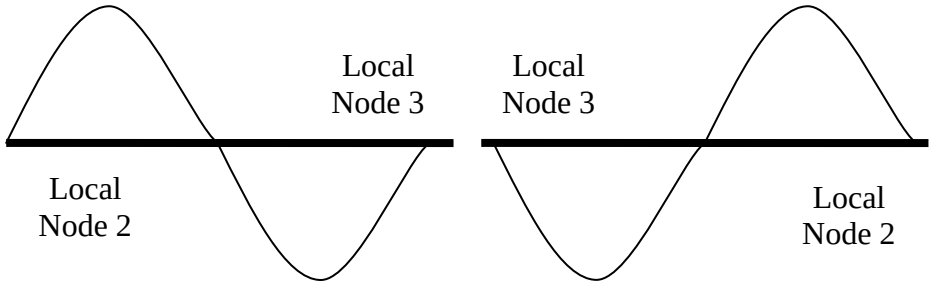


Figure 2.9 Illustration of the sensitivity of same shape functions to edge directionality.

The same issue arose in the work of Xin, Pinching, and Flaherty [46] during implementation of hierarchical simplicial elements in FEMLAB. They referred to the problem as maintaining "interelement continuity of the basis," and their solution to this problem was modified and found to be sufficient for the test problems used in connection with hierarchical elements.

1. Sort corner nodes by node number in ascending order, not taking into account their position in the coordinate system.
2. Define all edges in sorted pairs by node number.
3. Define all faces in sorted triplets by node number.
4. If, when calculating the volume of an element, that volume is found to be less than zero, swap local nodes 2 and 3 and mark that element as a "swapped" element.

If nodes 2 and 3 are the only nodes that can be out of order, then the only edge and faces that could be shared but have differing orientations would be edge 2-3 and faces that include edge 2-3. The orientation of all edges except 2-3, and all faces without edge 2-3, will match for all elements sharing them. If the order of these locals nodes is swapped during preprocessing, that element is marked as a "swapped" element; calculations such as the stiffness, stress, or error estimation for that element will be based on shape functions where the order of local nodes 2 and 3 have been reversed for edges and faces. Figures 2.10 and 2.11 illustrates how this solution works.

Xin et. al. [46] found that several approaches existed in the literature which indicated that the most straightforward approach is to produce shape functions based on permutations, but that method was not compatible with their FEM implementation, nor is it compatible with the assembly of the stiffness matrix and error estimation implementation used in this research.

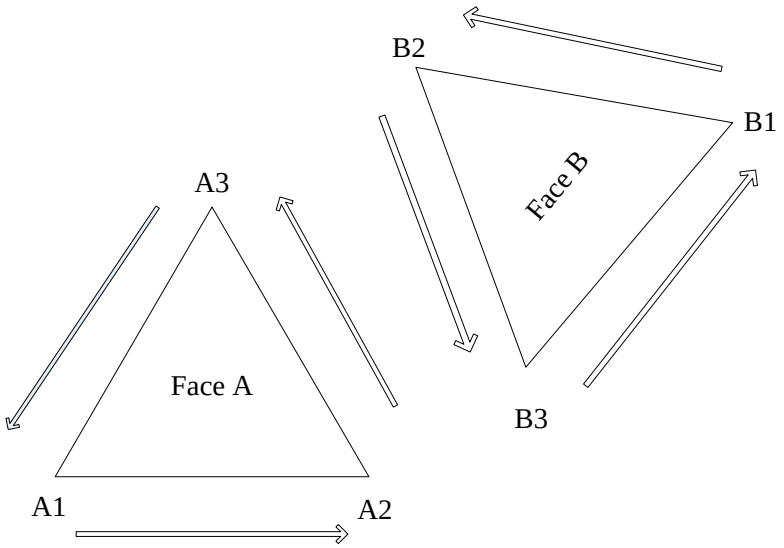


Figure 2.10 Two faces with a shared edge but with different edge orientations.

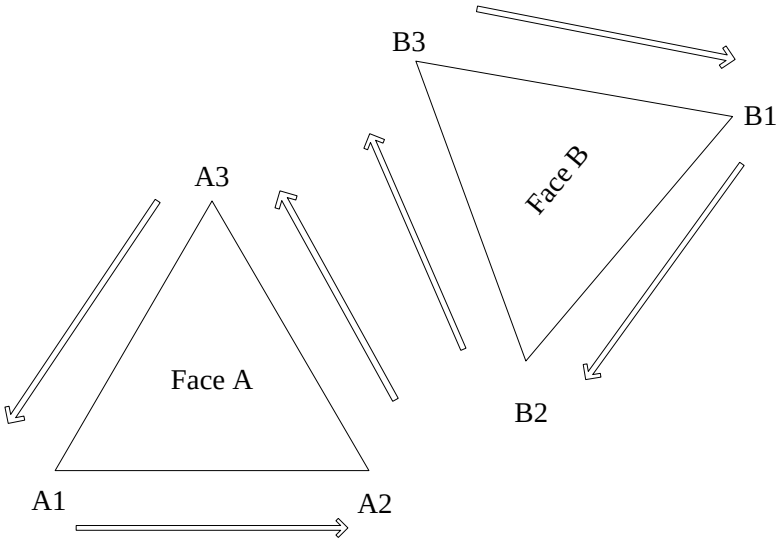


Figure 2.11 Same faces with edge and nodes swapped to ensure that the shared edge is based on the same edge orientation.

For isoparametric, similar issues were resolved by making node 1 the node with the smallest y-coordinate, node 4 the node with the largest z-coordinate. Nodes 2 and 3 are swapped as needed to preserve positive volume for the element.

Chapter 3 presents the development of the closed-form stiffness matrices, and also discusses their numerical implementation.

CHAPTER 3

STIFFNESS MATRICES

One of the major objectives of the derivation that follows is to find an expression for the stiffness matrix [*K*] that can be efficiently implemented in closed-form, not just for numerical quadrature. If the expression for the stiffness matrix can be broken into matrices, and some of those matrices can be calculated once for each type of element or once for each element, the closed-form implementation will see an increase in efficiency. Many of the equations that follow will be manipulated with this purpose in view, and follows closely the derivation provided by Shiakolas [4]. Note that this derivation is applicable for both hierarchical and isoparametric straight-sided elements.

3.1 Derivation and Manipulation of the Stiffness Matrix

The shape functions for the tetrahedral element are described in terms of a local coordinate system, whose coordinates, indicated by *L*, are called volume or natural coordinates. The equation below is used to obtain the global coordinates of any point in the element. L_i represents the local volume coordinates and x_i, y_i, and z_i represent the global Cartesian coordinates of the vertices of the tetrahedron.

$$\begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \end{Bmatrix} \tag{3.1}$$

Because the sum of the volume coordinates will always be one, it is possible to eliminate L_4: $L_4 = 1 - L_1 - L_2 - L_3$. Substitution of L_4 into Equation 3.1 results in the following equation, where $x_{ij} = x_i - x_j$, $y_{ij} = y_i - y_j$, and $z_{ij} = z_i - z_j$.

$$\begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} = \begin{bmatrix} x_{14} & x_{24} & x_{34} & x_4 \\ y_{14} & y_{24} & y_{34} & y_4 \\ z_{14} & z_{24} & z_{34} & z_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ 1 \end{Bmatrix} \tag{3.2}$$

The volume coordinates (L_1, L_2, L_3, L_4) are functions of the coordinates of the corner nodes of the element. They are obtained from the inversion of Equation 3.2 above, which yields Equation 3.3. This also results in the expressions of the form shown in Equation 3.4, where x, y, and z represent the global Cartesian coordinates of any point in the tetrahedron.

$$\begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ 1 \end{Bmatrix} = \begin{bmatrix} x_{14} & x_{24} & x_{34} & x_4 \\ y_{14} & y_{24} & y_{34} & y_4 \\ z_{14} & z_{24} & z_{34} & z_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{bmatrix} \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} \tag{3.3}$$

$$L_i = \left(C_{i1}x + C_{i2}y + C_{i3}z + C_{i4}\right)/\left|J\right| \tag{3.4}$$

Note that C_{ik} represents the entry at (i,k) in the inverse coefficient matrix (see Equation 3.3 above) and $\left|J\right|$ is the determinant of the Jacobian representing a transformation from Cartesian to volume coordinates, and is equal to 6 x *volume*.

The element shape functions and nodal displacements of an element are used to determine the displacement of any point in the element, as shown in Equation 3.5 for displacement in the *x*-direction. Variable N_i is the shape function for node *i* and δu_i is the global displacement of node *i* in the *x*-direction. Similar expressions are used to find the displacements in the *y*- and *z*- directions, using *v* and *w*, respectively.

$$u = \sum_{i=1}^{nodes} N_i \delta u_i \tag{3.5}$$

Equation 3.6 is used to find the displacement of any point in the element in all directions (not just one node in a particular direction). In this expression, $[I_3]$ is an identity matrix of order 3, *nodes* represents the total number of nodes in the element, $[N]$ is the element shape function, and δu_1 would represent the displacement of node 1 in the *x*-direction.

$$\begin{Bmatrix} u(x,y,z) \\ v(x,y,z) \\ w(x,y,z) \end{Bmatrix} = \begin{bmatrix} [I_3]N_1 & [I_3]N_2 & \dots & [I_3]N_{nodes} \end{bmatrix} \cdot \begin{Bmatrix} \delta u_1 \\ \delta v_1 \\ \delta w_1 \\ \vdots \\ \delta u_{nodes} \\ \delta v_{nodes} \\ \delta w_{nodes} \end{Bmatrix} \tag{3.6}$$

The equation below shows an example how Equation 3.6 can be used to determine global displacement by looking at the x-displacement, represented by $u(x,y,z)$.

$$u = N_1 \delta u_1 + N_2 \delta u_2 + \ldots + N_{nodes} \delta u_{nodes} \tag{3.7}$$

This allows the calculation of global displacement as a function of nodal displacement and element shape functions. The next step is to look at the strain vector, which, for a three-dimensional domain, can be represented by Equation 3.8. Note that *u*, *v*, and *w* are displacements in the *x*, *y*, and *z* directions, respectively.

$$\{\varepsilon\} = \left\{ \frac{\partial u}{\partial x} \quad \frac{\partial v}{\partial y} \quad \frac{\partial w}{\partial z} \quad \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \quad \frac{\partial w}{\partial y} + \frac{\partial v}{\partial z} \quad \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \right\}^T \tag{3.8}$$

The strain displacement matrix $[B_i]$, based on the definition of strain shown above, is given in Equation 3.9, and is applicable for any node *i*. Note that $[B_i]$ is a differential operator matrix that operates on global displacements *u*, *v*, and *w*, not on local displacements. The strain displacement matrix $[B]$ for an entire element is shown in Equation 3.10, where *nodes* represents the total number of nodes in the element; it is a function of the strain displacement matrices for each node in the element, and its dimensions are 6 x (3 x *nodes*).

$$[B_i] = \begin{bmatrix} \frac{\partial u}{\partial x} & 0 & 0 \\ 0 & \frac{\partial v}{\partial y} & 0 \\ 0 & 0 & \frac{\partial w}{\partial z} \\ \frac{\partial v}{\partial x} & \frac{\partial u}{\partial y} & 0 \\ 0 & \frac{\partial w}{\partial y} & \frac{\partial v}{\partial z} \\ \frac{\partial w}{\partial x} & 0 & \frac{\partial u}{\partial z} \end{bmatrix} \tag{3.9}$$

$$[B] = [B_1 \quad B_2 \quad \ldots \quad B_{nodes}] \tag{3.10}$$

The strain vector of Equation 3.8 can be represented in an alternate form. If $u_{,x,y,z} = \left\{ \frac{\partial}{\partial x} \quad \frac{\partial}{\partial y} \quad \frac{\partial}{\partial z} \right\}^T u$ represents the partial derivates of u with respect to x, y, and z, and matrix [A] is comprised of constants, the strain vector representation takes on the shortened form shown in Equation 3.11, where the matrix of constants is named [A].

$$\{\varepsilon\} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \left\{ \frac{\partial u}{\partial x} \quad \frac{\partial u}{\partial y} \quad \frac{\partial u}{\partial z} \quad \frac{\partial v}{\partial x} \quad \frac{\partial v}{\partial y} \quad \frac{\partial v}{\partial z} \quad \frac{\partial w}{\partial x} \quad \frac{\partial w}{\partial y} \quad \frac{\partial w}{\partial z} \right\}^T$$

$$\{\varepsilon\} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_{,x,y,z} \\ u_{,x,y,z} \\ u_{,x,y,z} \end{Bmatrix}$$

$$\{\varepsilon\} = [A] \begin{Bmatrix} u_{,x,y,z} \\ v_{,x,y,z} \\ w_{,x,y,z} \end{Bmatrix} \tag{3.11}$$

It is now necessary to determine the relationship between $\{u_{,x,y,z} \quad v_{,x,y,z} \quad w_{,x,y,z}\}^T$ and the derivatives with respect to the volume coordinates L_1, L_2, and L_3.

For the x-direction, the chain rule gives Equation 3.12 below.

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial L_1} \times \frac{\partial L_1}{\partial x} + \frac{\partial u}{\partial L_2} \times \frac{\partial L_2}{\partial x} + \frac{\partial u}{\partial L_3} \times \frac{\partial L_3}{\partial x} \tag{3.12}$$

Implementation of the chain rule for the other derivatives provides the following equations.

$$\begin{Bmatrix} u_{,x,y,z} \\ v_{,x,y,z} \\ w_{,x,y,z} \end{Bmatrix} = \begin{bmatrix} [I_3]\frac{\partial L_1}{\partial x} & [I_3]\frac{\partial L_2}{\partial x} & [I_3]\frac{\partial L_3}{\partial x} \\ [I_3]\frac{\partial L_1}{\partial y} & [I_3]\frac{\partial L_2}{\partial y} & [I_3]\frac{\partial L_3}{\partial y} \\ [I_3]\frac{\partial L_1}{\partial z} & [I_3]\frac{\partial L_2}{\partial z} & [I_3]\frac{\partial L_3}{\partial z} \end{bmatrix} \begin{Bmatrix} \{u \quad v \quad w\}^T_{,L1} \\ \{u \quad v \quad w\}^T_{,L2} \\ \{u \quad v \quad w\}^T_{,L3} \end{Bmatrix} \tag{3.13}$$

$$\begin{Bmatrix} u_{,x,y,z} \\ v_{,x,y,z} \\ w_{,x,y,z} \end{Bmatrix} = [\Gamma]\{g\} \tag{3.14}$$

Therefore, [Γ] can be represented as shown in Equation 3.15, and $\{g\}$ is shown in Equation 3.16. Note that an expression such as $\{u \quad v \quad w\}^T_{,L1}$ refers to the vector formed by the partials of *u*, *v*, and *w* with respect to L_1.

$$[\Gamma] = \begin{bmatrix} [I_3]\frac{\partial L_1}{\partial x} & [I_3]\frac{\partial L_2}{\partial x} & [I_3]\frac{\partial L_3}{\partial x} \\ [I_3]\frac{\partial L_1}{\partial y} & [I_3]\frac{\partial L_2}{\partial y} & [I_3]\frac{\partial L_3}{\partial y} \\ [I_3]\frac{\partial L_1}{\partial z} & [I_3]\frac{\partial L_2}{\partial z} & [I_3]\frac{\partial L_3}{\partial z} \end{bmatrix} \tag{3.15}$$

$$\{g\} = \begin{Bmatrix} \{u \quad v \quad w\}^T_{,L1} \\ \{u \quad v \quad w\}^T_{,L2} \\ \{u \quad v \quad w\}^T_{,L3} \end{Bmatrix} \tag{3.16}$$

The partial derivatives of the volume coordinates can be obtained from Equation 3.4 and are shown in Equations 3.17 – 3.19, where C_{ik} represents the entry at (i,k) in the inverse coefficient matrix (see Equation 3.3).

$$\frac{\partial L_i}{\partial x} = \frac{C_{i1}}{|J|}, \frac{\partial L_i}{\partial y} = \frac{C_{i2}}{|J|}, \frac{\partial L_i}{\partial z} = \frac{C_{i3}}{|J|} \qquad (3.17 - 3.19)$$

Recall that $|J|$ is equal to six times the volume of the element. The above equations indicate that the [Γ] matrix in Equation 3.15 is constant for each element since it depends only on the coordinates of the nodes of the element (see Equation 3.3). It can easily be implemented in closed form, and is calculated once for each element.

The next step is to determine the derivatives of the displacements with respect to the volume coordinates. This results in Equations 3.20 – 3.22 for displacement in the x-direction, and similar formulations for the y- and z-directions.

$$\frac{\partial u}{\partial L_1} = \sum_{i=1}^{nodes} \frac{\partial N_i}{\partial L_1} \times \delta u_i, \frac{\partial u}{\partial L_2} = \sum_{i=1}^{nodes} \frac{\partial N_i}{\partial L_2} \times \delta u_i, \frac{\partial u}{\partial L_3} = \sum_{i=1}^{nodes} \frac{\partial N_i}{\partial L_3} \times \delta u_i$$

(3.20 – 3.22)

When expanded for all directions, the derivatives of the displacements with respect to volume coordinates result in the formula below.

$$\begin{Bmatrix} \{u \quad v \quad w\}^T_{,L1} \\ \{u \quad v \quad w\}^T_{,L2} \\ \{u \quad v \quad w\}^T_{,L3} \end{Bmatrix} = \begin{bmatrix} [N]_{,L1} \\ [N]_{,L2} \\ [N]_{,L3} \end{bmatrix} \begin{Bmatrix} \delta u \\ \delta v \\ \delta w \end{Bmatrix} = [R]\{\delta \tilde{u}\} \qquad (3.23)$$

The [R] matrix will be the same for each element type because it is a function of the polynomial order of the element, and its dimensions are 9 x (3·*nodes*). It needs to be calculated only once for any element type.

If the element stiffness matrix is implemented using local volume coordinates instead of global Cartesian coordinates, the strain energy for an element is written as shown in Equation 3.24 below, where U_e is the element strain energy, $\{\varepsilon\}$ is the strain vector, [D] is the elasticity matrix, and Ω is the domain of integration.

$$U_e = \frac{1}{2}\int_{\Omega} \{\varepsilon\}^T [D]\{\varepsilon\} d\Omega = \frac{1}{2}\int_{vol} \{\varepsilon\}^T [D]\{\varepsilon\} \cdot dx \cdot dy \cdot dz \tag{3.24}$$

Note that the elasticity matrix [D] is represented as shown below [22].

$$[D] = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \nu/(1-\nu) & \nu/(1-\nu) & 0 & 0 & 0 \\ & 1 & \nu/(1-\nu) & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & \frac{(1-2\nu)}{2(1-\nu)} & 0 & 0 \\ & & & & \frac{(1-2\nu)}{2(1-\nu)} & 0 \\ sym & & & & & \frac{(1-2\nu)}{2(1-\nu)} \end{bmatrix} \tag{3.25}$$

The equation for strain energy, based on substitution of the strain vector as the product of the strain displacement matrix [B] times the nodal displacement vector $\{\delta u\}$, is shown in the following equations, where the domain integration Ω is the volume of the element.

$$U_e = \frac{1}{2}\int_{\Omega} \left([B]\{\delta u\}\right)^T [D]\left([B]\{\delta u\}\right) d\Omega \tag{3.26}$$

$$U_e = \frac{1}{2}\{\delta u\}^T \left(\int_{\Omega} [B]^T [D][B] d\Omega \right) \{\delta u\} \tag{3.27}$$

Expressing the strain energy in terms of the element stiffness matrix, [K_e], results in Equation 3.28. This formulation will aid in determining the element stiffness matrix.

$$U_e = \frac{1}{2}\{\delta u\}^T [K_e]\{\delta u\} \tag{3.28}$$

When Equation 3.27 and Equation 3.28 are compared, it is readily seen that the element stiffness matrix can be represented as shown in Equation 3.29.

$$[K_e] = \int_{\Omega} ([B])^T [D][B] d\Omega \tag{3.29}$$

Based on the formulations developed, the strain matrix can be manipulated into the form shown in Equation 3.31, where [P] = [A] [Γ].

$$\{\varepsilon\} = [B]\{\delta\tilde{u}\} = [A][\Gamma]\{g\} = [A][\Gamma][R]\{\delta\tilde{u}\} \tag{3.30}$$

$$\{\varepsilon\} = [P][R]\{\delta\tilde{u}\} \tag{3.31}$$

This means that the strain displacement matrix [B] can be calculated as the product $[B] = [P][R]$. Since the [P] matrix is a function of the [A] matrix (comprised of all constants) and the [Γ] matrix (which depends on the nodal coordinates of the element vertices and is constant for each element) [P] only needs to be calculated once for each element. Note that the dimensions for [P] are always 6 x 9.

These manipulated formulations can now be substituted into the equation for the element stiffness matrix [K_e].

$$[K_e] = \int_{vol} [R]^T [P]^T [D][P][R] \cdot |J| \cdot dL_1 \cdot dL_2 \cdot dL_3 \tag{3.32}$$

If [*G*] is defined as [*G*] = [*P*]^T [*D*] [*P*], it will be geometry and material dependent, needing to be calculated only once for each element and results in a 9x9 symmetric matrix of constants. Substitution of [*G*] into Equation 3.32 results in the final form of the stiffness matrix, shown in Equation 3.33.

$$[K_e] = \int_{vol} [R]^T [G][R] \cdot |J| \cdot dL_1 \cdot dL_2 \cdot dL_3 \tag{3.33}$$

3.2 Closed-Form Implementation

For closed-form calculations, the integration indicated in Equation 3.33 was accomplished using the formula below [22], where the domain of integration is the volume of the element, *a*, *b*, *c*, and *d* represent the powers to which the natural coordinates have been raised, and *V* is the volume of the element.

$$\iiint L_1^a L_2^b L_3^c L_4^d dL_1 dL_2 dL_3 dL_2 dL_4 = \left(a!\, b!\, c!\, d! \,/ (a+b+c+d+3)! \right) 6V \tag{3.34}$$

3.3 Curved-sided Elements

Curved-sided elements are defined in terms of local coordinates (ξ, η, ζ). These local coordinates represent the undistorted element; in the local coordinate system, the element has straight sides. The global coordinate system is defined in terms of the Cartesian coordinates (*x*, *y*, *z*); in the global coordinate system, the element is distorted, and thus has curved sides.

The steps to obtain the stiffness matrix (as well as equivalent nodal loads, stresses, and error estimation) involve several steps and two transformations. In the previous sections, the stiffness matrix was derived in terms of the global coordinates.

The first step is to define the shape functions [*N*] (global coordinates) as [*N'*] (in terms of the local coordinates). If we define the local coordinates in terms of the volume coordinates as shown in Equation 3.35, the local coordinates an be easily substituted into the shape functions.

$$L_1 = \xi, L_2 = \eta, L_3 = \zeta, L_4 = 1 - \xi - \eta - \zeta \tag{3.35}$$

The same procedure outlined previously in this chapter was used to calculate the stiffness matrix for curved elements. Modifications to the formulations are outlined below.

The $[\Gamma]$ matrix of Eq. 3.15 is modified to relate the derivatives of the global displacements *u*, *v*, *w* with respect to the local coordinates *ξ*, *η*, *ζ*, forming the curvilinear matrix $[\Gamma_c]$. Each entry Γ_{ij} is the *ij*th entry in the inverse Jacobian matrix $[J_c]$, which means that the determinant of the Jacobian matrix is present in the denominator of each entry in this matrix.

$$[\Gamma_c] = \begin{bmatrix} [I_3]\Gamma_{11} & [I_3]\Gamma_{12} & [I_3]\Gamma_{13} \\ [I_3]\Gamma_{21} & [I_3]\Gamma_{22} & [I_3]\Gamma_{23} \\ [I_3]\Gamma_{31} & [I_3]\Gamma_{32} & [I_3]\Gamma_{33} \end{bmatrix} \tag{3.36}$$

The $[R]$ matrix of Eq. 3.23 is modified so that it relates the derivatives of the global displacements with respect to the local coordinates *ξ*, *η*, *ζ*, forming the $[R_c]$ matrix, where $[N']$ represents the shape functions in terms of the local coordinates.

$$[R_c] = \begin{bmatrix} [N']_{,\xi} \\ [N']_{,\eta} \\ [N']_{,\zeta} \end{bmatrix} \tag{3.37}$$

The curved element stiffness matrix then takes on the following form, based on Eqns. 2.29, 2.30, and 3.32. Note that $[P_c] = [A]\left[\Gamma_c\right]$.

$$[K_{ec}] = \int_0^1 \int_0^{1-\eta} \int_0^{1-\eta-\zeta} [R_c]^T [P_c]^T [D][P_c][R_c] det(J_c)\, d\xi\, d\eta\, d\zeta \qquad (3.38)$$

If, as in the case of Eq. 3.33, $[G_c]$ is defined as $[G_c] = [P_c]^T[D]\,[P_c]$, the matrix $[G_c]$ is be geometry and material dependent, needing to be calculated only once for each curved element, as in the case of straight sided elements. Substitution of $[G_c]$ into Equation 3.33 results in the final form of the curved stiffness matrix, shown in Equation 3.39.

$$[K_{ec}] = \int_0^1 \int_0^{1-\eta} \int_0^{1-\eta-\zeta} [R_c]^T [G_c][R_c] det(J_c)\, d\xi\, d\eta\, d\zeta \qquad (3.39)$$

Because of the presence of the determinant of the curved Jacobian in the denominator of the integrand, numerical integration is required, as discussed in the next section.

3.4 Numerical Integration Implementation

For straight-sided elements, Gaussian cubature was used. Fellipa's summary of FEM integration rules [41] and the work of Gellert and Harboud [47] is the source for the Gauss points used, implemented according to Equation 3.40 below. In the equation, *nGauss* is the number of Gauss points used, w_i is the weight for that Gauss point, $F(\varphi_i)$ is the value of the integrand at Gauss point *i*.

$$\int_\Omega F(\varphi) d\Omega = vol \times \sum_{i=1}^{nGauss} w_i\, F(\varphi_i) \qquad (3.40)$$

Gauss points used are as follows: *p*-level 1, 1 point; *p*-level 2, 8 points; *p*-level 3, 14 points; *p*-level 4, 20 points. The least number of points required for accuracy

were used. This was determined by first implementing more Gauss points than needed, then reducing the number of Gauss points used until the results were no longer acceptable.

Chapter 4 presents both the closed-form and numerical error estimators, and ends with a discussion of measures of error.

CHAPTER 4

ERROR ESTIMATION

This research uses the Zienkiewicz-Zhu error estimator [33] with nodal averaging as developed by Byrd [35] in his dissertation, and implemented by ANSYS [36]. The theory behind this method is as follows: interelement displacement continuity is guaranteed for C_0 continuous elements, such as hierarchical elements or isoparametric elements, but continuity of either stress or strain is not guaranteed. This implies that if a node is shared by two elements, it will most likely have two different stress (or strain) values. A reasonable assumption would be that the actual stress (or strain) at this node lies somewhere between the two values.

It follows that discontinuities in either stress or strain could provide an estimate of the error in the finite element solution. Since the exact solution is not known, an improved estimate of the stress or strain distribution could be obtained by nodal smoothing, and the smoothing must be performed separately for each material region of the element [48]. For this research, the error estimation is based on the stresses.

4.1 Equation Development

If $\|e_i\|^2$ represents the error in the energy norm in element *i*, it can be calculated as shown in Equation 4.1, where $[C]$ is the compliance matrix, e_σ is the error of the stresses, and Ω is the domain of the element.

$$\|e_i\|^2 = \int_\Omega e_\sigma^T [C] e_\sigma d\Omega \tag{4.1}$$

The error of the stresses is based on Equation 4.2, where σ^*represents the nodally averaged stresses and $\hat{\sigma}$ represents the stresses based on the finite element solution.

$$e_\sigma = \sigma^* - \hat{\sigma} \tag{4.2}$$

Substitution of Equation 4.1 into 4.2 results in the expression shown in Equation 4.3, and expanded in Equation 4.4.

$$\|e_i\|^2 = \int_\Omega (\sigma^* - \hat{\sigma})^T [C](\sigma^* - \hat{\sigma}) d\Omega \tag{4.3}$$

$$\|e_i\|^2 = \int_\Omega \sigma^{*T} [C]\sigma^* d\Omega - 2\int_\Omega \sigma^{*T} [C]\hat{\sigma} d\Omega + \int_\Omega \hat{\sigma}^T [C]\hat{\sigma} d\Omega \tag{4.4}$$

The first and last terms of Equation 4.4 are equal to two times the strain energy as estimated by the smoothed solution and two times the strain energy obtained by the finite element solution, respectively.

4.2 Stresses for Error Estimation

The smoothed stresses σ^* are interpolated over the mesh, on an element by element basis, based on the averaged nodal stresses, represented by $\overline{\sigma^*}$. These smoothed stresses are obtained using Equation 4.5, where *N* represents the shape function matrix used for interpolation.

$$\sigma^* = \begin{bmatrix} N & 0 & 0 & 0 & 0 & 0 \\ 0 & N & 0 & 0 & 0 & 0 \\ 0 & 0 & N & 0 & 0 & 0 \\ 0 & 0 & 0 & N & 0 & 0 \\ 0 & 0 & 0 & 0 & N & 0 \\ 0 & 0 & 0 & 0 & 0 & N \end{bmatrix} \overline{\sigma^*} \tag{4.5}$$

Because not all element nodes are displacement nodes [7], the hierarchical shape functions did not produce correct results when used for the interpolation. The solution to this problem was to use isoparametric shape functions for the smoothed stresses. In higher order elements, the shape functions used for interpolating stresses do not have to be the same as those used to produce the stiffness matrix.

For *p*-level 2, linear strain tetrahedral shape functions are used; for *p*-levels 3 and 4, quadratic strain shape functions are used. The linear strain and quadratic strain shape functions are summarized in Tables 4.1 and 4.2, as found in Zienkiewicz [22]. For the fourth order isoparametric element, the same shape functions used to obtain stiffness were also used for smoothing.

Table 4.1 Linear Strain Tetrahedral Element Shape Functions.

Node	Type	Shape Function	Node	Type	Shape Function
1	Vertex	$2L_1^2-L_1$	6	Edge 13	$4L_1L_3$
2	Vertex	$2L_2^2-L_2$	7	Edge 12	$4L_1L_2$
3	Vertex	$2L_3^2-L_3$	8	Edge 14	$4L_1L_4$
4	Vertex	$2L_4^2-L_4$	9	Edge 24	$4L_2L_4$
5	Edge 23	$4L_2L_3$	10	Edge 34	$4L_3L_4$

Table 4.2 Quadratic Strain Tetrahedral Element Shape Functions

Node	Type	Shape Function	Node	Type	Shape Function
1	Vertex	$(3L_1-1)(3L_1-2)\frac{L_1}{2}$	11	Edge 23	$(9L_2L_3)(3L_3-1)/2$
2	Vertex	$(3L_2-1)(3L_2-2)\frac{L_2}{2}$	12	Edge 13	$(9L_1L_3)(3L_3-1)/2$
3	Vertex	$(3L_2-1)(3L_3-2)\frac{L_3}{2}$	13	Edge 12	$(9L_1L_2)(3L_2-1)/2$
4	Vertex	$(3L_4-1)(3L_4-2)\frac{L_4}{2}$	14	Edge 14	$(9L_1L_4)(3L_4-1)/2$
5	Edge 23	$(9L_2L_3)(3L_2-1)/2$	15	Edge 24	$(9L_2L_4)(3L_4-1)/2$
6	Edge 13	$(9L_1L_3)(3L_1-1)/2$	16	Edge 34	$(9L_2L_4)(3L_4-1)/2$
7	Edge 12	$(9L_1L_2)(3L_1-1)/2$	17	Face 234	$27L_2L_3L_4$
8	Edge 14	$(9L_1L_4)(3L_1-1)/2$	18	Face 134	$27L_1L_3L_4$
9	Edge 24	$(9L_2L_4)(3L_2-1)/2$	19	Face 124	$27L_1L_2L_4$
10	Edge 34	$(9L_2L_4)(3L_3-1)/2$	20	Face 123	$27L_1L_2L_3$

Before the isoparametric shape functions can be used with *p*-levels 3 and 4, the averaged nodal stresses have to be obtained at the isoparametric node points along the edge, as opposed to the hierarchical node points.

All hierarchical edge nodes are located at the midpoint of the edge, and all hierarchical face nodes are located at the midpoint of the face. For isoparametric elements, if there is one face node, it is at the midpoint; if there are three face nodes, they are evenly distributed on the face of the element; if there is one edge node, it lies at

the midpoint of the edge; if there are three edge nodes, they are evenly distributed along the edge.

For *p*-level 4, the corresponding cubic strain tetrahedron can not be used because there were not enough known hierarchical stresses to allow for interpolation, thus a *p*-level 3 error estimation was used for terms 1 and 2 of the error estimator; this issue is illustrated in Fig. 4.1. This method did not produce satisfactory error estimates, and was found unsuitable for the fourth order hierarchic tetrahedral element.

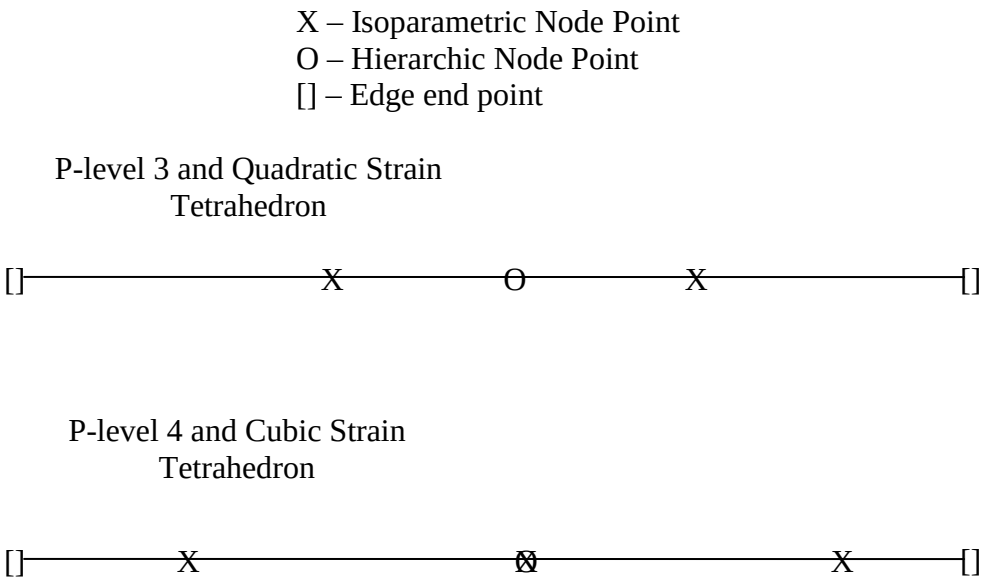


Figure 4.1 Stress interpolation illustration for p-level 3 and 4.

The finite element stress vector, $\{\hat{\sigma}\}$, is derived from elasticity matrix $[D]$ and the strain vector $\{\epsilon\}$.

$$\{\hat{\sigma}\} = [D]\{\epsilon\} \tag{4.6}$$

Recalling the expression of $\{\epsilon\}$ in Equation 3.31, the following equation for the finite element stress vector is obtained.

$$\{\hat{\sigma}\} = [D][P][R]\{\delta\tilde{u}\} \tag{4.7}$$

Note that $[P][R] = [B]$, and $\{\delta\tilde{u}\}$ represents the displacement vector.

For curved-sided elements, shape functions based on local coordinates ξ, η, ζ were used in the above equations. In the case of Eq. 4.6, $[P_c]$ and $[R_c]$ were implemented, as in the case of the stiffness matrix. See Eqn. 3.37 for details on the formulation of $[R_c]$.

$$\{\hat{\sigma}\} = [D][P_c][R_c]\{\delta\tilde{u}\} \tag{4.8}$$

4.3 Closed-form Implementation

The closed-form implementation deals with each term in Equation 4.4 separately. The manipulation and implementation of these terms will be discussed in the following sections.

4.3.1 Term 1 of the Error Estimator

Term 1 (*Term*1) of the error estimator is shown in Equation 4.9 below, where σ^* represents the nodally averaged stresses, and [*C*] is the element compliance matrix.

$$Term1 = \int_{\Omega} \sigma^{*T} [C]\sigma^* d\Omega \tag{4.9}$$

Recalling the expression of σ^* found in Equation 4.5, the form of *Term*1 can be expanded as shown.

$$Term1 = \int_{\Omega} ([N]\bar{\sigma}^*)^T [C]([N]\bar{\sigma}^*) d\Omega \tag{4.10}$$

This expression can be further simplified by recalling that the nodal averaged stress vector can be taken outside the integral because it is not a function of the volume coordinates. Equation 4.11 shows the form of *Term*1 that was implemented in closed-form using *Mathematica*.

$$Term1 = \bar{\sigma}^{*T} \left(\int_{\Omega} [N]^T [C][N] d\Omega \right) \bar{\sigma}^{*} \tag{4.11}$$

4.3.2 Term 2 of the Error Estimator

Term 2 (*Term*2) of the error estimator is repeated in Equation 4.12 below, where σ^* represents the nodally averaged stresses, $\hat{\sigma}$ represents the finite element stresses, and [*C*] is the element compliance matrix.

$$Term2 = \int_{\Omega} \sigma^{*T} [C] \hat{\sigma} d\Omega \tag{4.12}$$

Substituting for the nodally averaged stresses in Equation 4.5 and the finite element stresses in Equation 4.6, the following expression is obtained. Recall that [*D*] is the elasticity matrix, [*B*] is the strain displacement matrix, and $\{\delta\tilde{u}\}$ represents the displacement vector.

$$Term2 = \int_{\Omega} ([N]\bar{\sigma}^{*})^T [C][D][B]\{\delta\tilde{u}\} d\Omega \tag{4.13}$$

This expression can be simplified by taking $\{\delta\tilde{u}\}$ and $\bar{\sigma}^*$ outside the integral because they are not dependent on the volume coordinates. The form shown in 4.12 was implemented in *Mathematica* to obtain a closed-form expression.

$$Term2 = \bar{\sigma}^{*T} \left(\int_{\Omega} [N]^T [B] d\Omega \right) \{\delta\tilde{u}\} \tag{4.14}$$

4.3.3 Term 3 of the Error Estimator

Term 3 (*Term*3) of the error estimator is shown below, where $\hat{\sigma}$ represents the finite element stresses, and [*C*] is the element compliance matrix.

$$Term3 = \int_{\Omega} \hat{\sigma}^T [C] \hat{\sigma} d\Omega \tag{4.15}$$

Substituting for the finite element stresses in Equation 4.6, the following expression is obtained. Recall again that [*D*] is the elasticity matrix, [*B*] is the strain displacement matrix, and $\{\delta\tilde{u}\}$ represents the displacement vector.

$$Term3 = \int_{\Omega} ([B]\{\delta\tilde{u}\})^T [C]([B]\{\delta\tilde{u}\}) d\Omega \tag{4.16}$$

It is known that [*B*] = [*P*][*R*], and thus can be substituted into Equation 4.16.

$$Term3 = \int_{\Omega} ([D][P][R]\{\delta\tilde{u}\})^T [C]([D][P][R]\{\delta\tilde{u}\}) d\Omega \tag{4.17}$$

Rearranging some terms, and taking the displacement vector outside the integral, another expression is obtained.

$$Term3 = \{\delta\tilde{u}\}^T \left(\int_{\Omega} [R]^T [P]^T [D]^T [C][D][P][R] d\Omega \right) \{\delta\tilde{u}\} \tag{4.18}$$

If [*G*] is again defined as [*G*] = $[P]^T[D]$ [*P*], another form of *Term*3 can be obtained by combining the middle matrices as shown in Equation 4.17.

$$Term3 = \{\delta\tilde{u}\}^T \left(\int_{\Omega} [R]^T [G][R] d\Omega \right) \{\delta\tilde{u}\} \tag{4.19}$$

This is the form of *Term*3 implemented in *Mathematica* to obtain the closed-form solution expressions.

4.4 Numerical Integration Implementation

The numerical implementation of the error estimator is performed in accordance with Eq. 4.3 (reproduced below), which is applicable to both curved-sided and straight

sided elements, and involves integration of the difference between σ^* and $\hat{\sigma}$ over each element.

$$\|e_i\|^2 = \int_{\Omega} (\sigma^* - \hat{\sigma})^T [C](\sigma^* - \hat{\sigma}) d\Omega \tag{4.3}$$

Fellipa's summary of FEM integration rules was the source for the Gauss points used [41], according to Equation 4.20 below. In the equation, *nGauss* is the number of Gauss points used, w_i is the weight for that Gauss point, $F(\varphi_i)$ is the value of the integrand at Gauss point *i*.

$$\int_{\Omega} F(\varphi) d\Omega = vol \times \sum_{i=1}^{nGauss} w_i F(\varphi_i) \tag{4.20}$$

The least number of points required for accuracy were used. The Gauss points used are as follows: *p*-level 1, 1 point; *p*-level 2, 14 points; *p*-level 3, 24 points; *p*-level 4, 24 points.

For the curved-sided isoparametric fourth order elements, 4-4-6 Gauss-Lobatto points were used as described by Peano [49]. The expression used is shown below in Eq. 4.21, where *n* represents the number of Gauss points for *i* and *j*, and the number of Lobatto points for *k*, W^G is the associated weight for Gauss points *i* and *j*, respectively, W^G is the associated weight for Lobatto point *k*, and ζ_i, η_j, ξ_k represent the two Gauss points *i* and *j* and the Lobatto point *k*, respectively. Note that $L_1 = \frac{1}{4}(1-\eta)(1-\zeta)$, $L_2 = \frac{1}{4}(1+\eta)(1-\zeta)$, and $L_3 = \frac{1}{4}(1+\eta)(1+\zeta)$.

$$\int_{\Omega} F(L_1, L_2, L_3) dL_1 dL_2 dL_3 = \sum_{i,j,k=1}^{n} W_j^G W_j^G W_i^L \left(1 - \zeta_k^2 / 32\right) F\left(\zeta_i, \eta_j, \xi_k\right) \tag{4.21}$$

4.5 Measurements of Error

The global error for the model can be calculated according to Equation 4.22, after the error estimate for each element has been evaluated [22]. The variable *nel* represents the number of elements in the model and $\|e_i\|^2$ represents the error in the energy norm in element *i*.

$$\|e\|^2 = \sum_{i=1}^{nel} \|e_i\|^2 \tag{4.22}$$

The total finite element energy norm is given by the equation below, where $\|\hat{u}_i\|^2$ is the strain energy of element *i*, equal to ½ of the third term of the error estimator.

$$\|\hat{u}\|^2 = \sum_{i=1}^{nel} \|\hat{u}_i\|^2 \tag{4.23}$$

To obtain the total strain energy of the model, Eq. 4.24 is used, where the sum is taken of the sum of total error energy norm $\|e\|^2$ and the finite element energy norm, $\|\hat{u}\|^2$. Finally, the total (global) error of a model is estimated as the ratio between the total error energy norm, $\|e\|^2$, and the total strain energy of the model, $\|u\|^2$ [22].

$$\|u\|^2 = \|e\|^2 + \|\hat{u}\|^2 \tag{4.24}$$

$$\eta = {\|e\|}/{\|u\|} \tag{4.25}$$

If a specified maximum mean permissible error $\overline{\eta}$ needs to be achieved, a mean permissible error that exceeds the value would indicate that the next *p*-level should be utilized for this model. Once available *p*-levels have been exhausted, the model should be remeshed and the process started again at *p*-level 1. The mean permissible error is calculated as shown below in Eq. 4.26 [22].

$$e_m = \overline{\eta} \sqrt{\|u\|^2 / nel} \tag{4.26}$$

Note that the finite element solution seeks to minimize the strain energy of the error, and is thus a logical measure of the overall quality of the solution, but a small error in the energy norm does not necessarily imply a small error in other quantities of interest, such as displacement or stress concentration.

Chapter 5 presents the development of the equivalent nodal loads for pressure, shear, and temperature for both straight-sided and curved-sided elements.

CHAPTER 5

EQUIVALENT NODAL LOAD VECTORS

Equivalent nodal load vectors refer to the discretization of applied surface loads, such as pressure, shear, or temperature. This discretization involves correctly allocating the loads to element nodal points. Pressure loads are assumed perpendicular to the element face, shear loads are assumed parallel to the element face, and the temperature change of an element is calculated as the average of the vertex temperatures.

The discussion of methodology that follows for temperature, pressure, and shear is closely based on the derivation and approach used by Shiakolas [4]. Pressure, shear, and temperature equivalent nodal load vectors will be derived, and both closed-form implementation used for straight-sided elements, and numerical implementation used for curved-sided elements, will be discussed.

5.1 Equivalent Temperature

If an elastic body is unconstrained, a temperature change results in expansion or contraction. Temperature changes from a preset datum are treated as an initial strain; this strain must be allocated to the element nodes, resulting into an equivalent nodal temperature load. [4, 50]. Equation 5.1 below shows the equivalent nodal temperature load vector, where [B] is the strain displacement matrix, [D] is the elasticity matrix, ε_0

is the initial strain due to the temperature load, and Ω represents the domain of integration [50].

$$\{r\}_{Temp} = \int_{\Omega} [B]^T [D] \varepsilon_0 \, d\Omega \tag{5.1}$$

The initial strain is a function of α, the coefficient of thermal expansion, and ΔT, the average of the vertex temperatures. If we assume an isotropic material, the initial strain takes on the following form.

$$\varepsilon_0 = \alpha \Delta T \{1,1,1,0,0,0\}^T \tag{5.2}$$

Substituting Equation 5.2 into Equation 5.1, noting that the domain of integration is over the whole element (thus its volume), and taking all non-volume coordinate dependent terms out of the integral, the expression for the equivalent nodal temperature load is expressed as follows.

$$\{r\}_{Temp} = \left(\int_{Vol} [B]^T \, dVol \right) [D] \alpha \Delta T \{1,1,1,0,0,0\}^T \tag{5.3}$$

Recall that [*B*] = [*P*][*R*], and thus the integrand is expressed as the volume coordinates raised to various powers and evaluated according to Equation 3.34, reproduced below.

$$\iiint L_1^a L_2^b L_3^c L_4^d dL_1 dL_2 dL_3 dL_2 dL_4 = \left(a!\, b!\, c!\, d! \Big/ (a+b+c+d+3)! \right) 6V \tag{3.34}$$

As with the stiffness matrix and error estimation terms, the temperature loads, when solved symbolically, also exhibit nested terms for hierarchical elements. For example, the hierarchical *p*-level 4 equivalent nodal load vector includes the terms of the *p*-level 3 equivalent nodal load vector. This is not true of isoparametric elements.

Numerical implementation, used for the curved elements, involves the use of Gaussian quadrature to evaluate the integral of Equation 5.3.

5.2 Applied Pressure or Shear

The equivalent nodal load vector for surface loads such as pressure and shear is given in Equation 5.4 below, where [*N*] represents the shape function matrix, $\{\Phi\}$ is the applied load, and the domain of integration is the surface to which the load is applied (the area of the tetrahedral face). [50]

$$\{r\} = \begin{Bmatrix} r_x \\ r_y \\ r_z \end{Bmatrix} = \int_{surface} [N]^T \{\Phi\}\, dS \tag{5.4}$$

The shape function matrix used in Equation 5.4 is expressed as follows, where [*I*] is a 3x3 identity matrix and N_i is the shape function for node *i*, and *nodes* is the number of nodes in the element.

$$[N] = \left[[I]N_1, [I]N_2, \cdots, [I]N_{nodes} \right] \tag{5.5}$$

To use Equation 5.4, the applied load $\{\Phi\}$ is expressed as Φ_v, the direction cosine vector, and Φ_m, the magnitude of the applied pressure or shear load.

$$\{r\} = \int_{surface} [N]^T \{\Phi_v\} \Phi_m \, dS = \left(\int_{surface} [N]^T \, dS \right) \{\Phi_v\} \Phi_m \tag{5.6}$$

Note that the direction cosine vector and magnitude are taken outside the integral, and the resulting integrand consists of volume coordinates raised to various powers. In closed-form, Equation 5.6 is integrated using Equation 5.7 [22].

$$\iiint L_1^a L_2^b L_3^c dL_1 dL_2 dL_3 = \left(a!\, b!\, c!\, d! \Big/ (a + b + c + 2)! \right) 2A \tag{5.7}$$

Note that pressure loads act normal to the face, thus load magnitude, the face to which the pressure is applied (or node opposite to the face), and whether it is compressive or tensile is the only information required for calculating pressure loads. Shear loads will require the load magnitude and the face where the shear acts, as well as the direction cosine vector.

The following discussion deals with the direction cosine vector, and comes from the work of Shiakolas [4].

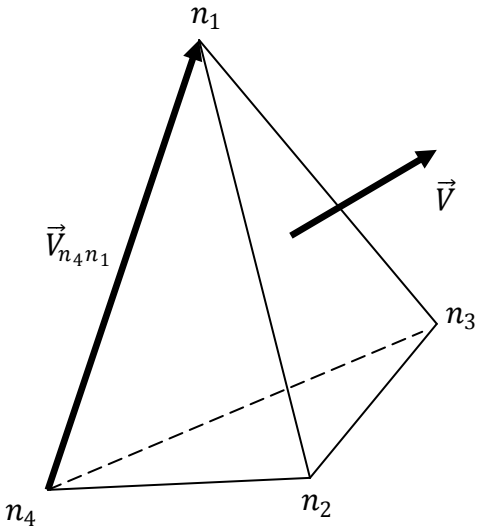


Figure 5.1 Illustration for derivation of a direction cosine vector applied to face n_1, n_2, and n_3.

Figure 5.1 shows an applied pressure load on a face composed of local nodes n_1, n_2, and n_3. Note vector $\vec{V}$ perpendicular to the face. If vectors $\vec{V}_{ij}$ are defined as vectors from node j to node i, then the normalized unit vector $\overrightarrow{V_u}$ can be calculated as follows.

$$\vec{V} = \vec{V}_{n_1 n_2} \times \vec{V}_{n_1 n_3} \tag{5.8}$$

$$\overrightarrow{V_u} = \vec{V} \Big/ \|\vec{V}\| \tag{5.9}$$

To determine if the resulting vector is pointing towards the face or away from the face, the dot product between $\overrightarrow{V_u}$ and $\vec{V}_{n_4 n_1}$ can be taken. If it is greater than or equal to zero, the vectors point in the same direction and $\overrightarrow{V_u}$ is directed away from the loaded face; otherwise, $\overrightarrow{V_u}$ is directed towards the loaded face.

The applied load $\{\Phi\}$ can now be expressed in terms of as $\overrightarrow{V_u}$, as shown below.

$$\{\Phi\} = \left|\vec{V}\right|\overrightarrow{V_u} \tag{5.10}$$

Finally, the area of the loaded face can be obtained by taking magnitude of the cross product of any two vectors of the face.

$$A_f = \frac{1}{2}\left\|\vec{V}_{n_1 n_2} \times \vec{V}_{n_1 n_3}\right\| \tag{5.11}$$

5.3 Modifications for Curved-sided Elements

Modifications required to implement the expressions in this chapter include recognizing that the domain of integration, Ω, is now the area of the curved face, rather than a planar face. The application of the load is assumed to be on the curved face.

Vector *dA* is assumed normal to the curved surface, represented by Eq. 5.12 below [22], where $\frac{\partial x}{\partial \xi}$, and similar expressions, are evaluated according to Eq. 2.6.

$$dA = \begin{Bmatrix} \frac{\partial x}{\partial \xi} \\ \frac{\partial y}{\partial \xi} \\ \frac{\partial z}{\partial \xi} \end{Bmatrix} \times \begin{Bmatrix} \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \eta} \end{Bmatrix} d\xi\, d \tag{5.12}$$

$$dA = \begin{Bmatrix} dA_{xy} \\ dA_{yz} \\ dA_{zx} \end{Bmatrix} = \begin{Bmatrix} \begin{vmatrix} \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \end{vmatrix} & - \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial z}{\partial \eta} \end{vmatrix} & \begin{vmatrix} \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{vmatrix} \end{Bmatrix}^T d\xi\, d\eta \tag{5.13}$$

Numerical integration is required, and is performed based on the following representation for curvilinear elements, where $[\bar{H}(\xi, \eta)]$ is the integrand in terms of the local coordinate system.

$$\int_0^1 \int_0^{1-\eta} [\bar{H}(\xi, \eta)] d\xi\, d\eta \tag{5.14}$$

For curved-sided elements, Equation 5.14 is integrated using Gaussian quadrature according to Equation 5.15 [41], where *nGauss* represents the number of Gauss points used, w_i is the weight at Gauss point i, and $F(\varphi_i)$ is the value of the integrand at Gauss point i.

$$\int_{surface} F(\varphi) dS = Area \times \sum_{i=1}^{nGauss} w_i\, F(\varphi_i) \tag{5.15}$$

Seventy-nine Gauss points were found sufficient for the curved-sided fourth order isoparametric element implementation**.**

Discussed in the following chapter is the source code compaction algorithm, including a discussion of its implementation, verification, and a simple example of how it works.

CHAPTER 6

SOURCE CODE COMPACTION

Mathematica 12© is used to produce the closed-form source code files (as discussed in the next chapter), drawing on both its mathematical features and its ability to convert expressions to a Fortran compatible format. One of the major hindrances in developing closed-form expressions for *p*-level 3 and greater is the length of the expressions, and the resulting size of the source code files containing these expressions.

Source code files produced for this research grew large for hierarchical *p*-levels 3 and 4, and many of the expressions were excessively long, spreading over more than one hundred lines. Because these expressions often contained repeated terms, a unique variable name could be substituted for repeated terms in a source code file, thus size of the file would be reduced, and by deduction it can be assumed that the processing time would also be reduced.

For the *p*-level 4 isoparametric element, the Mathematica output for some source code files involved expressions of such length that they exceeded the line continuation limits for standard Fortran compilers. Compaction was a definite necessity for these files to get the source code into a form for implementation.

This chapter describes the compaction program written to take advantage of repeated terms and reduce the size of the source code. It is based on the concept of developing a "dictionary" for a specific code unit and replacing the actual terms with references to the dictionary. This process is often used to compact electronic transmission of messages [51].

Sample results of compaction will also be discussed.

6.1 Production of Source Code Files

As mentioned, Mathematica scripts are prepared to produce as much of the code as possible, including both closed-form and numerical implementations. Expressions are generally factored based on common variables shared by the expressions in a source code file, such as nodal stress or displacement components, and thus many terms in a source code file can be placed in parentheses. This made it easier to detect possible repeated terms when compaction was performed. Note that each type of source code file (error estimator terms, stiffness, centroidal stresses, etc.) generally required some study of the resulting expressions in order to determine a good variable to perform such grouping upon. Details of how this was accomplished can be found in Chapter 7.

6.2 Compaction Implementation

The compaction program views each expression as a combination of characters and digits, or in what most programming languages refer to as a string. Visual Basic 2005 was chosen as the language for this program because of its string manipulation capabilities and its ability to handle strings that are 2 GB in size [52], thus allowing the each expression to be manipulated without concern as to its length.

The general algorithm for the compaction process is as follows:

1. Perform an initial scan of the source code file
    a. Replace any instances of Sqrt with an appropriate variable name and add the Sqrt terms to the dictionary
    b. Replace any instances of terms such as sx1**2 with the equivalent sx1*sx1
    c. Note that both actions will reduce calculation time when implemented in Fortran
    d. Save the new source code file to temp.txt
2. Scan temp.txt for terms that appear within parentheses
    a. If a term already exists, the counter for how many times it has appeared will be incremented
    b. If a term does not exist, it will be added to the end of the dictionary and the counter for how many times it has appeared will be initialized to 1
    c. No changes are made to temp.txt
3. Rewind the file temp.txt
4. Create the new compacted source code file
    a. Begin the source code file with a list of variable names set equal to the terms they represent
    b. Only terms that appear more than twice in the source code file (or at least once in the case of Sqrt) are included in this list
5. For each line in temp.txt

a. For each term in the dictionary
    i. Check to see if that term appears in the line
    ii. If a term appears and its counter indicates that it has been used in the original source code file at least twice, replace that term in the expression with the appropriate variable name
b. After all possible substitutions have been made, the modified expression contained in this line is written to the compacted source code file

This scanning portion of this algorithm requires that the program be able to identify certain tokens (including the assignment operator "=", a call to the Sqrt function of Fortran, the exponentiation operator "**" used in Fortran), differentiate between a negative sign and a minus sign, be able to match parentheses, and recognize and extract a term as identified by the presence of opening and closing parentheses without being preceded by Sqrt.

This program was written to abstractly view the stream of characters comprising an expression; it performs no mathematical operations in order to reduce the expression size, using instead fundamental concepts of string manipulation and comparison. The compaction program requires minimal knowledge of what the actual expressions represent and what mathematical operations are taking place.

Note that various permutations of terms such as sx1*u1, sx1**2, and tx1*sx1 were investigated for substitution also, but it was found that the algorithm above provided the best compaction for the majority of the files.

6.3 Simple Compaction Example

Below is an example of a line from a non-compacted source code file as it would appear with white space removed.

```
t3el11=sx1*(c11*u1+c12*u2+c13*u3-c11*u4-c12*u4-c13*u4)+sx2*(c11*u1
$+c12*u2+c13*u3-c11*u4-c12*u4-c13*u4)+sx3*(c11*u1+c12*u2+c13*u3-c11
$*u4-c12*u4-c13*u4)+sx4*(c11*u1+c12*u2+c13*u3-c11*u4-c12*u4-c13*u4)
```

Here is the variable that Compactor determined could be substituted into this term.

```
q0=(c11*u1+c12*u2+c13*u3-c11*u4-c12*u4-c13*u4)
```

Next, the same line is shown in the compacted file with the above substitution made.

```
t3el11=sx1*q0+sx2*q0+sx3*q0+sx4*q0
```

The expression is much smaller, and thus requires less space on disk to store the source code file, and also means a smaller executable. Note that this compaction routine provides benefits beyond smaller source code fields.

In terms of run-time efficiency, the term associated with q0 needs to be calculated only once. If this represents a calculation performed four times per element, now performed only once, and suppose there were 500 elements in the model, then 500 x (4 - 1) = 1,500 unnecessary calculations have been eliminated by compacting this one line of code.

By replacing all instances of Sqrt with an appropriate variable name, the program will run more efficiently because it will need to evaluate these terms once per

sub procedure call, rather than multiple times in a sub procedure. Even for *p*-levels for which file size is not an issue, an increase in efficiency during run-time is beneficial.

6.4 Compacted Code Verification

It is vitally important that the compacted source code file calculations produce the same numerical results as the non-compacted calculations. To verify that compaction does not change the nature of the calculations, a basic check was run using an original source code file to calculate a value, followed by a compacted source code file. Both files produced identical results to at least eight decimal places, and possibly more.

Further testing was performed after the compacted source code files were implemented in the finite element program. For example, concerns arose over displacement results that could have been due to errors in the compacted stiffness source code. To check this, a non-compacted version of the stiffness calculation was placed in the finite element program, and the exact same results for displacement were produced for a representative test case.

6.5 Code Formatting Issues

Mathematica has the capability of producing output in Fortran format, but it is extremely difficult with Mathematica 12 to obtain that output in fixed-format Fortran 77. Recall that Fortran 77 fixed-formatting reserves column 1 for marking a comment, columns 1 – 5 for statement labels, column 6 as the position for a line-continuation character, and columns 7 – 72 for the actual Fortran statement. The Mathematica

produced source code required conversion to fixed-format Fortran, which also involved splitting the extremely long lines using line continuation characters.

For purposes of the compaction program, Fortran 77 fixed-format was not an issue, thus each expression appears on its own line of "unlimited" length prior to compaction. Such formatting is not appropriate, however, for use as Fortran source code because of line-length limitations.

To convert a file (usually subsequent to compaction) to the appropriate Fortran formatting, a program was written in Visual Basic to read each line of code from a compacted source code file (or an original Mathematica output source code file), remove all whitespace, and reformat that line such that it meets fixed-format requirements and uses.

Note that limits on the number of line continuation characters used made it advisable to remove all whitespace from the source code files prior to reformatting. Removal of white-space also reduced source code file size, since each blank space is treated as a character requiring 1-byte of storage space. Note that, if a file was to be compacted, this program was used subsequent to compaction.

Chapter 7 provides a brief discussion of the usage of computer algebra systems in this research, including major features of *Mathematica* that were implemented as well as various areas of this research where those features were used.

CHAPTER 7

COMPUTER ALGEBRA SYSTEM USAGE

Extensive use of *Mathematica* 12© was made in the development and implementation of the family of elements used in this research. The proper use of a computer algebra system reduces the likelihood of simple mathematical errors that can devastate problem solutions, minimizes mistakes that programmers can make while typing in long expressions, and allow for manipulation of equations that is simply not feasible by hand in a time-efficient manner, such as the *p*-level 4 shape function development. Once the correct syntax and usage of Mathematica is understood properly, it allows for better quality code to be produced in a shorter time with less debugging necessary.

7.1 Major Features Used

Various feature of *Mathematica* were implemented in the scripts used for this research. These features included matrix manipulation, expansion of polynomial products [52], and the subsequent usage of transformation rules [53] to implement symbolic integration such as reproduced below, for integration over an element volume.

$$\iiint L_1^a L_2^b L_3^c L_4^d dxdydz = \frac{a!b!c!d!}{(a+b+c+d+3)!} 6V \tag{3.34}$$

Mathematica integration commands were not used because all integrations performed took the form of Eq. 3.34 above.

The Visual Basic compaction program discussed in Chapter 6 looks for repeated terms found in parentheses; these terms were mainly extracted using the Mathematica command Collect, which collects terms involving the same power of user-selected variables [54]. This process would not have been feasible by hand and was necessary for compaction. It also allowed for experimentation to determine which set of variables to "Collect" on in order to achieve improved compaction.

Note that by thus pre-conditioning the source code files in *Mathematica* so that many terms were grouped in parentheses, the compaction code could be written abstractly and there was no need to spend excessive time implementing code in Visual Basic to do what *Mathematica* could do in a few short commands and with far less likelihood of error. Compaction could be performed by viewing the expressions as streams of characters, rather than be concerned with the mathematical operations being performed.

Possibly the most important feature exploited in this work was the ability of *Mathematica* to reformat mathematical expressions to match Fortran programming syntax, then being able to write these expressions out to a text file [55]. This capability allows the computer algebra system to take care of the complex matrix, algebraic, and symbolic manipulation, as well as differentiation, and then reproduce the resulting expressions in a form that can easily be placed into an existing program structure. By

removing the programmer from the "translation" process, the possibility of mistakes in the expressions is reduced and the resulting code is likely more reliable.

7.2 Areas of Implementation

The first use of *Mathematica* scripts in this research was in obtaining the correct form of the *p*-level 2 through 4 hierarchic shape functions, and especially in developing the non-documented *p*-level 4 isoparametric shape functions [9]. While *p*-levels 1 and 2 can easily be obtained by hand, high-order elements are more complicated and difficult to keep in the correct order when developing by hand.

The equivalent nodal loads, including pressure, shear, and temperature, were developed for both straight-sided and curved-sided implementation. The results for pressure loads provided a good check of the shape function development (i.e., if the loaded face is assumed opposite of node 4, then you do not expect to see loads either on node 4 or faces and edges with node 4 at one end).

Obviously, *Mathematica* scripts were used to produce the closed-form solutions for stiffness and the error estimator for the straight-sided tetrahedral elements; Appendix A contains an example of a script used to obtain the stiffness matrix while Appendices B and C contain an example for the closed-form error estimator and numerical error estimator, respectively. A sample script used to determine closed-form temperature loads is in Appendix D, while equivalent nodal pressure/shear loads can be found in Appendix E.

Expressions used in the straight-sided numerical development, such as the [*B*] matrix, σ^*, $\hat{\sigma}$, and centroidal stresses, were also produced using *Mathematica*, although

for lower *p*-levels they could possibly be produced by hand. Appendix D contains a Mathematica script used for the numerical implementation.

Mathematica was also used extensively for the curved-sided development. The Jacobian matrices, inverses, and determinants, were obtained by making extensive use dummy variables to reduce expression complexity. This was followed by development of the stiffness matrices, equivalent nodal loads, [*B*] matrix, σ^*, $\hat{\sigma}$, and centroidal stresses. Examples of these scripts can be found in Appendix E.

Numerical values for Gauss quadrature rules based on the compilation by Felippa [41] were obtained by utilizing the *Mathematica* scripts developed in his research. His script for tetrahedral Gauss points was added to a script to produce a separate Fortran 77 fixed-format source code file for each tetrahedral rule set, with values for the points and weights calculated by *Mathematica*, then hard-coded into a set of if-elseif statements. This allowed implementation of the Gauss rules with no calculation of Gauss points required during program execution, and eliminated the possibility of incorrectly typing in the value of a Gauss point or a weight as read from a table (also note that Cools [44] discovered that not all tables of Gauss points in the literature are 100% accurate, and as a result he recalculated Gauss points used in his encyclopedia of cubature).

Usage of *Mathematica* in this instance also made possible simple "copy and paste" into a source code file with little if any further modification required. The source code files produced made it easier to experiment with how many Gauss points were required to achieve convergence for various *p*-level numerical implementations.

Chapter 8 presents the various test and verification problems for both the straight-sided elements and the curve-sided elements. Geometries, loadings, boundary conditions, and theoretical results, including formulas used, are discussed in detail.

CHAPTER 8

TEST AND VERIFICATION PROBLEMS

This chapter discusses the test problems used on the elements developed, including both straight-sided and curved-sided, closed-form and numerical. The test problems were chosen because they posses well-documented solutions. Note that all beam problems have a length-height ratio of 10-1 to ensure slender beam behavior, and all problems use $E = 10^7$ psi and $\nu = 0.33$

8.1 Straight-sided Elements

The testing of straight-sided elements began with the patch test to verify that the elements can represent a constant state of strain, followed by test problems that involved axial loading, bending, torsional loading, and stress concentration factors.

8.1.1 Element Patch Test

The patch test was performed to verify that all four *p*-levels could accurately represent a state of constant strain. Following the procedure outlined by Hughes [57] for the "engineering version" of the patch test, a model of a unit cube was created with boundary conditions and loading as shown in Fig. 8.1. The boundary conditions are as follows: 1 represents displacement in *x*, 2 represents displacement in *y*, and 3 represents

displacement in *z*; thus, a boundary condition of 123 represents displacement constraints in all three directions.

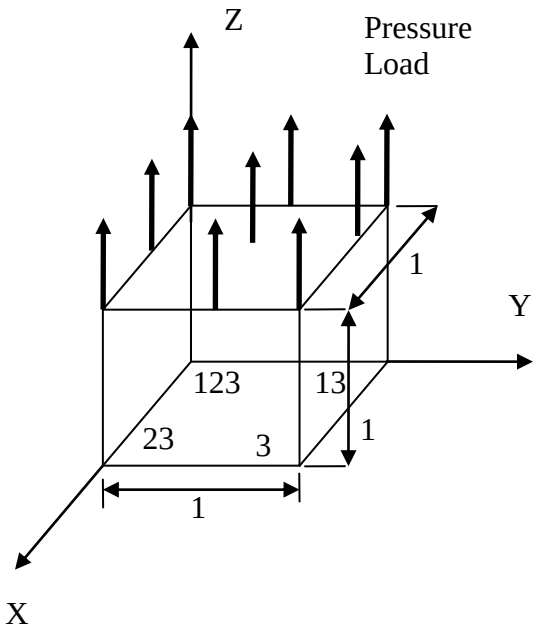


Figure 8.1 Geometry, boundary conditions, and loading used for the patch test.

A uniform pressure loading of 1000 psi was applied to a 1 in. x 1 in. x 1 in. geometry. The mesh was formulated so that all tetrahedral elements met at a common point in the center of the cube; in order to pass the patch test, the stress solution at this common point must be equal to the applied stress.

8.1.2 Axial Loading Test

The next test applied to the straight-sided elements was an axially loaded slender cantilever beam with a uniform pressure of 4000 psi applied to the free end of the beam. The boundary conditions and geometry are shown in Figure 8.2.

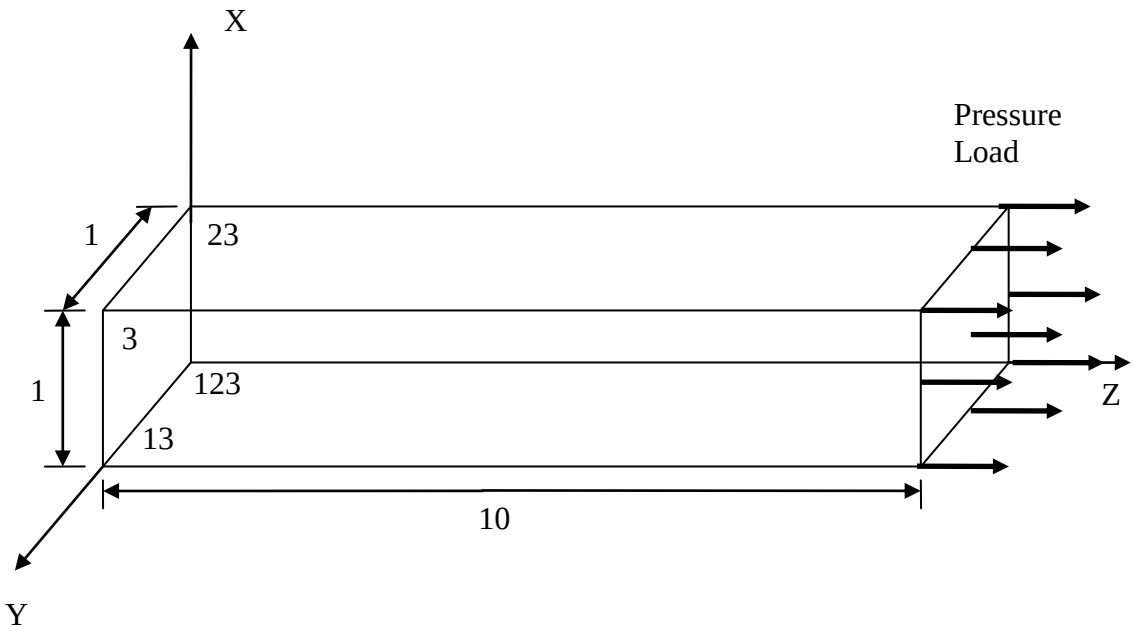


Figure 8.2 Geometry, boundary conditions, and loading used for the axial loading test.

The boundary conditions used, combined with the uniform axial loading, result in a constant stress problem. The error estimator should be equal to zero, and the calculated stresses at each node in the *z*-direction should equal the applied loading, with all other stresses equal to zero. Theoretical solutions for constant strain problems such as this give the following results: $\delta_{tip} = 0.004$ inches and strain energy $U = 8$ in-lbs.

8.1.3 Bending Test

The next test applied to the straight-sided elements was a cantilever beam with a shear load applied to the end, as shown in Fig. 8.3. Boundary conditions were applied to restrain the beam on the *x-y* plane. The beam has an applied uniform shear load of 400 lbs.

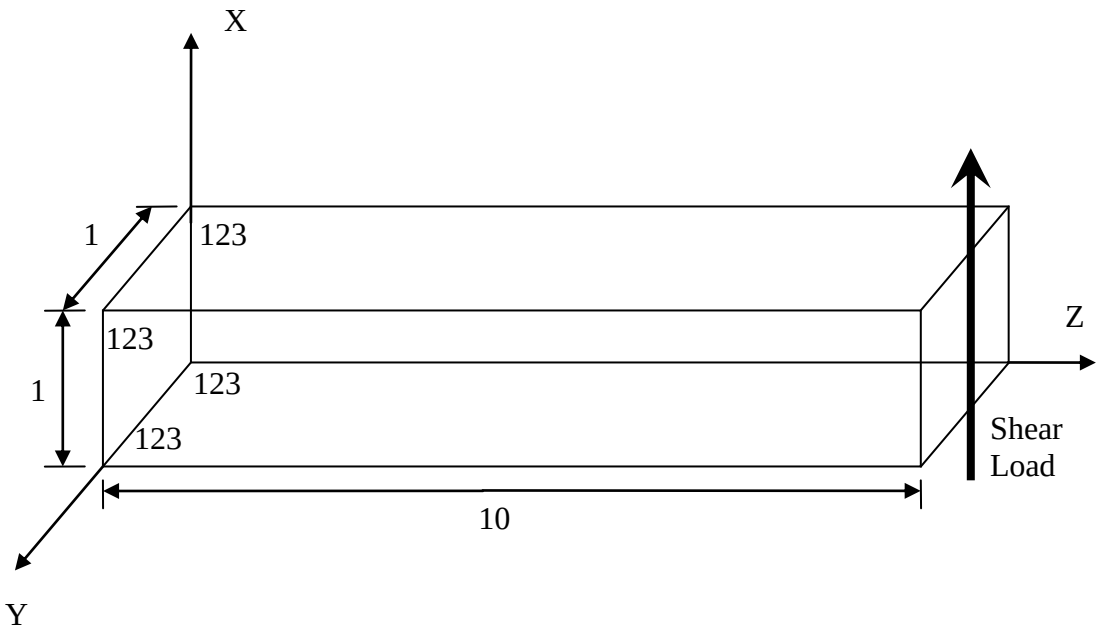


Figure 8.3 Geometry, boundary conditions, and loading used for the beam bending test.

The transverse displacement distribution for this problem is third order in *x*, and the stress distribution is second order in *y*. According to the theory of elasticity [58] and elementary beam theory (for an estimate of stress) [59], the tip displacement and strain energy are given by the equations below, where *F* is the applied force, *L* is the length of the beam, *E* is the modulus of elasticity, *M* is the internal moment, *G* is the shear modulus, *I* is the moment of inertia, *w* is the width of the beam, and *h* is the height of the beam. Substitution of material properties and geometry used in this model results in δ_{tip} = 0.1613 in, *U* = 32.0 in-lbs and σ_{max} = 12,000 psi.

$$\delta_{tip} = {FL^3}/{3EI} + {6FL}/{5Gwh} \tag{8.1}$$

$$U = {F^2L^3}/{6EI} + {3FL^2}/{5Gwh} \tag{8.2}$$

$$\sigma_{z\,max} = {-M\left(h/2\right)}/{I} \tag{8.3}$$

Note that the bending stress was computed in the middle of the beam length in order to avoid the effects of the fixed end constraints.

8.1.4 Torsional Test

The final test applied to the cantilever beam models involves torsional loading of a cantilever beam. A torque was applied to the end of the beam using two shear loads applied at the centroids of the free end element faces, as was done by Shiakolas [4]. This method is illustrated in Fig. 8.4, while Figure 8.5 shows the geometry and boundary conditions used. The beam has applied shear loads of 1950 lbs separated by 0.333 in. for an equivalent torque of 325 in-lbs.

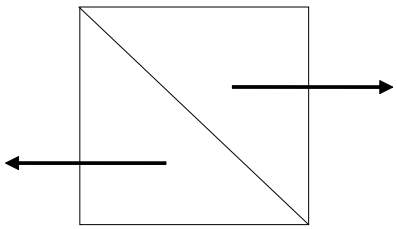


Figure 8.4 Example of how an equivalent moment is applied to simulate torsion.

While the equivalent nodal load vector generated the desired shear loads, it also produced undesired tensile loads which increased the strain energy in the model. The theoretical torsional strain energy U_{tor} can be calculated using the equation that follows [59], where *T* is the applied torque, *L* is the length of the beam, *G* is the shear modulus, *k* is the torsional stiffness constant, and *a* is one-half the width of the beam. The

calculated strain energy was found to be 0.999 in-lbs for an equivalent torque of 325 in-lbs.

$$U_{tor} = {TL^2}/{2Gk} = {TL^2}/{2G(2.25a^4)} \tag{8.4}$$

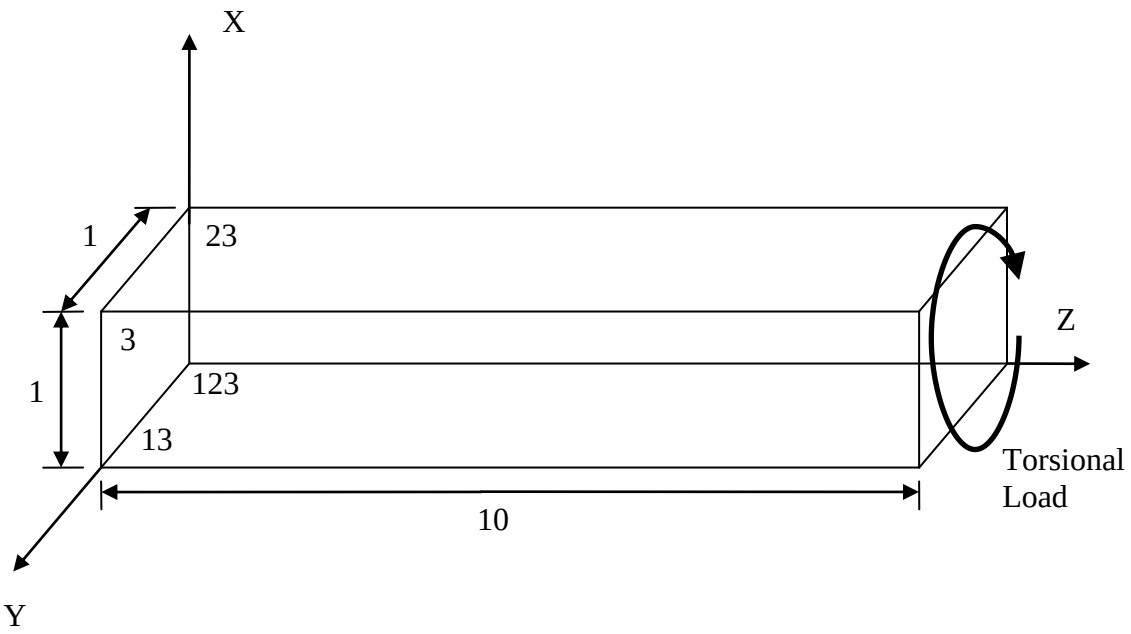


Figure 8.5 Geometry, boundary conditions, and loading used for the torsional load test.

The maximum shear stress (away from the wall) τ_A can be estimated using the simplified formula below [59] where *T* is the equivalent torque, *w* is the width of the beam, and *A* is located at the bottom outermost fiber along the center of the beam.

$$\tau_A = {0.601\ T}/{(0.5w^3)} \tag{8.5}$$

Note that the theory of elasticity behind Eqns. 8.4 and 8.5 assumes the application of a pure torsional load and an equal and opposing moment present at the fixed end of the beam, not a fixed end beam with a torque simulated by equivalent

moments. It is expected that the finite element results will thus differ slightly from the theoretical results.

8.1.5 Uniform Temperature Load

In order to verify the stress recovery and equivalent nodal temperature loads, a uniform temperature load was applied to a cantilever beam with geometry and boundary conditions as shown in Fig. 8.2. This results in a problem with no stress, because the beam is allowed to expand in all directions, and thus all stresses should be zero.

A temperature difference ΔT of 100°F was used with α = 1.0^{-6} /°F. The theoretical displacement can be calculated as follows, where α is the coefficient of thermal expansion, L is the length of the beam, and ΔT is the temperature difference [59]. For this model, given the material properties and temperature difference discussed, the tip displacement should be 0.01 in.

$$\delta_{tip} = \alpha L \Delta T \tag{8.6}$$

If identical boundary conditions are added to the free end of Fig. 8.2, the beam will be constrained at both ends. This results in a problem with no displacement along the *z*-axis, but constant compressive stress along the *z*-axis. This stress is calculated according to the equation below, and results in -1,000 psi.

$$\sigma = \alpha E \Delta T \tag{8.7}$$

8.1.6 Stress Concentration Factor Test

The final test applied to the straight-sided elements involved a thin plate with a hole subjected to a uniform pressure load, creating a stress concentration. Because of the symmetry of both loading and geometry, only one-quarter of the model was used.

Boundary conditions and the pressure loading are illustrated in Fig. 8.6. The stress concentration factor at point A in the figure and the displacement along the loaded edge were used for evaluation.

For this test problem, the applied uniform pressure load was 100 psi. According to Roark's Stress and Strain [60], the stress concentration factor should be 2.16 for this geometry and loading.

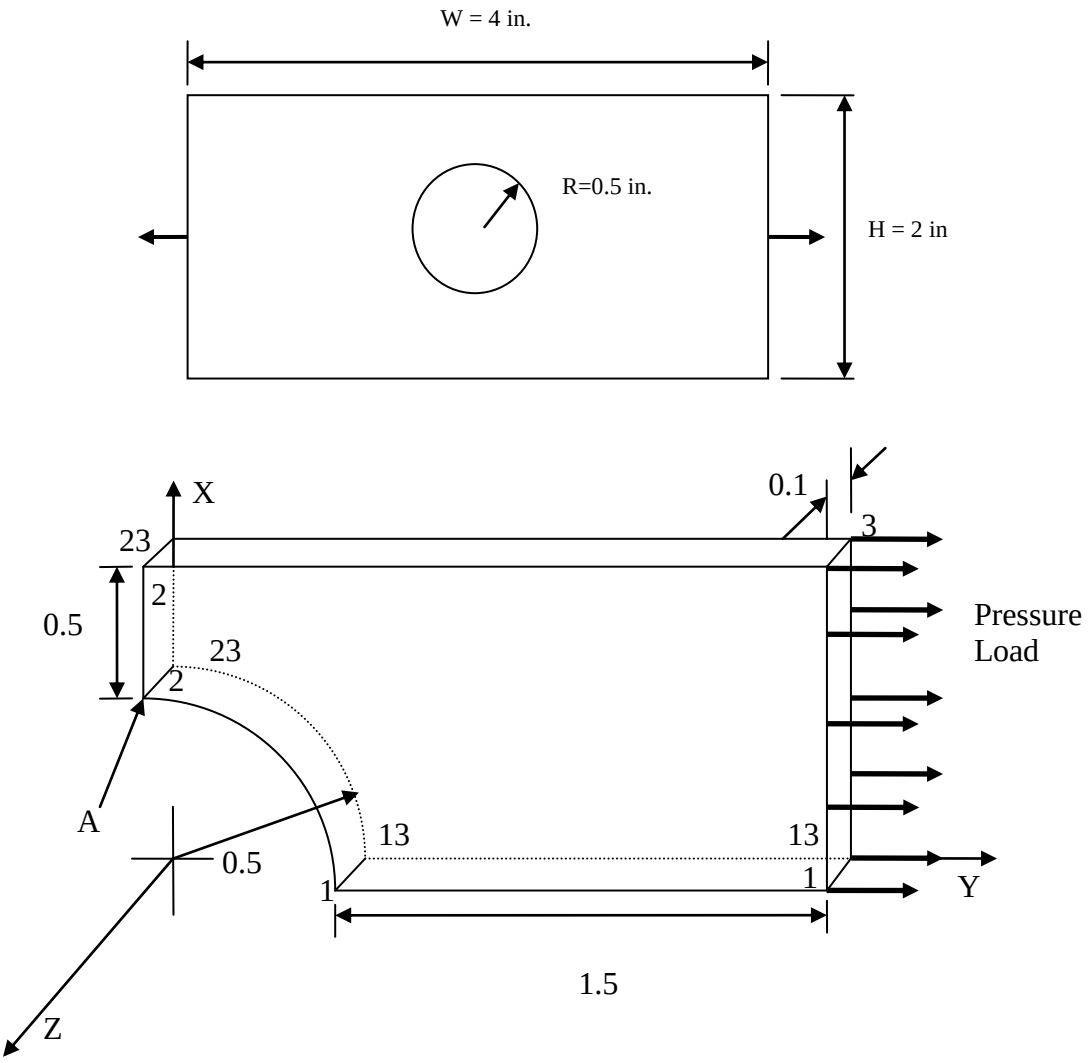


Figure 8.6 Geometry, boundary conditions, and loading used for the stress concentration factor test.

8.2 Curved-sided Elements

The testing of curved-sided elements began with a straight-sided verification problem, followed by an internally pressurized thick-walled cylinder. The cylinder problem was specifically chosen because of its well-documented theoretical solution.

8.2.1 Straight-sided Verification Test

A closed-form, straight-sided element is a subset of the curved element, provided the edge and face nodes have been correctly placed. The first test applied to the curved-sided elements was the axially loaded beam, as shown again in Fig. 8.7 below.

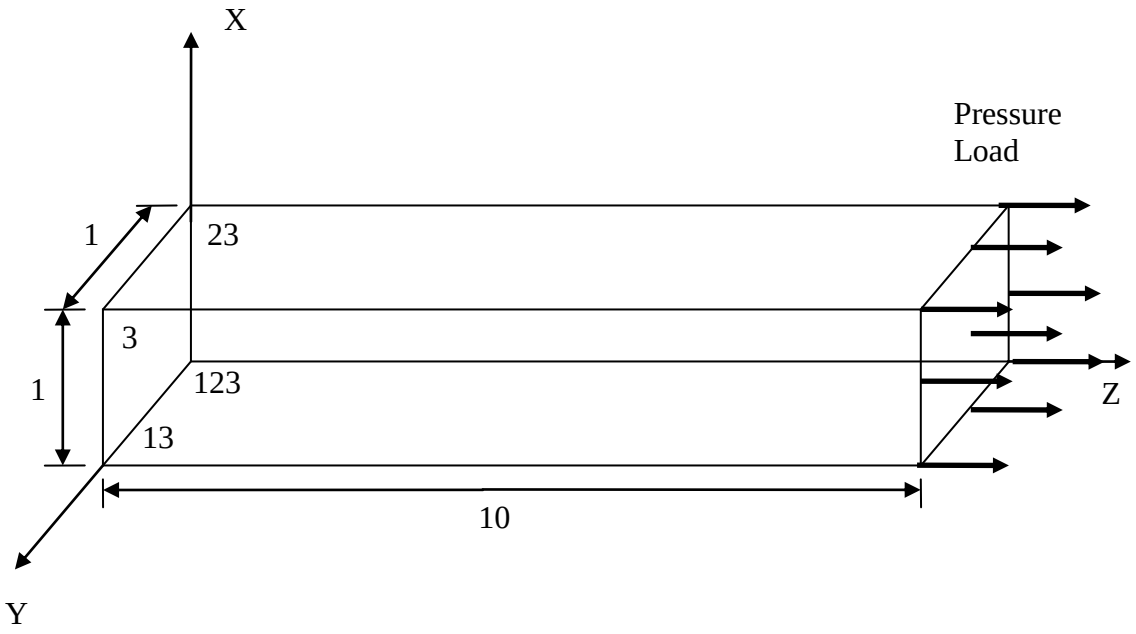


Figure 8.7 Geometry, boundary conditions, and loading used for the verification test.

Material properties, boundary conditions, pressure loadings, etc. were the same as used for the straight-sided element test (see Section 8.1.2). If the curved-sided elements are working correctly, the same results should be produced when this model is solved using the curved-sided element implementation.

This test served a further purpose in allowing for verification of pressure calculations as well as the number of Gauss points used for the numerical pressure implementation. If the calculated nodal loads are evaluated for both the straight-sided and the curved-sided implementation using the same geometry and mesh, the calculated values should be the same.

8.2.2 Thick-walled Cylinder Test

The next test applied to the curved-sided elements was a thick-walled cylinder problem with an internal pressure load, for which a theoretical solution is readily available. When a is the inner radius, b is the outer radius, E is the modulus of elasticity, ν is Poisson's ratio, p_i is the internal pressure, t is the thickness, and r is the radius of interest, and δ_a is the displacement at the inner surface. Eqns. 8.8 – 8.8 represent displacement, strain energy, radial stress, and hoop stress, respectively [59].

$$\delta = {a^2 p_i}\Big/{E(b^2 - a^2)}\left((1 - \nu) + (1 + \nu)\,{b^2}\Big/{r^2}\right) \tag{8.8}$$

$$U_e = \pi a t \delta_a p_i / 4 \tag{8.9}$$

$$\sigma_r = {a^2 p_i}\Big/{(b^2 - a^2)}\left(1 - {b^2}\Big/{r^2}\right) \tag{8.10}$$

$$\sigma_\theta = {a^2 p_i}/{(b^2 - a^2)}\left(1 + {b^2}/{r^2}\right) \tag{8.11}$$

A uniform pressure loading of 20,000 psi was applied to the inner surface, with an inner radius of 2 in., an outer radius of 10 in., and a thickness of 1 in. Note that only one quarter of cylindrical cross section was used to take advantage of the symmetry of the problem, as shown in Fig. 8.8.

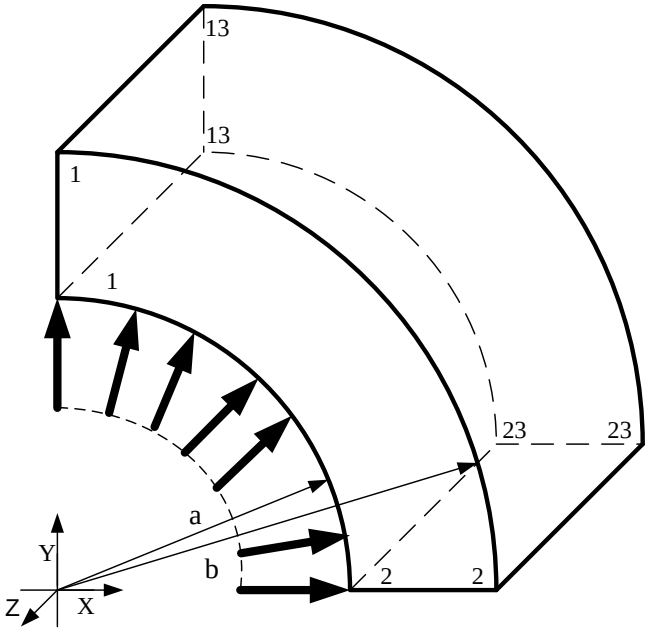


Figure 8.8 Geometry, boundary conditions, and loading for thick-walled cylinder test.

For the given properties, loading, and geometry, the theoretical values for the inner surface and the outer surface of the cylinder, as show in Table 8.1, are obtained.

Table 8.1 Summary of Theoretical Values for Thick-walled Cylinder Problem

	δ ($x10^{-3}$ in)	σ_r (psi)	σ_θ (psi)	Strain Energy
Theoretical Results				177.6
$r = a$	5.65	-20,000	21,667	
$r = b$	1.67	0	1,667	

Computed results for the test problems discussed are presented in the next chapter, along with summaries of compaction results and timing results, conclusions, and suggested areas for future work.

CHAPTER 9

RESULTS AND CONCLUSIONS

This chapter summarizes the results of testing the family of elements developed. Results are summarized for models with straight-sided elements and models comprised of curved-sided elements. The timing comparison between closed-form and numerical implementations is also discussed, as well as results from the code compaction process.

9.1 Straight-sided Elements

The testing of straight-sided elements began with the patch test, to verify that the elements can represent a constant state of strain. That was followed with test problems that involved axial loading, bending loading, torsional loading of a slender beam, and geometries with curved surfaces. All of the test problems had the same material properties: $E = 10^7$ psi, $\nu = 0.33$. Also note that all beam problems involve a length to width ratio of 10 to 1.

9.1.1 Element Patch Test

The patch test, using the geometry and boundary conditions found in Fig. 9,1 was performed to verify that all elements could accurately represent a state of constant strain. A uniform pressure loading of 1000 psi was applied to a 1 in. x 1 in. x 1 in. geometry. The mesh was formulated so that all tetrahedral elements met at a common

point in the center of the cube; in order to pass the patch test, the solution at this common point must be exact, and the stresses must be exact within each element. All four *p*-level hierarchical elements and the fourth order isoparametric element passed the patch test.

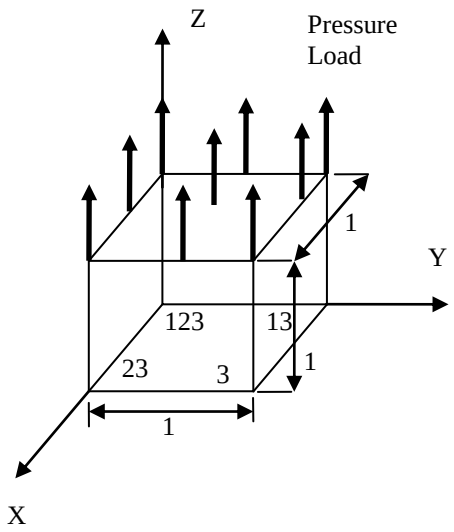


Figure 9.1 Geometry, boundary conditions, and loading used for the patch test.

9.1.2 Axial Loading Test

The next test problem using straight-sided elements was an axially loaded slender cantilever beam with a uniform pressure of 4000 psi at the free end of the beam. See Fig. 9.2 for an illustration of the geometry and boundary conditions.

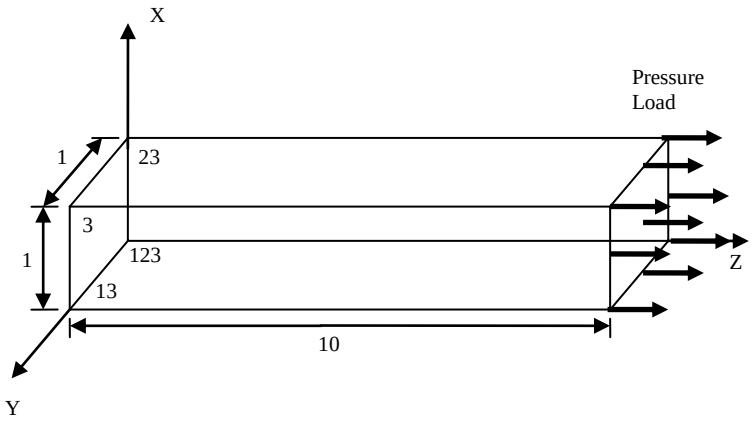


Figure 9.2 Geometry, boundary conditions, and loading used for the axial loading test.

The boundary conditions used, combined with the uniform axial loading, result in a constant stress problem. The results for all four *p*-levels and for the fourth order isoparametric implementation, using a 12 element beam model, matched theory for constant strain problems: δ_{tip} = 0.004 inches, strain energy U = 8 in-lbs, and error estimate was zero for all elements.

Table 9.1 Summary of Axial Loading Results.

	Hierarchical				*Isoparametric*
P-level	*1*	*2*	*3*	*4*	*4*
δ_{tip} *(in.)*	0.004	0.004	0.004	0.004	0.004
Stress (psi)	4000	4000	4000	4000	4000
Strain Energy (lb. – in.)	8.00	8.00	8.00	8.00	8.00

9.1.3 Bending Test

The next test problem for the straight-sided elements was cantilever beam with a shear load applied to the free end, with boundary conditions that restrained motion at the other end of the beam. The uniform shear load was of 400 lbs, applied as shown in Fig. 9.3.

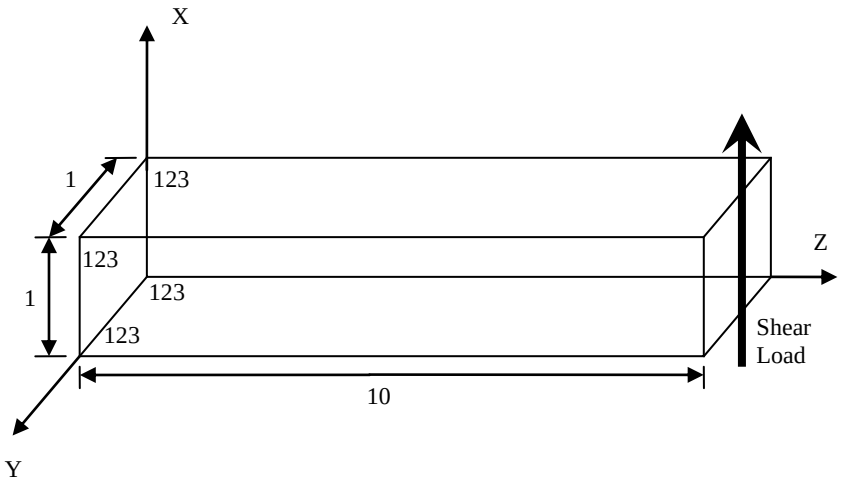
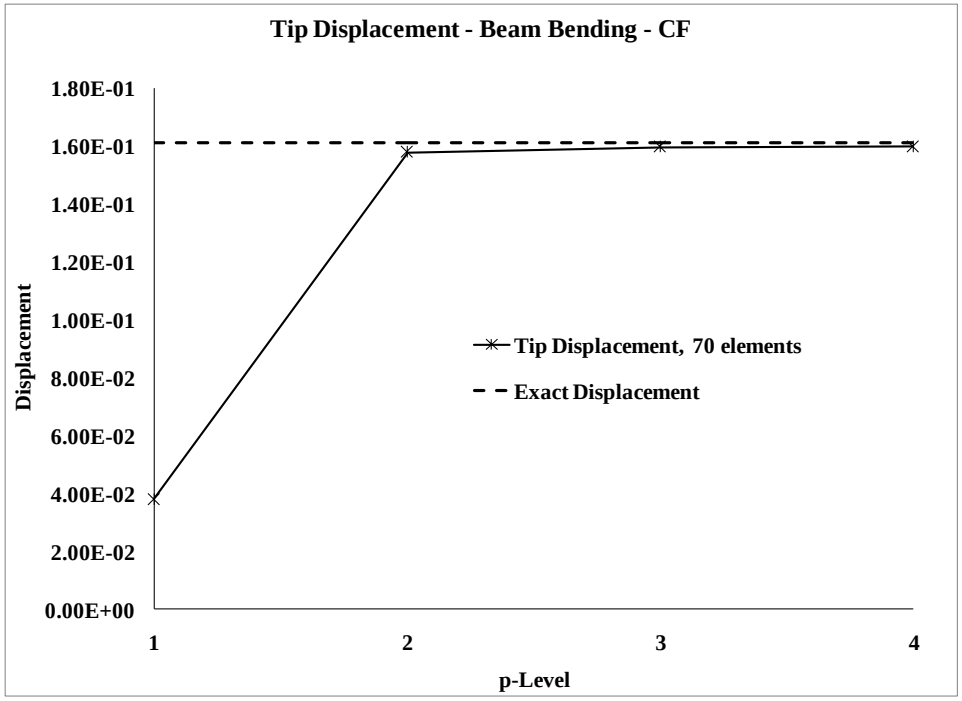


Figure 9.3 Geometry, boundary conditions, and loading used for the beam bending test.

The transverse displacement distribution with length for this problem is third order, and the stress distribution through the thickness is second order; it is expected that *p*-level 2 and higher will perform markedly better for this model than *p*-level 1. According to the theory of elasticity [59], the tip displacement, stress, and strain energy are δ_{tip} = 0.1613 in., $\sigma_{z\,max}$ is 12,000 psi, and U = 32.0 in-lbs, as discussed previously. Note that the stress value was evaluated at the center of the beam, outer edge. A 70 element model was used to obtain the results of testing, as summarized in Figures 9.4 – 9.6.



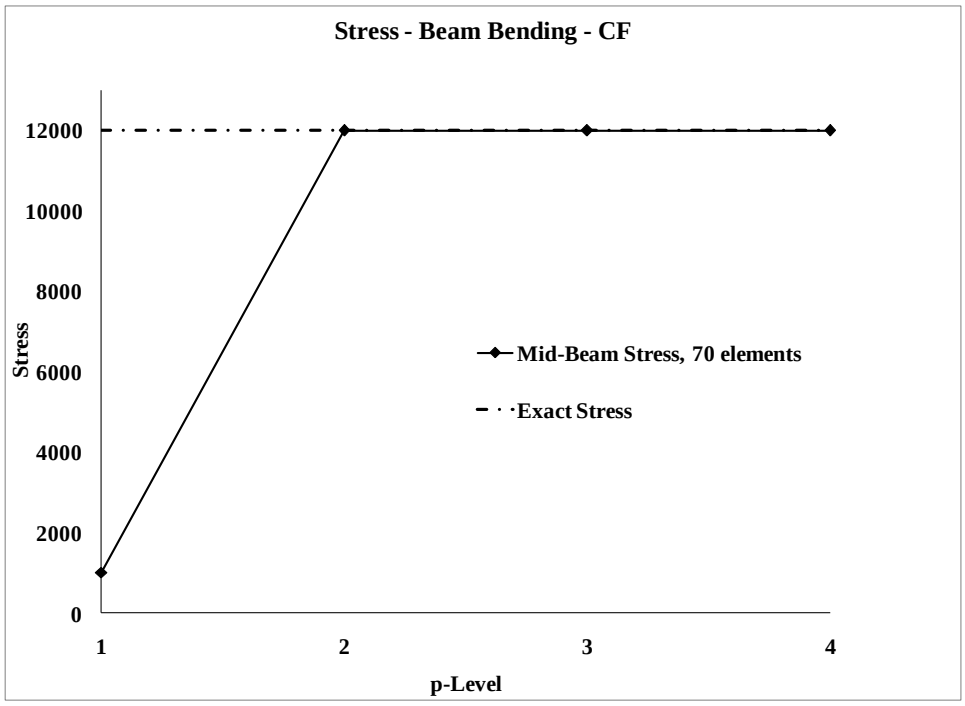
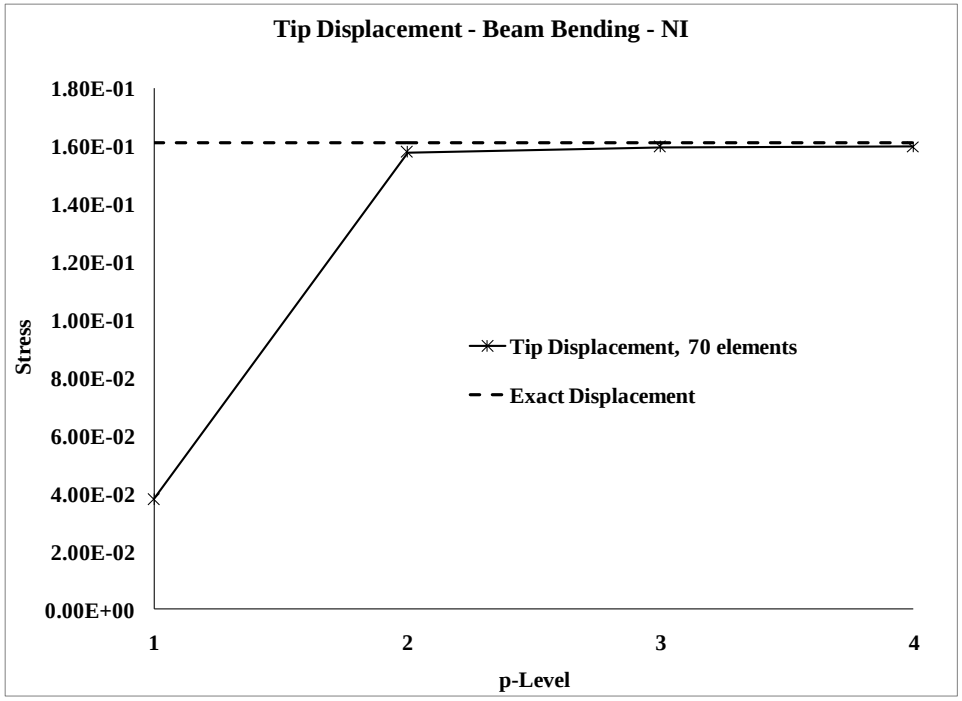


Figure 9.4 Displacement and stress for beam bending using closed-form implementation.



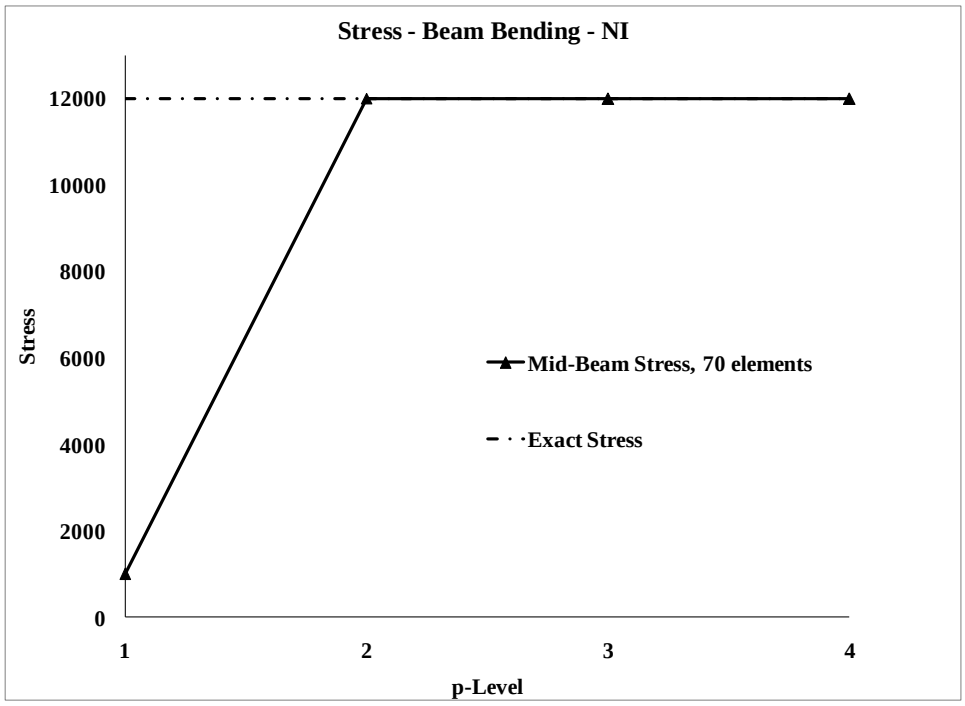
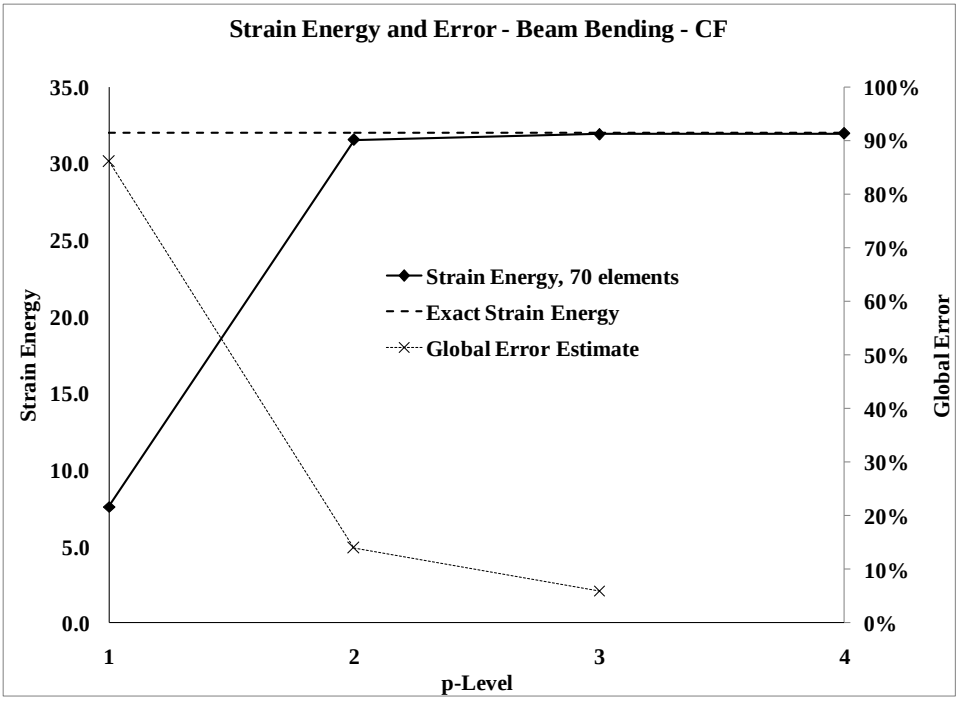


Figure 9.5 Displacement and stress for beam bending using numerical implementation.



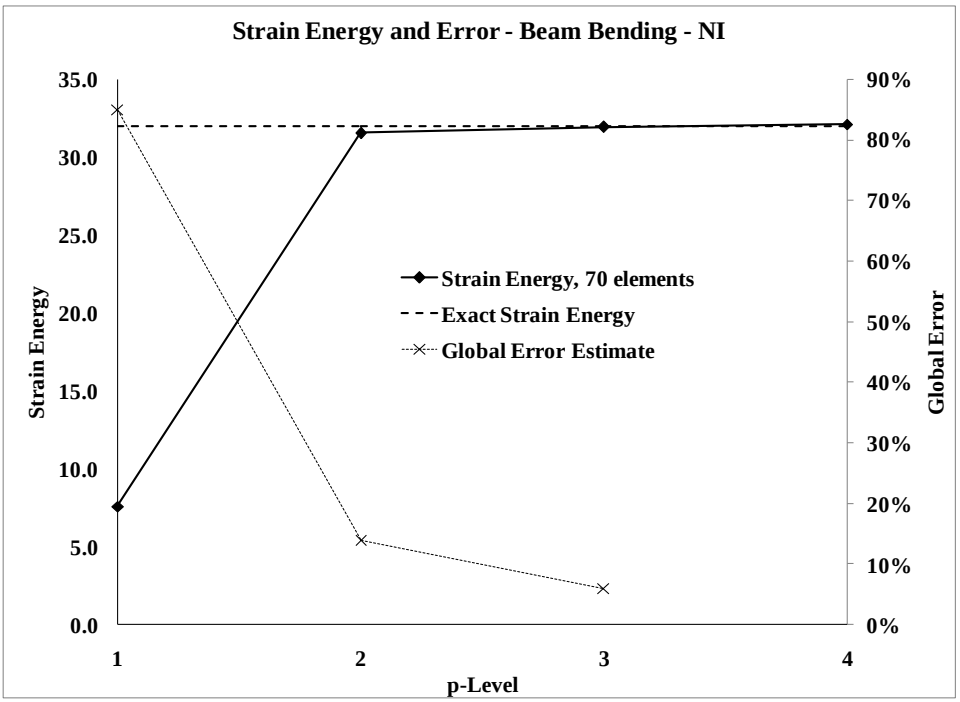


Figure 9.6 Strain energy and global error results, numerical and closed-form, for beam bending.

These results show convergence to a reasonable approximation of the tip displacement and strain energy with *p*-level 4 providing the most accurate values for the tip displacement. Also note that the results for closed-form were essentially the same as those for numerical implementation.

Table 9.2 summarizes the closed-form results for displacement, stress, strain energy, and error estimates for the hierarchical *p*-levels, as well as the fourth order isoparametric *p*-level 4.

Table 9.2 Summary of Closed-form Bending Results.

	Hierarchical				*Isoparametric*
P-level	*1*	*2*	*3*	*4*	*4*
δ_{tip}	0.0379	0.158	0.160	0.160	0.159
Stress (psi)	1010	12,005	12,000	11,999	11,950
Strain Energy (lb. – in.)	7.58	32.0	32.0	32.0	31.8
Error Estimate	86%	14%	6.0%	--	1.1%

9.1.4 Torsional Test

The final test applied to the cantilever beam models involved torsional loading of a cantilever beam. Figure 9.7 shows an example of one of the meshes used in testing. The beam has an applied shear load of 1950 lbs (for an equivalent torque of 325 in-lbs).

The calculated strain energy was found to be 0.999 in-lbs for an applied equivalent torque of 325 in-lbs.

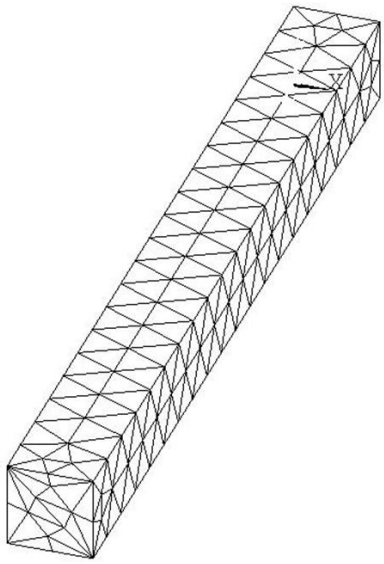
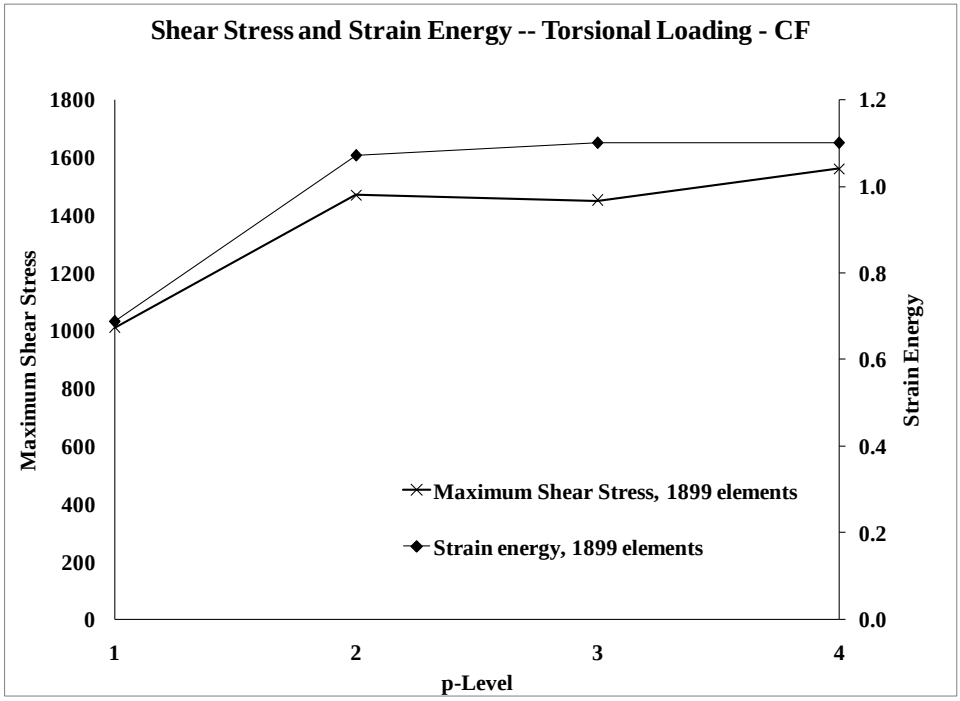


Figure 9.7 Typical mesh used for the torsional loading test.

The mesh used for the torsional loading test had 1899 elements. The maximum shear stress and strain energy results are illustrated in Fig. 9.8. It is apparent that the strain energy values were overestimated, and the theoretical strain energy value thus served as a lower bound. Recall from previous discussion that the theory of elasticity formulations assume the application of a pure torsional load, an equal and opposing moment present at the fixed end of the beam, and unrestrained warping. The actual model implemented was a fixed end beam with a torque simulated by equivalent moments.



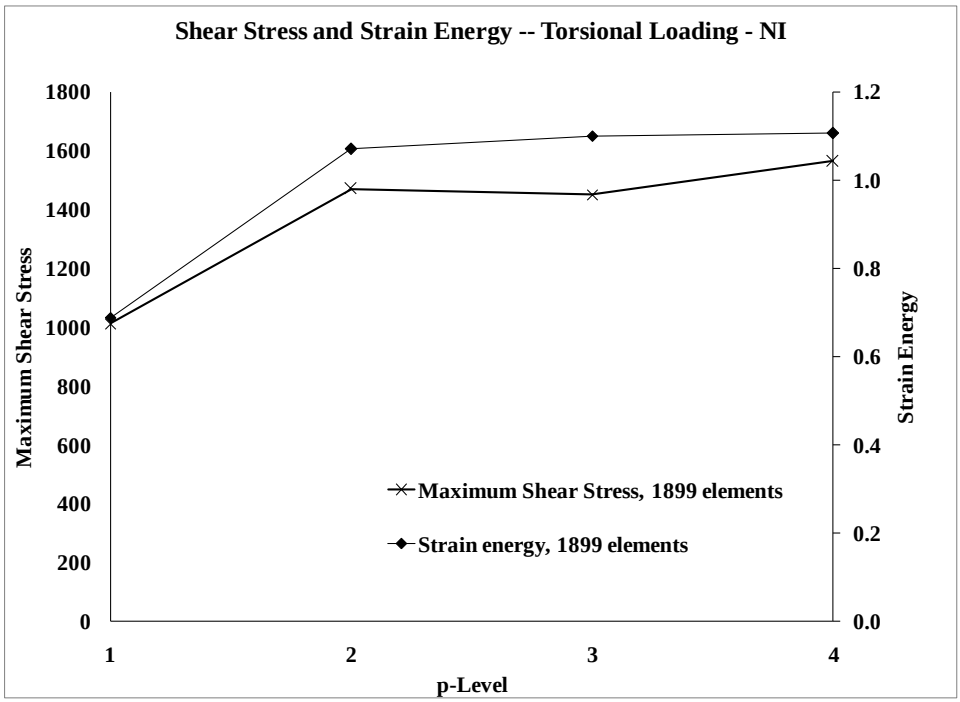


Figure 9.8 Charts showing the shear stress and strain energy for the torsional problem.

For this model, the theoretical strain energy of 0.999 in-lbs. provided a lower bound for the calculated strain energy, while the theoretical shear stress appears to provide an upper bound for the maximum shear stress.

Table 9.3 summarizes the results for strain energy and the error in the energy norm, including results for the fourth order isoparametric implementation. The isoparametric fourth order results shows error of only 6.32% for values similar to hierarchical *p*-level 4, adding credence to the hierarchical *p*-level 4 results. Also note how the strain energy converged to 1.10 in-lbs, which is above the lower estimate provided by theory.

Table 9.3 Summary of Closed-form Results for the Torsional Beam Problem.

	Hierarchical				*Isoparametric*
P-level	*1*	*2*	*3*	*4*	*4*
Strain Energy (lb-in)	0.688	1.07	1.10	1.10	1.10
Energy Norm Error Estimate (lb-in)	0.43	0.07	0.0014	0.0181	0.0088
Maximum Shear Stress (psi)	1,010	1,470	1,450	1,561	1,565
Error Estimate	55%	18%	3.0%	--	6.3%

9.1.5 Uniform Temperature Load

In order to verify the stress recovery and equivalent nodal temperature load, a uniform temperature load was applied to a cantilever beam with geometry and boundary conditions as used in the axially loaded beam problem. This results in a problem with no stress. For the model tested, the tip displacement should be 0.01 in. All four *p*-levels gave this result, with stress at all nodes equivalent to zero.

The second temperature problem involved a beam constrained at both ends, with the same temperature difference applied. All four *p*-levels responded the same: no displacement along the constrained axis, with a compressive stress of -1,000 psi along that axis.

9.1.6 Stress Concentration Factor Test

The final test for straight-sided elements is a thin plate with a hole subjected to a uniform pressure load, creating a stress concentration at A shown in Fig. 9.9, below.

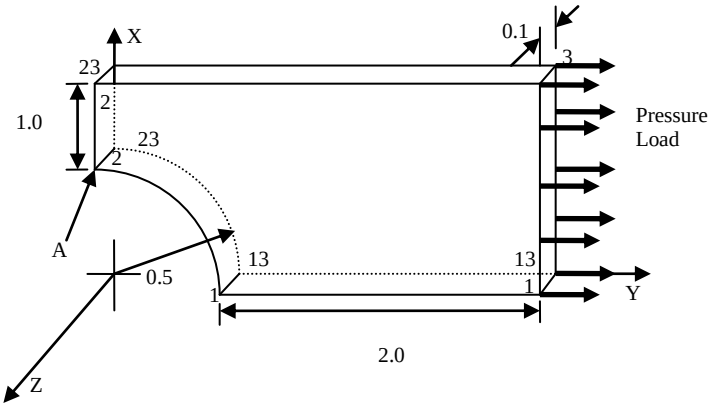


Figure 9.9 Geometry and boundary conditions, and loading used for the stress concentration factor test.

For the loadings applied, Roark's Stress and Strain [60] indicates for this geometry and loading a stress concentration factor of 2.16.

The closed-form results and numerically integrated results were comparable, therefore only the closed-form results are presented. Four models were tested: 53 elements, 295 elements, 449 elements, and 1350 elements. Only results for the 1350 element model are presented. Note that the models were created in ANSYS and modified for use in this research.

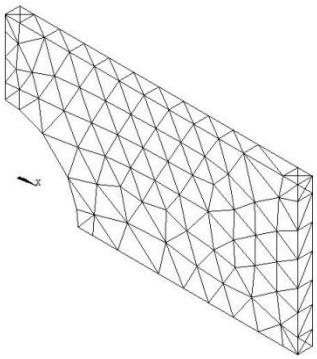


Figure 9.10 Typical plate with a hole mesh using 449 elements.

Figures 9.11 through 9.12 graphically summarize the hierarchical straight-sided, closed-form results obtained using the 1350 element model. Note the convergence of the strain energy in Fig. 9.11.

Luo et. al. [61] noted that the maximum stress will be overestimated for problems where straight-sided elements are used to approximate the curved surface at the center of the plate, because as the *p*-level increases a solution is approximated where the theoretical stress goes to infinity due to the straight-edges and corners at the curved surface.

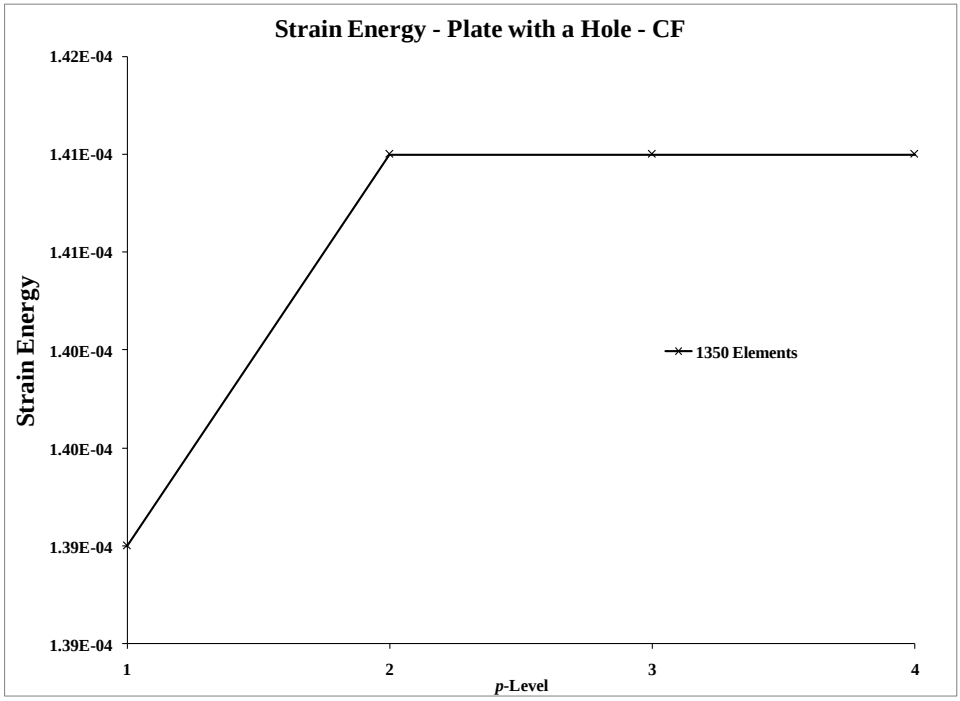


Figure 9.11 Closed-form strain energy results for plate with a hole problem.

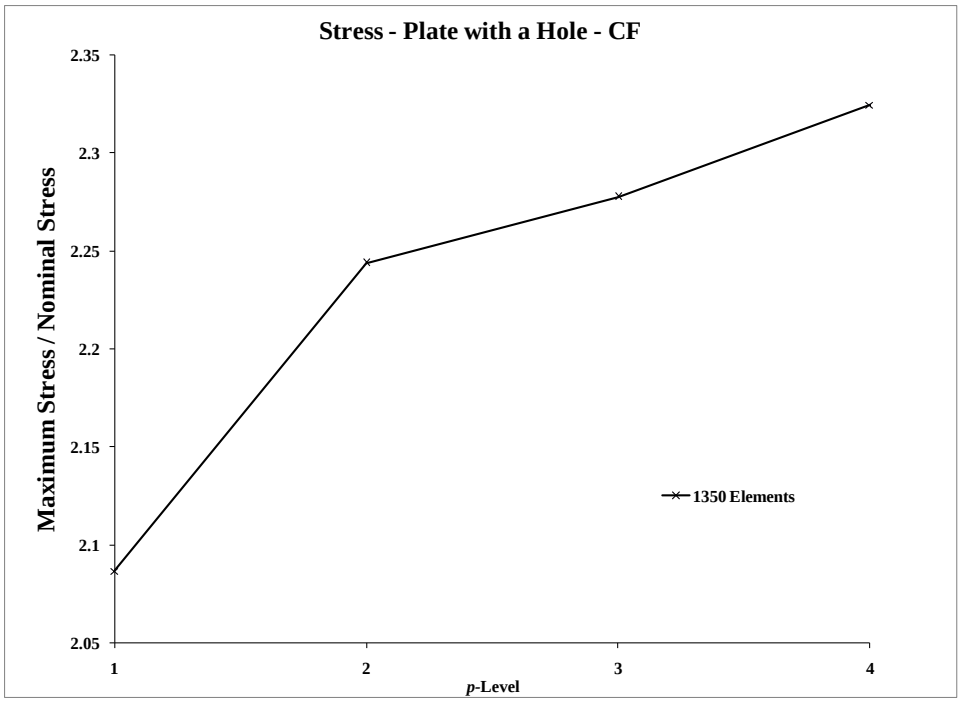


Figure 9.12 Closed-form stress results for plate with a hole problem.

For this model, the ratio of maximum stress to nominal stress is not expected to be the same as the stress concentration factor. The results do show, however, that the elements are behaving as expecting for straight-sided elements approximating a curved surface: the ratio of maximum stress to nominal stress is rising as the *p*-level is increased, as illustrated in Fig. 9.12.

Table 9.4 summarizes the strain energy, ratio maximum stress to nominal stress, error, and displacement of the loaded face for all four hierarchical *p*-levels and isoparametric *p*-level 4.

Table 9.4 Summary of Closed-form Results for the Plate with a Hole Problem.

	Hierarchical				*Isoparametric*
P-level	*1*	*2*	*3*	*4*	*4*
Strain Energy ($x10^{-4}$lb-in)	1.39	1.41	1.41	1.41	1.41
Ratio of Maximum Stress to Nominal Stress	2.08	2.24	2.28	2.32	2.34
Displacement at the Loaded Face ($x10^{-5}$ in)	2.91	2.89	2.91	2.92	2.90
Error Estimate	8.5%	2.7%	0.14%	--	3.08%

9.2 Curved-sided Elements

The testing of curved-sided elements began with a straight-sided verification problem, followed by an internally pressurized thick-walled cylinder, as discussed in the previous chapter.

9.2.1 Straight-sided Verification Test

The verification test for the curved-sided elements, using a 12 element model, gave results identical to those obtained using straight-sided elements. The displacements, stresses, and strain energy were correct, with no measurable error reported.

9.2.2 Thick-walled Cylinder Test

The material properties used in the pressurized thick-walled cylinder test were $E = 10^7$ psi and $\nu = 0.33$, with a uniform pressure loading of 20,000 psi, an inner radius of 2 in., an outer radius of 10 in., and a thickness of 1 in.

A 20 element model was used, and positions where the stress and displacement values were checked are shown in Figure 9.13. Table 9.5 contains results taken at the mid-thickness of the slice compared with the theoretical values at the inner and outer surface. For this set of values, the maximum error in the stresses is 5.86%, and the maximum displacement error is less than 1%.

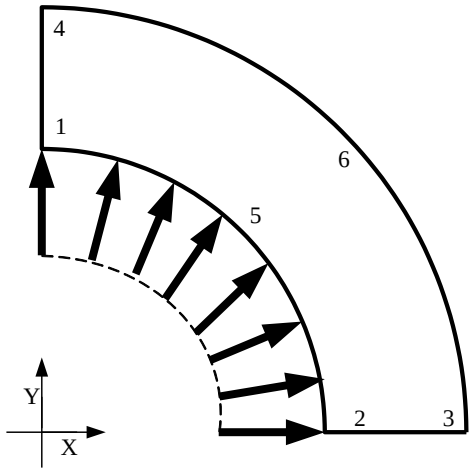


Figure 9.13 Points at which the stress and displacement values were checked for the cylinder problem.

Table 9.5 Summary of Mid-Thickness Results for Thick-walled Cylinder Problem

	δ *(x10^{-3} in)*	*Error*	σ_r *(psi)*	*Error*	σ_θ *(psi)*	*Error*
20 Elements, 1 in. thick						
1	5.69	0.71%	-18,829	5.86%	21,690	0.11%
2	5.69	0.71%	-18,829	5.86%	21,671	0.02%
Theoretical	5.65		-20,000		21,667	
3	1.68	0.59%	93	--	1,730	3.78%
4	1.68	0.59%	95	--	1,738	4.26%
Theoretical	1.67		0		1,667	

9.3 Compaction Results

Table 9.6 summarize the results of compaction and conversion for source code files that are required to implement the hierarchical elements. The initial file sizes are for the Mathematica source code files converted to Fortran 77 fixed-format with white space removed; the compacted file sizes are for the source code files subsequent to

compaction, fixed-format, with white space removed. Note that the results are summarized for hierarchical element for the *p*-levels 2 through 4, and isoparametric fourth order element. The *percent reduction* is calculated as shown in Eq. 9.1, where *Original* is the original (un-compacted) file size, and *Compacted* is the size of the file after compaction.

$$percent\ reduction = \left[(Original - Compacted) / Original \right] x\ 100 \qquad (9.1)$$

Table 9.6 Summary of Compaction Results for Hierarchical Straight-sided Elements.

	Max Percent Reduction	*Min Percent Reduction*
Error *Term1*	16%	3%
Error *Term2*	57%	16%
Error *Term3*	57%	51%
[*K*]	12%	7%
[*B*]	55%	27%
σ^*	21%	8%
$\hat{\sigma}$	80%	67%
Centroidal Stress	81%	64%

P-level 4 elements, of both the hierarchical and isoparametric type, had large file sizes in comparison to lower *p*-levels. For example, term 3 of the error estimator for *p*-level 3 is 88.9 Kb (41.8 Kb after compaction), compared to hierarchical *p*-level 4 at 368 Kb (160 Kb after compaction) and isoparametric *p*-level 4 at 271 Kb (163 Kb after compaction). Note that the tables in Appendix F include compaction results for

the isoparametric curve-sided implementation, which averaged 62% of the original file size.

Inspection of the actual source code file for Term3 of the isoparametric *p*-level 4 element, in uncompacted form, had commands with line continuations in excess of 100, which cannot be compiled using the Silverfrost Fotran compiler used in this research because it is limited to 39 continuation lines. Term 2, also for the isoparametric elements, had several instances of up to 396 continuation lines for one command, which cannot be easily implemented in fixed-form source code without some form of compaction. Even more powerful compilers, such as Sun Microsystems F95, cannot handle such code without modification: F95 allows for 999 continuation lines for free-form source code, but only 19 for fixed-form and 39 for free-form [62].

Without compaction, such files are extremely difficult to implement and are unwieldy to modify by hand. Compaction caused the excessively long instructions in the Term 2 file to be reduced from 396 continuation lines to a maximum of 52 continuation lines, which can be easily implemented.

Table 9.7 summarizes the original required memory for the higher order elements compared to the required memory subsequent to compaction. Note that for *p*-level 4 the memory required after compaction is almost half the 1.4 Mb originally required.

Note that compaction was also performed on files used for the numerical implementation. Timing comparisons were performed between straight-sided closed-form and numerical implementations. By performing compaction on the numerical

source code files, any bias in timing due to the run-time reduced by compaction was virtually eliminated.

Table 9.7 Required Memory for Higher Order Elements

Element Type	*Original Memory, Kb*	*Compacted Memory, Kb*	*% of Original Size*
Straight-sided Hierarchical p-level 3	817.2	407.97	50%
Straight-sided Hierarchical p-level 4	1395.1	741.4	53%
Straight-sided Isoparametric p-level 4	1108.8	813.3	73%
Curve-sided Isoparametric p-level 4	306.2	190	62%

9.4 Closed-form and Numerical Timing Comparisons

A major objective of this work as the evaluation of potential computation speed gains resulting the use of closed-form element expressions as opposed to numerically integrated element quantities. Thus, timing comparisons between the closed-form and the numerical integration implementation were performed to determine if the closed-form implementation is more efficient. The Fortran programs were built using Silverfrost Fortran 95©, and the timing for the stiffness matrix and error estimation was performed using its TimingAnalysis profiling tool. This tool allows for accurate measurement of CPU time for each subroutine and function in a Fortran program, and, before reporting the timing values, removes any overhead timing effects caused by the

timing itself. Inclusive timing values report the time spent in a subroutine as well as all other routines it calls [63].

The values in Table 9.8 below presents results of timing evaluation based on the procedures developed in this work. Inclusive timing for numerical routines divided by inclusive timing for the closed-form routines are given. For all *p*-levels, the closed-form stiffness implementation was found to be more efficient than the numerical implementation. Closed-form error estimation was also found to be more efficient for *p*-levels 3 and 4.

Speed up ratios for the element stiffness matrix evaluation range from 4 to 76, that is, for *p*-level 1, the closed form element stiffness evaluation is 4 times faster than the numerically integrated equivalent. For the higher order elements, the speed ratio increased to 76 because of the larger number of matrix terms involved.

Table 9.8 Timing Results for Straight-sided Elements.

P-Level	Error Estimation Num/CF	Stiffness Num/CF
1	0.75	4
2	0.74	37
3	2.9	68
4	5.9	76

9.5 Conclusions and Summary

Closed-form implementations of straight-sided tetrahedral element stiffness matrices were found to have deflection and stress results of high accuracy. This is to be

expected since no quadrature is introduced in the process. The closed-form stiffness matrix calculations were more efficient by a factor of 4 for $p = 1$ and a factor of 76 for $p = 4$. For p-levels 3 and 4, closed-form implementations were found to be more efficient for error estimation by a factor of up to 5.9.

The straight-sided isoparametric fourth order closed-form elements performed as well, and provided results comparable to the p-level 4 hierarchical elements. The results of testing numerically integrated isoparametric curved elements demonstrated the accuracy of this formulation. Since both the straight-sided and curved-sided elements are based on the same shape functions interelement continuity is preserved, and thus straight-sided and curved-sided elements can be successfully combined in a single model. Furthermore, the demonstrated efficiency of the closed-form straight-sided elements combined with the curved-sided elements provides the most computationally efficient model for element stiffness evaluation. This supports earlier work [64] and extends its application to hierarchical and fourth order isoparametric element formulations.

The results of compaction demonstrated that the algorithm used allows large source code files to be substantially reduced in size. Benefits include smaller storage requirements, smaller executables, and improved execution speed.

9.6 Recommendations for Future Work

The use of blending functions for hierarchical curved-sided elements [22], to allow for combined straight-sided and curved-sided hierarchical models, would be an

advisable extension of the current research and would most likely entail the development of a mesh generator tailored specifically for the blending function method.

Other areas include allowing multiple *p*-level elements in a single model, automated *p*-refinement, and graphical post-processing capabilities that would allow plots of displacements, element errors, and stresses. Also, an graphical user-interface for the pre-processor that allows users to apply loads using "point-and-click" would be highly desirable.

The efficiency of closed-form element formulation has been demonstrated. A natural extension of this work would include consideration of *p*-levels 5 and greater, for either hierarchical or isoparametric elements, to determine if closed-form procedure continues to be more efficient and to determine if expression growth can be controlled by compaction to produce files of manageable size. Finally, an improved error estimator for *p*-level 4 and higher hierarchical elements may prove useful.

APPENDIX A

P-LEVEL 4 STIFFNESS GENERATION

```
(* Order of the shape functions *)
p = 4

i3 = IdentityMatrix[3];
A = {
{1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 1, 0, 1, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 0, 0, 1, 0, 0}
};

(* ELA will be the D matrix used in Nambiar's Calculations *)
ELA={
{e1, e2, e2, 0, 0, 0}, {e2, e1, e2, 0, 0, 0}, {e2, e2, e1, 0, 0, 0},
{0, 0, 0, e3, 0, 0}, {0, 0, 0, 0, e3, 0}, {0, 0, 0, 0, 0, e3}
};

L4 = 1 - L1 - L2 - L3;

(* Initialize shape function matrix N for 35 nodes *)
nn = Table[0, {35}];
nTot=35

(* 0, 1, 2, . . . *)
LPoly = {1, x, 1/2 (3 x ^ 2 - 1), 1/2 (5 x ^ 3 - 3 x ), 1/8 (35 x ^ 4 - 30 x ^ 2 + 3), 1/8 (63
x ^ 5 - 70 x ^ 3 + 15 x) };

Legendre[i_] := LPoly[[i]];

Ei[i_, t1_, t2_] := - 8 Sqrt[4 i + 2] / (i (i + 1)) (D[Legendre[i+1],x])/.x->(t2 - t1)
Fi[r1_, r2_, t1_, t2_, t3_] := (Legendre[r1+1]/.x->(t2 - t1))*(Legendre[r2+1]/.x->(2 t3 -
1))

Print["Creating vertices"];
(* Coordinate system:  eta, zeta, xi *)
(* Four nodal shape functions, one for each node *)
nn[[1]] = L1;  nn[[2]] = L2;  nn[[3]] = L3;  nn[[4]] = L4;

(* Edge modes *)
Print["Creating edge modes"];
nn[[5]] = Simplify[L2 L3 Ei[1,L2,L3]];
nn[[6]] = Simplify[L1 L3 Ei[1,L1,L3]];
nn[[7]] = Simplify[L1 L2 Ei[1,L1,L2]];
nn[[8]] = Simplify[L1 L4 Ei[1,L1,L4]];
nn[[9]] = Simplify[L2 L4 Ei[1,L2,L4]];
nn[[10]] = Simplify[L3 L4 Ei[1,L3,L4]];
```

```
nn[[11]] = Simplify[L2 L3 Ei[2,L2,L3]];
nn[[12]] = Simplify[L1 L3 Ei[2,L1,L3]];
nn[[13]] = Simplify[L1 L2 Ei[2,L1,L2]];
nn[[14]] = Simplify[L1 L4 Ei[2,L1,L4]];
nn[[15]] = Simplify[L2 L4 Ei[2,L2,L4]];
nn[[16]] = Simplify[L3 L4 Ei[2,L3,L4]];

(* Face modes *)
nn[[17]] = Simplify[ L2 L3 L4 Fi[0, 0, L2, L3, L4 ]];
nn[[18]] = Simplify[ L3 L4 L1 Fi[0, 0, L1, L3, L4 ]];
nn[[19]] = Simplify[ L4 L1 L2 Fi[0, 0, L1, L2, L4 ]];
nn[[20]] = Simplify[ L1 L2 L3 Fi[0, 0, L1, L2, L3 ]];

nn[[21]] = Simplify[L2 L3 Ei[3,L2,L3]];
nn[[22]] = Simplify[L1 L3 Ei[3,L1,L3]];
nn[[23]] = Simplify[L1 L2 Ei[3,L1,L2]];
nn[[24]] = Simplify[L1 L4 Ei[3,L1,L4]];
nn[[25]] = Simplify[L2 L4 Ei[3,L2,L4]];
nn[[26]] = Simplify[L3 L4 Ei[3,L3,L4]];

(* Face modes *)
nn[[27]] = Simplify[ L2 L3 L4 Fi[1, 0, L2, L3, L4 ]];
nn[[28]] = Simplify[ L3 L4 L1 Fi[1, 0, L1, L3, L4 ]];
nn[[29]] = Simplify[ L4 L1 L2 Fi[1, 0, L1, L2, L4 ]];
nn[[30]] = Simplify[ L1 L2 L3 Fi[1, 0, L1, L2, L3 ]];
nn[[31]] = Simplify[ L2 L3 L4 Fi[0, 1, L2, L3, L4 ]];
nn[[32]] = Simplify[ L3 L4 L1 Fi[0, 1, L1, L3, L4 ]];
nn[[33]] = Simplify[ L4 L1 L2 Fi[0, 1, L1, L2, L4 ]];
nn[[34]] = Simplify[ L1 L2 L3 Fi[0, 1, L1, L2, L3 ]];

(* Bubble mode *)
nn[[35]] = L1 L2 L3 L4;

(* From Shiakolas *)

(* Put into appropriate format for use with developed equations *)
NT = Flatten[Table[i3 * nn[[i]], {i,1,nTot}],1];
NN = Transpose[NT];
NN = Simplify[NN];

(* Form the R matrix *)
RL1 = D[NN,L1];RL2 = D[NN,L2];RL3 = D[NN,L3];
R = Flatten[{RL1, RL2, RL3}, 1];
```

```
R = Simplify[R];
Print["Dimensions of R: ", Dimensions[R]];

(* Generate the P matrix *)
g1 = Transpose[Flatten[{i3 c11, i3 c12, i3 c13}, 1]];
g2 = Transpose[Flatten[{i3 c21, i3 c22, i3 c23}, 1]];
g3 = Transpose[Flatten[{i3 c31, i3 c32, i3 c33}, 1]];
GAM = Flatten[{g1, g2, g3}, 1];
P = A.GAM;
Clear[g1, g2, g3, A, GAM];
Print["Dimensions of P: ", Dimensions[P]];

(* Generate the G matrix *)
G = Simplify[Transpose[P].ELA.P];
Print["Dimensions of GG: ",Dimensions[G]];
(* Create a temp matrix for G *)
GG = G;

(* Redefine G to avoid expression growth . . . *)
G = {
{g11, g12, g13, g14, g15, g16, g17, g18, g19},
{g12, g22, g23, g24, g25, g26, g27, g28, g29},
{g13, g23, g33, g34, g35, g36, g37, g38, g39},
{g14, g24, g34, g44, g45, g46, g47, g48, g49},
{g15, g25, g35, g45, g55, g56, g57, g58, g59},
{g16, g26, g36, g46, g56, g66, g67, g68, g69},
{g17, g27, g37, g47, g57, g67, g77, g78, g79},
{g18, g28, g38, g48, g58, g68, g78, g88, g89},
{g19, g29, g39, g49, g59, g69, g79, g89, g99}
};

(* Define dummy variables and symbolic integration rule *)
mult = L1^t L2^t L3^t;
rule = {L1^aa_.L2^ab_.L3^ac_.->aa!ab!ac!/(aa+ab+ac+3)!};
(* Generate the element stiffness matrix *)
Print["Calculate K"];
K = Transpose[R].G.R;
K = K mult;
K = Expand[K];
Print["Applying rule and setting t = 0"];
K = K/.rule;
K = K/.t->0;
Print["Simplifying K"];
K = Simplify[K];
```

```
Print["Dimensions of K: ",Dimensions[K]];

(* Save the upper triangular part of the stiffness matrix in a file in the FORTRAN
language syntax *)
counter=1;
Print["Writing the Stiffness file . . . : / "];
strm=OpenWrite["k4.f90",FormatType->FotranForm, PageWidth->70];
str1 = "akqst("; str2 =")=";str3 = "";
For[
        ii=1, ii<=3*nTot,
        For[
                jj=ii, jj<=3*nTot,

        WriteString[strm,"akqst("<>ToString[counter]<>")=("<>ToString[FortranForm[
K[[ii,jj]]]]<>")*det"<>"\n"];
                counter++;
                jj++
      ];
        ii++
];
Close[strm];
```

APPENDIX B

ISOPARAMETRIC FOURTH ORDER CLOSED-FORM ERROR ESTIMATION

NQUINITEEORIG.TXT

```
(* Order of the shape functions *)
p = 4;
nTot=35;
Print["p = ",p];
i3 = IdentityMatrix[3];
i6 = IdentityMatrix[6];
A = {
{1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 1, 0, 1, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 0, 0, 1, 0, 0, 0}
};

L4 = 1 - L1 - L2 - L3;

(* Initialize shape function matrix N for 20 nodes *)
nn = Table[0, {nTot}];

(* Four nodal shape functions, one for each node *)
nn[[1]] = 32/3 (L1 - 3/4) (L1 - 1/2) (L1 - 1/4) L1;
nn[[2]] = 32/3 (L2 - 3/4) (L2 - 1/2) (L2 - 1/4) L2;
nn[[3]] = 32/3 (L3 - 3/4) (L3 - 1/2) (L3 - 1/4) L3;
nn[[4]] = 32/3 (L4 - 3/4) (L4 - 1/2) (L4 - 1/4) L4;

(* Edge modes *)
Print["Creating edge modes"];
nn[[5]]  = 128/3 (L1 - 1/2) (L1 - 1/4) L1 L2;
nn[[6]]  = 128/3 (L2 - 1/2) (L2 - 1/4) L1 L2;
nn[[7]]  = 64 (L1 - 1/4) (L2 - 1/4) L1 L2;
nn[[8]]  = 128/3 (L2 - 1/2) (L2 - 1/4) L2 L3;
nn[[9]]  = 128/3 (L3 - 1/2) (L3 - 1/4) L2 L3;
nn[[10]] = 64 (L3 - 1/4) (L2 - 1/4) L3 L2;
nn[[11]] = 128/3 (L1 - 1/2) (L1 - 1/4) L1 L3;
nn[[12]] = 128/3 (L3 - 1/2) (L3 - 1/4) L1 L3;
nn[[13]] = 64 (L3 - 1/4) (L1 - 1/4) L3 L1;
nn[[14]] = 128/3 (L3 - 1/2) (L3 - 1/4) L4 L3;
nn[[15]] = 128/3 (L4 - 1/2) (L4 - 1/4) L4 L3;
nn[[16]] = 64 (L3 - 1/4) (L4 - 1/4) L3 L4;
nn[[17]] = 128/3 (L4 - 1/2) (L4 - 1/4) L4 L1;
nn[[18]] = 128/3 (L1 - 1/2) (L1 - 1/4) L4 L1;
nn[[19]] = 64 (L1 - 1/4) (L4 - 1/4) L1 L4;
nn[[20]] = 128/3 (L4 - 1/2) (L4 - 1/4) L4 L2;
nn[[21]] = 128/3 (L2 - 1/2) (L2 - 1/4) L4 L2;
nn[[22]] = 64 (L2 - 1/4) (L4 - 1/4) L2 L4;
```

```
(* Face modes *)
Print["Creating face modes"];
nn[[23]] = 128 L1 L2 L3 (L1 - 1/4);
nn[[24]] = 128 L1 L2 L3 (L2 - 1/4);
nn[[25]] = 128 L1 L2 L3 (L3 - 1/4);
nn[[26]] = 128 L2 L3 L4 (L2 - 1/4);
nn[[27]] = 128 L2 L3 L4 (L3 - 1/4);
nn[[28]] = 128 L2 L3 L4 (L4 - 1/4);
nn[[29]] = 128 L1 L2 L4 (L1 - 1/4);
nn[[30]] = 128 L1 L2 L4 (L2 - 1/4);
nn[[31]] = 128 L1 L2 L4 (L4 - 1/4);
nn[[32]] = 128 L1 L3 L4 (L1 - 1/4);
nn[[33]] = 128 L1 L3 L4 (L3 - 1/4);
nn[[34]] = 128 L1 L3 L4 (L4 - 1/4);
nn[[35]] = 256 L1 L2 L3 L4;

(* From Shiakolas *)

(* Put into appropriate format for use with developed equations *)
(* We use i6 because we have six stresses possible per node *)
NT = Flatten[Table[i6 * nn[[i]], {i,1,nTot}],1];
NN = Transpose[NT];

(* Form the R matrix *)
NTi = Flatten[Table[i3 * nn[[i]], {i,1,nTot}],1];
NNi = Transpose[NTi];

RL1 = D[NNi,L1];RL2 = D[NNi,L2];RL3 = D[NNi,L3];
R = Flatten[{RL1, RL2, RL3}, 1];
Print["Dimensions of R: ", Dimensions[R]];

(* Generate the P matrix *)
Print["Generating GAM"];
g1 = Transpose[Flatten[{i3 c11, i3 c12, i3 c13}, 1]];
g2 = Transpose[Flatten[{i3 c21, i3 c22, i3 c23}, 1]];
g3 = Transpose[Flatten[{i3 c31, i3 c32, i3 c33}, 1]];
GAM = Flatten[{g1, g2, g3}, 1];
Print["Generating P matrix as A.GAM"];
P = A.GAM;
Print["Dimensions of P: ", Dimensions[P]];
(* Note that Transpose[P].ELA.P does provide G as a 9x9 matrix *)

(* We know that B = P R *)
```

```
B = P.R;
Print["Dimensions of B: ",Dimensions[B]];
```

ERRQUINTIC.TXT

```
(* Modified from Shiakolas research *)

(* com:  compliance matrix, u: nodal displacement vector; sav: nodal averaged stresses
*)
com = { { cm11, cm12, cm12, 0, 0, 0},
        { cm12, cm11, cm12, 0, 0, 0},
        { cm12, cm12, cm11, 0, 0, 0},
        { 0, 0, 0, cm13, 0, 0},
        { 0, 0, 0, 0, cm13, 0},
        { 0, 0, 0, 0, 0, cm13} };

cmm = {cm11, cm12, cm13 };

u =     {
u1, v1, w1, u2, v2, w2, u3, v3, w3, u4, v4, w4, u5, v5, w5,
u6, v6, w6, u7, v7, w7, u8, v8, w8, u9, v9, w9, u10, v10, w10,
u11, v11, w11, u12, v12, w12, u13, v13, w13, u14, v14, w14, u15, v15, w15,
u16, v16, w16, u17, v17, w17, u18, v18, w18, u19, v19, w19, u20, v20, w20,
u21, v21, w21, u22, v22, w22, u23, v23, w23, u24, v24, w24, u25, v25, w25,
u26, v26, w26, u27, v27, w27, u28, v28, w28, u29, v29, w29, u30, v30, w30,
u31, v31, w31, u32, v32, w32, u33, v33, w33, u34, v34, w34, u35, v35, w35}

cc = { {c11, c12, c13}, {c21, c22, c23}, {c31, c32, c33} };

(* Should be for p = 4,35 *)
sav = { sx1, sy1, sz1, tx1, ty1, tz1, sx2, sy2, sz2, tx2, ty2, tz2,
        sx3, sy3, sz3, tx3, ty3, tz3, sx4, sy4, sz4, tx4, ty4, tz4,
        sx5, sy5, sz5, tx5, ty5, tz5, sx6, sy6, sz6, tx6, ty6, tz6,
        sx7, sy7, sz7, tx7, ty7, tz7, sx8, sy8, sz8, tx8, ty8, tz8,
        sx9, sy9, sz9, tx9, ty9, tz9, sx10, sy10, sz10, tx10, ty10, tz10,
        sx11, sy11, sz11, tx11, ty11, tz11, sx12, sy12, sz12, tx12, ty12, tz12,
        sx13, sy13, sz13, tx13, ty13, tz13, sx14, sy14, sz14, tx14, ty14, tz14,
        sx15, sy15, sz15, tx15, ty15, tz15, sx16, sy16, sz16, tx16, ty16, tz16,
        sx17, sy17, sz17, tx17, ty17, tz17, sx18, sy18, sz18, tx18, ty18, tz18,
        sx19, sy19, sz19, tx19, ty19, tz19, sx20, sy20, sz20, tx20, ty20, tz20,
        sx21, sy21, sz21, tx21, ty21, tz21, sx22, sy22, sz22, tx22, ty22, tz22,
        sx23, sy23, sz23, tx23, ty23, tz23, sx24, sy24, sz24, tx24, ty24, tz24,
        sx25, sy25, sz25, tx25, ty25, tz25, sx26, sy26, sz26, tx26, ty26, tz26,
```

```
        sx27, sy27, sz27, tx27, ty27, tz27, sx28, sy28, sz28, tx28, ty28, tz28,
        sx29, sy29, sz29, tx29, ty29, tz29, sx30, sy30, sz30, tx30, ty30, tz30,
        sx31, sy31, sz31, tx31, ty31, tz31, sx32, sy32, sz32, tx32, ty32, tz32,
        sx33, sy33, sz33, tx33, ty33, tz33, sx34, sy34, sz34, tx34, ty34, tz34,
        sx35, sy35, sz35, tx35, ty35, tz35};


(* Read in Mathematica script to obtain shape functions, R, P, and B *)
<<NQuinticEEorig.txt
b=B;

(* Define the G matrix *)
Print["Defining the G matrix . . ."];
(* Redefine G to avoid expression growth . . . *)
(* It appears that G is symmetric *)
(* We will probably define these values using the gmatrix.f file *)
G = {
{g11, g12, g13, g14, g15, g16, g17, g18, g19},
{g12, g22, g23, g24, g25, g26, g27, g28, g29},
{g13, g23, g33, g34, g35, g36, g37, g38, g39},
{g14, g24, g34, g44, g45, g46, g47, g48, g49},
{g15, g25, g35, g45, g55, g56, g57, g58, g59},
{g16, g26, g36, g46, g56, g66, g67, g68, g69},
{g17, g27, g37, g47, g57, g67, g77, g78, g79},
{g18, g28, g38, g48, g58, g68, g78, g88, g89},
{g19, g29, g39, g49, g59, g69, g79, g89, g99}};

gVec = {g11, g12, g13, g14, g15, g16, g17, g18, g19,
g22, g23, g24, g25, g26, g27, g28, g29,
g33, g34, g35, g36, g37, g38, g39,
g44, g45, g46, g47, g48, g49,
g55, g56, g57, g58, g59,
g66, g67, g68, g69,
g77, g78, g79,
g88, g89,
g99};

(* Define dummy variables and symbolic integration rule *)
mult = L1^t L2^t L3^t;
rule = {L1^aa_.L2^ab_.L3^ac_.->aa!ab!ac!/(aa+ab+ac+3)!};

(* Evaluate the error estimator using three terms
TERM 1: (N.sav)T.COM.(N.sav)
TERM 2: 2 (N.sav)T.B.u
```

```
TERM 3: (R.u)T.G.(R.u) *)

strm = OpenWrite["term1_QuinticOrig.f90"];

(* TERM 1: (N.sav)T.COM.(N.sav) *)
Print["Working on term 1 . . . "];
dum = NN.sav;
Print["Simplifying dum"];
dum = Simplify[dum];
Print["term1 = dum.com.dum"];
term1 = dum.com.dum;

(* First, expand term1 into individual terms, then collect together those terms that
involve the same powers of objects matching L1, L2, and L3 *)
Print["Expand . . ."];
term1 = term1 mult;
term1 = Expand[term1];

(* Integrate over volume by applying rule, substituting t = 0 *)
Print["About to integrate term 1 . . . [:|"];
term1 = term1/.rule;
term1 = term1/.t->0;

(* Puts the terms in a sum over a common denominator *)
Print["Put the terms in a sum over a common denominator"];
term1 = Together[term1, Extension->Automatic];

(* Extract the aforementioned denominator and store in dt1 *)
Print["Prepare to extract the denominator"];
dt1 = Denominator[term1];

(* Set term1 = dt1 x term1 . . . eliminates denominator in term1, now we just have a
sum of terms without denominators *)
Print["Set term1 = dt1 x term . . ."];
term1 = dt1 term1;

(* Expand term1, then collect together those terms that involve the same powers of
onjects matching cm11, cm12, and cm13 *)
Print["Expand term 1 . . ."];
term1 = Collect[Expand[term1], sav];

(* Coefficent gives the coefficient of cm11 in term1 *)
Print["Break term1 into parts"];
str2 = "term1 =";
```

```
Do[
        tnt = Coefficient[term1, cmm[[i]]];
        term1 = Expand[term1 - tnt cmm[[i]]];

        tnt = Collect[tnt, sav];
        str1 = "t1cm1" <> ToString[i] <> "=";

        str1 = "t1cm1" <> ToString[i] <> "=";
        str2 = str2 <> " t1cm1" <> ToString[i] <> "*cm1" <> ToString[i] <> "+";
        WriteString[strm, str1, ToString[FortranForm[tnt]], "\n"],
{i,1,3}];
str2 = str2<> "term1\n";
WriteString[strm,str2];
str1 = "dt1 =";
WriteString[strm, str1, ToString[FortranForm[dt1]], "\n"];
WriteString[strm, "term1=term1/dt1","\n"];
Print["Finished writing term 1 to the term1_2.f90 file . . ."];
Print["End of Term1"];
Close[strm];

 (* Term 2 *)

(* Evaluate the error estimator using three terms
TERM 1: (N.sav)T.COM.(N.sav)
TERM 2: 2 (N.sav)T.B.u
TERM 3: (R.u)T.G.(R.u) *)

strm = OpenWrite["term2_QuinticOrig.f90"];

(* TERM 2: 2 (N.sav)T.B.u *)
Print["Working on term 2 . . . "];

Print["Creating dum1"];
dum1 = NN.sav;
Print["Creating dum2"];
dum2 = b.u;
Print["Creating term2"];
term2 = dum1.dum2;

(* Expand term2, then collect together those terms that involve the same powers of
objects matching L1, L2, L3 *)
Print["Expand term 2"];
term2 = Expand[term2 mult];
```

```
(* Puts the terms in a sum over a common denominator *)
term2 = term2/.rule/.t->0;

Print["Put terms in a sum over a common denominator"];
term2 = Together[term2];

(* Extract the aforementioned denominator and store in dt2 *)
Print["Extract the denonminator"];
dt2 = Denominator[term2];

(* eliminates denominator in term2, now we just have a sum of terms without
denominators *)
Print["Eliminate the denominator in term2"];
term2 = dt2 term2;
Print["Expand term2"];
term2 = Expand[term2];
Print["Collect term2"];
term2 = Collect[term2,{c11, c12, c13, c21, c22, c23, c31, c32, c33}];
Print["Break term2 into parts"];
str2 = "term2 = 0.0"

Do[
        Do[
        Print["Working on: ", ToString[i], " ", ToString[j]];
        tnt = Coefficient[term2, cc[[i,j]]];
        term2 = Expand[term2 - tnt cc[[i,j]]];
        tnt = Collect[tnt, u ];
        str1 = "t2c" <> ToString[i] <> ToString[j] <> " = ";
        str2 = str2 <> " + t2c" <> ToString[i] <> ToString[j] <> "*c" <> ToString[i] <>
ToString[j];
        WriteString[strm, str1, ToString[FortranForm[tnt]], "\n"],
        {j,1,3}],
{i,1,3}];

WriteString[strm, str2];
str1 = "\n"<>"dt2=";
WriteString[strm, str1, ToString[FortranForm[dt2]], "\n"];
WriteString[strm, "term2=term2/dt2","\n"];

Print["Finished writing term 2 . . ."];

Close[strm];

 (* Term 3 *)
```

```
(* Define dummy variables and symbolic integration rule *)
mult = L1^t L2^t L3^t;
rule = {L1^aa_.L2^ab_.L3^ac_.->aa!ab!ac!/(aa+ab+ac+3)!};

(* Evaluate the error estimator using three terms
TERM 1: (N.sav)T.COM.(N.sav)
TERM 2: 2 (N.sav)T.B.u
TERM 3: (R.u)T.G.(R.u) *)

strm = OpenWrite["term3_QuinticOrig.f90"];

(* TERM 3: (R.u)T.G.(R.u) *)
(* Using version from Shiakolas dissertation *)

Print["Creating term3"];
dum3 =  R.u;
term3 = dum3.G.dum3;

(* Expand term3, then collect on L1, L2, and L3 *)
term3 = term3 mult;
term3 = Expand[term3];

(* Integrate term3 by substitution *)
term3 = term3/.rule;
term3 = term3/.t->0;
term3 =Together[term3];
Print["Extracting denominator"];
dt3 = Denominator[term3];
Print["Eliminating denominator"];
term3 = dt3 term3;
Print["Collecting and Expanding . . . ; )"];
term3 = Collect[Expand[term3], gVec];

Print["Break term3 into parts"];
str2 = "term3 = 0.0"
Do[
        Do[
        Print["Working on: ", ToString[i], " ", ToString[j]];
        Print["Determining tnt"];
        tnt = Coefficient[term3, G[[i,j]]];

        term3 = Expand[term3 - tnt G[[i,j]]];
```

```
        tnt = Collect[tnt, u];

        str1 = "t3g" <> ToString[i] <> ToString[j] <> "=";

        str2 = str2 <> " + t3g" <> ToString[i] <> ToString[j] <> "*g" <> ToString[i] <>
ToString[j];
        WriteString[strm, str1, ToString[FortranForm[tnt]]<> "\n"],
        {j,i,9}],
{i,1,9}];

WriteString[strm, str2];
WriteString[strm,"\n"];
str1 = "dt3=";

WriteString[strm, str1 <> ToString[dt3] <> "\n"];
Close[strm]
```

APPENDIX C

P-LEVEL 3 NUMERICAL ERROR ESTIMATION

```
(* Based on Shiakolas *)
(* Modified and adapted by SE McCaslin *)
(* Order of the shape functions *)
p = 3;
nTot = 20;

Print["p = ",p];
i3 = IdentityMatrix[3];
i6 = IdentityMatrix[6];
A = {
{1, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 1, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 1, 0, 1, 0}, {0, 0, 1, 0, 0, 0, 1, 0, 0}
};

(* Elasticity matrix *)
DD = { {ev1, ev2, ev2, 0, 0, 0}, {ev2, ev1, ev2, 0, 0, 0 }, {ev2, ev2, ev1, 0, 0, 0 }, {0, 0,
0, ev3, 0, 0}, {0, 0, 0, 0, ev3, 0}, {0, 0, 0, 0, 0, ev3} };

(* Displacement vector *)
dd = {u1, v1, w1, u2, v2, w2, u3, v3, w3, u4, v4, w4, u5, v5, w5,
u6, v6, w6, u7, v7, w7, u8, v8, w8, u9, v9, w9, u10, v10, w10,
u11, v11, w11, u12, v12, w12, u13, v13, w13, u14, v14, w14, u15, v15, w15,
u16, v16, w16, u17, v17, w17, u18, v18, w18, u19, v19, w19, u20, v20, w20};

(* Nodal stresses *)
sav = { sx1, sy1, sz1, tx1, ty1, tz1, sx2, sy2, sz2, tx2, ty2, tz2,
        sx3, sy3, sz3, tx3, ty3, tz3, sx4, sy4, sz4, tx4, ty4, tz4,
        sx5, sy5, sz5, tx5, ty5, tz5, sx6, sy6, sz6, tx6, ty6, tz6,
        sx7, sy7, sz7, tx7, ty7, tz7, sx8, sy8, sz8, tx8, ty8, tz8,
        sx9, sy9, sz9, tx9, ty9, tz9, sx10, sy10, sz10, tx10, ty10, tz10,
        sx11, sy11, sz11, tx11, ty11, tz11, sx12, sy12, sz12, tx12, ty12, tz12,
        sx13, sy13, sz13, tx13, ty13, tz13, sx14, sy14, sz14, tx14, ty14, tz14,
        sx15, sy15, sz15, tx15, ty15, tz15, sx16, sy16, sz16, tx16, ty16, tz16,
        sx17, sy17, sz17, tx17, ty17, tz17, sx18, sy18, sz18, tx18, ty18, tz18,
        sx19, sy19, sz19, tx19, ty19, tz19, sx20, sy20, sz20, tx20, ty20, tz20};


(* ELA will be the D matrix used in Nambiar's Calculations *)
ela={
{el11, el12, el13, el14, el15, el16},
{el12, el22, el23, el24, el25, el26},
{el13, el23, el33, el34, el35, el36},
{el14, el24, el34, el44, el45, el46},
{el15, el25, el35, el45, el55, el56},
```

```
{el16, el26, el36, el46, el56, el66} };

L4 = 1 - L1 - L2 - L3;

(* Initialize shape function matrix N for 20 nodes *)
nn = Table[0, {nTot}];

(* Calculated nodes per edge *)
nEd = 6*(p-1);
Print["Edge nodes: ",nEd];
(* Face nodes *)
nFa = 2(p-1)(p-2);
Print["Face nodes: ",nFa];
(* Bubble (centroid) nodes *)
nBu = (p-1)(p-2)(p-3)/6;
Print["Bubble nodes: ",nBu];
nTot=4+nEd+nFa+nBu;
Print["Total nodes: ",nTot];

(* Hierarchic Shape Functions for the Tetrahedral Element *)
(* AUTHOR:            S.E. McCaslin *)
 (* REFERENCE:      Szabo and Babuska, Finite Element Analysis, pp. 242 - 244 *)

(* phi and Legendre polynomials tested and found to match Szabo and Babuska, p. 103
*)

(* 0, 1, 2, . . . *)
LPoly = {1, x, 1/2 (3 x ^ 2 - 1), 1/2 (5 x ^ 3 - 3 x ), 1/8 (35 x ^ 4 - 30 x ^ 2 + 3), 1/8 (63
x ^ 5 - 70 x ^ 3 + 15 x) };

Legendre[i_] := LPoly[[i]];

Ei[i_, t1_, t2_] := - 8 Sqrt[4 i + 2] / (i (i + 1)) (D[Legendre[i+1],x])/.x->(t2 - t1)
Fi[r1_, r2_, t1_, t2_, t3_] := (Legendre[r1+1]/.x->(t2 - t1))*(Legendre[r2+1]/.x->(2 t3 -
1))

Print["Creating vertices"];
(* Coordinate system:  eta, zeta, xi *)
(* Four nodal shape functions, one for each node *)
nn[[1]] = L1;  nn[[2]] = L2;  nn[[3]] = L3;  nn[[4]] = L4;


(* Edge modes *)
Print["Creating edge modes"];
```

```
nn[[5]] = Simplify[L2 L3 Ei[1,L2,L3]];
nn[[6]] = Simplify[L1 L3 Ei[1,L1,L3]];
nn[[7]] = Simplify[L1 L2 Ei[1,L1,L2]];
nn[[8]] = Simplify[L1 L4 Ei[1,L1,L4]];
nn[[9]] = Simplify[L2 L4 Ei[1,L2,L4]];
nn[[10]] = Simplify[L3 L4 Ei[1,L3,L4]];

(* nodes between 2 and 3 *)
nn[[11]] = Simplify[L2 L3 Ei[2,L2,L3]];
nn[[12]] = Simplify[L1 L3 Ei[2,L1,L3]];
nn[[13]] = Simplify[L1 L2 Ei[2,L1,L2]];
nn[[14]] = Simplify[L1 L4 Ei[2,L1,L4]];
nn[[15]] = Simplify[L2 L4 Ei[2,L2,L4]];
nn[[16]] = Simplify[L3 L4 Ei[2,L3,L4]];

(* Face modes *)
nn[[17]] = Simplify[ L2 L3 L4 Fi[0, 0, L2, L3, L4 ]];
nn[[18]] = Simplify[ L3 L4 L1 Fi[0, 0, L1, L3, L4 ]];
nn[[19]] = Simplify[ L4 L1 L2 Fi[0, 0, L1, L2, L4 ]];
nn[[20]] = Simplify[ L1 L2 L3 Fi[0, 0, L1, L2, L3 ]];
(* From Shiakolas *)

 (* Put into appropriate format for use with developed equations *)
(* We use i6 because we have six stresses possible per node *)
NT = Flatten[Table[i6 * nn[[i]], {i,1,nTot}],1];
NN = Transpose[NT];

(* Form the R matrix *)
NTi = Flatten[Table[i3 * nn[[i]], {i,1,nTot}],1];
NNi = Transpose[NTi];

RL1 = D[NNi,L1];RL2 = D[NNi,L2];RL3 = D[NNi,L3];
R = Flatten[{RL1, RL2, RL3}, 1];
Print["Dimensions of R: ", Dimensions[R]];

(* Generate the P matrix *)
Print["Generating GAM"];
g1 = Transpose[Flatten[{i3 c11, i3 c12, i3 c13}, 1]];
g2 = Transpose[Flatten[{i3 c21, i3 c22, i3 c23}, 1]];
g3 = Transpose[Flatten[{i3 c31, i3 c32, i3 c33}, 1]];
GAM = Flatten[{g1, g2, g3}, 1];
Print["Generating P matrix as A.GAM"];
P = A.GAM;
Print["Dimensions of P: ", Dimensions[P]];
```

```
(* Note that Transpose[P].ELA.P does provide G as a 9x9 matrix *)

(* We know that B = P R *)
B = P.R;
Print["Dimensions of B: ",Dimensions[B]];

(* Write b matrix to file *)
strm=OpenWrite["b3SORT.f90"];
For[
        ii=1, ii<=6,
        For[
                jj=1, jj<=3*nTot,
                WriteString[strm, "b(" <> ToString[ii] <> "," <> ToString[jj] <> ")=" <>
ToString[FortranForm[B[[ii,jj]]]] <> "\n"];
                jj++
      ];
        ii++
];
Close[strm];

(* Calculate D.B *)
temp = DD.B;
str = temp.dd;
cc = {c11, c12, c13, c21, c22, c23, c31, c32, c33 }
str = Collect[Expand[str],cc];
strm=OpenWrite["str3SORT.f90"];
jj = 1;
For[
        jj = 1, jj <= 6,
        Print[jj];
        strng = "str(" <> ToString[jj] <> ")=";
        Print[string];
        For[
                ii = 1, ii<=9,
                strng = strng <> "b" <> ToString[cc[[ii]]] <> "*" <> ToString[cc[[ii]]]
<> "+";
                tnt = Coefficient[str[[jj]], cc[[ii]]];
                tnt = Simplify[tnt];
                str[[jj]] = Expand[str[[jj]] - tnt cc[[ii]]];
                WriteString[strm,"b" <> ToString[cc[[ii]]] <> "=" <>
ToString[FortranForm[tnt]]"\n"];
                ii++
        ];
        If[jj == 1||jj==2||jj==3,
```

```
        WriteString[strm,strng <> ToString[FortranForm[str[[jj]]]] <> "-thermfac\n"],
        WriteString[strm,strng <> ToString[FortranForm[str[[jj]]]] <> "\n"]];
        jj++
];
Close[strm];

(* Calculate sigma hat *)
sigmaHat = DD.P.R.dd;
sigmaHat = Collect[Expand[sigmaHat],cc];
strm=OpenWrite["sHat3SORT.f90"];
jj = 1;
For[
        jj = 1, jj <= 6,
        Print[jj];
        strng = "shat(" <> ToString[jj] <> ")=";
        For[
                ii = 1, ii<=9,
                strng = strng <> "s" <> ToString[jj] <> ToString[cc[[ii]]] <> "*" <>
ToString[cc[[ii]]] <> "+";
                tnt = Coefficient[sigmaHat[[jj]], cc[[ii]]];
                tnt = Collect[tnt, {ev1, ev2, ev3}];
                sigmaHat[[jj]] = Expand[sigmaHat[[jj]] - tnt cc[[ii]]];
                WriteString[strm,"s" <> ToString[jj] <> ToString[cc[[ii]]] <> "=" <>
ToString[FortranForm[tnt]]"\n"];
                ii++
        ];
        If[jj == 1||jj==2||jj==3,
        WriteString[strm,strng <> ToString[FortranForm[str[[jj]]]] <> "-thermfac\n"],
        WriteString[strm,strng <> ToString[FortranForm[str[[jj]]]] <> "\n"]];
        jj++
];
Close[strm];

(* Re-derive NN such that L4 remains in the shape functions *)
Clear[L4];

(* Coordinate system:  eta, zeta, xi *)
(* Four nodal shape functions, one for each node *)
nn[[1]] = L1;  nn[[2]] = L2;  nn[[3]] = L3;  nn[[4]] = L4;

(* Edge modes *)
nn[[5]] = Simplify[L2 L3 Ei[1,L2,L3]];
nn[[6]] = Simplify[L1 L3 Ei[1,L3,L1]];
nn[[7]] = Simplify[L1 L2 Ei[1,L1,L2]];
```

```
nn[[8]] = Simplify[L1 L4 Ei[1,L1,L4]];
nn[[9]] = Simplify[L2 L4 Ei[1,L2,L4]];
nn[[10]] = Simplify[L3 L4 Ei[1,L3,L4]];
nn[[11]] = Simplify[L2 L3 Ei[2,L2,L3]];
nn[[12]] = Simplify[L1 L3 Ei[2,L3,L1]];
nn[[13]] = Simplify[L1 L2 Ei[2,L1,L2]];
nn[[14]] = Simplify[L1 L4 Ei[2,L1,L4]];
nn[[15]] = Simplify[L2 L4 Ei[2,L2,L4]];
nn[[16]] = Simplify[L3 L4 Ei[2,L3,L4]];

nn[[17]] = Simplify[ L2 L3 L4 Fi[0, 0, L2, L3, L4 ]];
nn[[18]] = Simplify[ L3 L4 L1 Fi[0, 0, L3, L4, L1 ]];
nn[[19]] = Simplify[ L4 L1 L2 Fi[0, 0, L4, L1, L2 ]];
nn[[20]] = Simplify[ L1 L2 L3 Fi[0, 0, L1, L2, L3 ]];

(* From Shiakolas *)

(* Put into appropriate format for use with developed equations *)
(* We use i6 because we have six stresses possible per node *)
NT = Flatten[Table[i6 * nn[[i]], {i,1,nTot}],1];
NN = Transpose[NT];

(* Calculate sigma* *)
sstar = NN.sav;
strm=OpenWrite["sstar3SORT.f90"];
jj = 1;
For[
        jj = 1, jj <= 6,
        WriteString[strm,"sstar(" <> ToString[jj] <> ")=" <>
ToString[FortranForm[Simplify[sstar[[jj]]]]]<>"\n"];
        jj++
];
Close[strm];
```

APPENDIX D

P-LEVEL 1 EQUIVALENT NODAL TEMPERATURE LOAD

```
(* Variables defined for symbolic integration *)
mult = L1^t L2^t L3^t;
rule = {L1^aa_.L2^ab_.L3^ac_.->aa!ab!ac!/(aa+ab+ac+3)!};

i3 = IdentityMatrix[3];

A = {
{1, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 1, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 1, 0, 1, 0}, {0, 0, 1, 0, 0, 0, 1, 0, 0}
};

L4 = 1 - L1 - L2 - L3;
nTot = 4;

(* Initialize shape function matrix nn *)
nn = Table[0, {nTot}];

(* Coordinate system:  eta, zeta, xi *)
(* Four nodal shape functions, one for each node *)
nn[[1]] = L1;  nn[[2]] = L2;  nn[[3]] = L3;  nn[[4]] = L4;

(* From Shiakolas *)

 (* Put into appropriate format for use with developed equations *)
Print["About to flatten"];
NT = Flatten[Table[i3 * nn[[i]], {i,1,nTot}],1];
Print["About to transpose"];
NN = Transpose[NT];

(* Form the R matrix *)
RL1 = D[NN,L1];RL2 = D[NN,L2];RL3 = D[NN,L3];
R = Flatten[{RL1, RL2, RL3}, 1];
Print["Dimensions of R: ", Dimensions[R]];

(* Generate the P matrix *)
g1 = Transpose[Flatten[{i3 c11, i3 c12, i3 c13}, 1]];
g2 = Transpose[Flatten[{i3 c21, i3 c22, i3 c23}, 1]];
g3 = Transpose[Flatten[{i3 c31, i3 c32, i3 c33}, 1]];
GAM = Flatten[{g1, g2, g3}, 1];
P = A.GAM;
Clear[g1, g2, g3, A, GAM];

(* We know that B = P R *)
B = P.R;
```

```
Print["Dimensions of B: ",Dimensions[B]];

(* Define initial strain vector, assume uniform temperature over the element,
 eo = {a dt, a dt, adt, 0, 0, 0}, where a dt is factored out for ease of calcs *)
eo = {1, 1, 1, 0, 0, 0};

ela = { {e1, e2, e2, 0, 0, 0}, {e2, e1, e2, 0, 0, 0}, {e2, e2, e1, 0, 0, 0}, {0, 0, 0, e3, 0, 0},
{0, 0, 0, 0, e3, 0}, {0, 0, 0, 0, 0, e3}};

bt = Transpose[B];
f0 = bt.ela;
f0 = f0.eo;
f0 = f0 mult;
f0 = Expand[f0];
f0 = f0/.rule;
f0 = f0/.t->0;
f0 = Expand[f0];
f0 = f0 adt;
f0 = Expand[f0];

strm = OpenWrite["temp1.f90"];
Do[ WriteString[strm, "tl(" <> ToString[i] <> ")=" <> ToString[FortranForm[f0[[i]]]]
<> "\n"], {i,3*nTot}];
Close[strm];
```

APPENDIX E

P-LEVEL 1 EQUIVALENT NODAL PRESSURE/SHEAR LOAD

```
(* Based on the work of Dr. Shiakolas *)

(* Assume that on the loaded face L4 is 0 *)
L4 = 0;
i3 = IdentityMatrix[3];

nTot = 4;

(* Variables defined for symbolic integration *)
mult = L1^t L2^t L3^t;
rule = {L1^aa_.L2^ab_.L3^ac_.->aa!ab!ac!/(aa+ab+ac+2)!};

(* Initialize shape function matrix nn *)
nn = Table[0, {nTot}];

(* Four nodal shape functions, one for each node *)
nn[[1]] = L1;  nn[[2]] = L2;  nn[[3]] = L3;  nn[[4]] = L4;

(* Put into appropriate format for use with developed equations *)
NT = Flatten[Table[i3 * nn[[i]], {i,1,nTot}],1];
NN = Transpose[NT];

(* Define the pressure or shear direction cosines *)
phi = {fx, fy, fz};

(* Evaluate the equivalent nodal load vector *)
Print["Evaluating equivalent nodal load vector"];
f = phi.NN;
f = f mult;
f = Expand[f];
f = f/.rule;
f = f/.t->0;
f = Collect[f, phi];

Print["Writing the Equivalent nodal load file . . . : )"];
strm=OpenWrite["press1.f90"];
Do[ WriteString[strm,"fp(" <> ToString[i] <> ")=" <> ToString[FortranForm[f[[i]]]]<>
"\n"], {i,3*nTot}];
Close[strm];
```

APPENDIX F

DETAILED COMPACTION RESULTS FOR HIGHER ORDER ELEMENTS

Comparison of Hierarchical *p*-level 3 and 4 File Sizes.

File Type	*Original Size, F77 Kb*	*Compacted, F77 Kb*	*% of Original Size*
Hierarchical P-level 3			
[B]	20.9	10.3	49%
[K]	91.3	82.8	91%
$\hat{\sigma}$	34.9	6.97	20%
σ^*	2.58	2.11	82%
Centroidal Stress	29.9	5.87	20%
Term1	16.1	16.5	102%
Term2	591.	261.	44%
Term3	88.9	41.8	47%
Hierarchical P-level 4			
B matrix	53.6	24.1	45%
[K]	327.	288.	88%
Centroidal Stress	93.0	15.9	17%
Term3	368.	160.	43%

Comparison of Isoparametric *p*-level 4 File Sizes.

File Type	*Original Size, F77 Kb*	*Compacted, F77 Kb*	*% of Original Size*
Isoparametric P-level 4			
B matrix	79.6	32.9	41%
[K]	379.	341.	90%
$\hat{\sigma}$	143.	31.9	22%
σ^*	7.28	7.28	100%
Centroidal Stress	122.	79.6	65%
Term1	79.8	80.7	101%
Term2	257.	149.	58%
Term3	271.	163.	60%
Curved Isoparametric P-level 4			
Jacobian	25.2	15.7	62%
[B]	98.6	58.5	59%
$\hat{\sigma}$	91.6	57.8	63%
σ^*	90.8	58.0	64%

REFERENCES

1. Zienkiewicz, O.C. The birth of the finite element method and of computational mechanics, *International Journal for Numerical Methods in Engineering* 2004; **60**:3 – 10.
2. Samuelsson, A., and Zienkiewicz, O.C., History of the stiffness method, *International Journal for Numerical Methods in Engineering* 2006; **67**:149 – 157.
3. Zienkiewicz, O.C. Achievements and some unsolved problems of the finite element method, *International Journal for Numerical Methods in Engineering* 2000; **47**:9 – 28.
4. Shiakolas, P.S. Closed form expressions for higher order tetrahedral finite elements, Doctoral Dissertation, The University of Texas at Arlington 1992.
5. Shiakolas, P.S., Lawrence, K.L., and Nambiar, R.V. Closed-form expressions for the linear and quadratic strain tetrahedral finite elements. *Computers and Structures* 1994; **50**:743 – 747.
6. Shiakolas, P.S., Lawrence, K.L., and Nambiar, R.V. Closed-form error estimators for the linear and quadratic strain tetrahedron finite elements. *Computers and Structures* 1993; **47**:907 - 915.

7. Babuska, I., Katz, I.N., and Szabo, B.A. Hierarchic families for the p-version of the finite element method. Advances in Computer Methods for Partial Differential Equations IMACS 1979; 272 – 286.
8. Szabo, B.A., and Babuska, I. *Finite Element Analysis*. New York: John Wiley & Sons, 1991.
9. Argyris, J.H., Fried, I., and Scharpf, D.W. The TET 20 and TEA 8 elements for the matrix displacement method, *The Aeronautical Journal of the Royal Aeronautical Society* 1968; **72**:618 – 623.
10. Dennis, B.H., Eberhart, R.C., Dulikravich, G.S., and Radons, S.W., Finite element simulation of cooling of 3-D human head and neck, *ASME Journal of Biomechanical Engineering* 2003; **125**:832 – 840.
11. Gallagher, R.H., Padlog, J, and Bijlaard, P.P. Stress analysis of heated complex shapes. *A. R. S. Journal* 1962; 700 – 707.
12. Melosh, R.J. Structural analysis of solids, *Proc. Amer. Soc. Civ. Eng.* 1963; **S.T.4**:205 – 223.
13. Pawlak, T.P., Yunus, S.M., Cook, R.D. Solid elements with rotational degrees of freedom: Part II – tetrahedron elements, *International Journal for Numerical Methods in Engineering* 1991: **31**:593 – 610.
14. Key, S.W., Heinstien, M.W., Stone, C.M., Mello, F.J., Blanford, M.L., and Budge, K.G. A suitable low-order, tetrahedral finite element for solids, *International Journal for Numerical Methods in Engineering* 1999; **44**:1785 – 1805.

15. Kong, M.J.S., Mulder, W.A., and Van Veldhuizen, Higher-order triangular and tetrahedral finite elements with mass lumping for solving the wave equation, *Journal of Engineering Mathematics* 1999; **35**:405 – 426.
16. Bittencourt, M.L. Fully tensorial nodal and modal shape functions for triangles and tetrahedra, *International Journal for Numerical Methods in Engineering* 2005; **63**:1530 – 1538.
17. Katz, I.N., and Rossow, M.P. Hierarchic finite elements and precomputed arrays. *International Journal for Numerical Methods in Engineering* 1978: **31**:977 – 999.
18. Babuska, I., and Szabo, B.A., On the rates of convergence of the finite element method, *International Journal for Numerical Methods in Engineering* 1982; **18**:323 – 341.
19. Babuska, I., and Suri, M. The p and h-p versions of the finite element method, an overview, *Computer Methods in Applied Mechanics and Engineering* 1990; **80:**5 – 26.
20. Carnevali, P., Morris, R.B., Tsuji, Y., and Taylor, G. New basis functions and computational procedures for p-version finite element analysis, *International Journal for Numerical Methods in Engineering* 1993: **36**:3759 – 3779.
21. Adjerid, S., Aiffa, M. and Flaherty, J.E. Hierarchical finite element bases for triangular and tetrahedral elements, *Computer Methods in Applied Mechanics and Engineering* 2001; **190:**2925 – 2941.

22. Zienkiewicz, O.C., and Taylor, R.L., *The Finite Element Method, Volume 1: Basic Formulation and Linear Problems*, 4th edition. New York: McGraw-Hill Book Company, 1994.

23. Tinawi, R.A. Anisotropic tapered elements using displacement models, *International Journal for Numerical Methods in Engineering* 1972; **4**:475 – 489.

24. Subramanian, G., and Bose, C.J. Convenient generation of stiffness matrices for the family of plane triangular elements, *Computers and Structures* 1982; **10**:119 – 124.

25. Subramanian, G., and Bose, C.J. On stiffness matrices for C_0 continuous tetrahedra, *Computers and Structures* 1983; **16**:603 – 611.

26. Rathod, H.T., and Karim, M.S. An explicit integration scheme based on recursion for the curved triangular finite elements, *Computers and Structures* 2002; **80**:43 – 76.

27. Babu, D., and Pinder, G.F. Analytical integration formulae for linear isoparametric finite elements, *International Journal for Numerical Methods in Engineering* 1984; **20**:1153 – 1166.

28. Nambiar, R.V. Closed form expressions for hierarchic triangular and tetrahedral finite elements, Doctoral Dissertation, The University of Texas at Arlington 1989.

29. Lawrence, K.L., Nambiar, R.V. and Bergmann, B. Closed form stiffness matrices and error estimators for plane hierarchic triangular elements, *International Journal for Numerical Methods in Engineering* 1991; **31**:879 – 894.

30. Dey, S., Flaherty, J.E., Ohsumi, T.K., and Shephard, M.S. Integration by table look-up for p-version finite elements on curved tetrahedra, *Computer Methods in Applied Mechanics and Engineering* 2006; **195**:4532 – 4543.
31. Zienkiewicz, O.C. The background of error estimation and adaptivity in finite element computations, *Computer Methods in Applied Mechanics and Engineering* 2006; **195**:207 – 213.
32. Babuska, I., and Rheinboldt, C. A-posteriori error estimates for the finite element method, *International Journal for Numerical Methods in Engineering* 1978; **12**:1597 – 1615.
33. Zienkiewicz, O.C. and Zhu, J.C. A simple error estimator and adaptive procedure for practical engineering analysis, *International Journal for Numerical Methods in Engineering* 1987; **24**:337 – 357.
34. Ainsworth, M. , Zhu, J.Z., Craig, A.W. and Zienkiewicz, O.C. Analysis of the Zienkiewicz-Zhu a-posteriori error estimator in the finite element method, *International Journal for Numerical Methods in Engineering* 1989; **28**:2161 – 2174.
35. Byrd, D.E. Identification and elimination of errors in finite element analysis, Doctoral Dissertation, University of Colorado 1988.
36. Pawlack, Tim. The ANSYS error estimation capability, *ANSYS News*, Swanson Analysis Systems Inc., Houston, PA, Fourth Issue, 1-2, 1989.

37. Zienkiewicz, O.C., and Zhu, J.Z. Superconvergent patch recovery (SPR) and adaptive finite element refinement, *Computer Methods in Applied Mechanics and Engineering* 1992; **101**:207 – 224.

38. Boroomand, B., and Zienkiewicz, O.C. Recovery by equilibrium in patches (REP), *International Journal for Numerical Methods in Engineering* 1997; **40**:137 – 164.

39. Cartensen, C., and Funken, S.A. Average technique for FE – a posteriori error control in elasticity. Part i: Conforming FEM, *Computer Methods in Applied Mechanics and Engineering* 2001; **190**: 2483 – 2498.

40. Shiakolas, P.S., Lawrence, K.L., and Nambiar, R.V. Closed-form error estimators for the linear strain and quadratic strain tetrahedron finite elements. *Computers and Structures* 1993; **47**:907 – 915.

41. Felippa, C.A. A compendium of FEM integration formulas for symbolic work, *Engineering Computations* 2004; **21**:867 – 890.

42. Cools, R. and Rabinowitz, P. Monomial cubature rules since "Stroud": a compilation, *Journal of Computational and Applied Mathematics*, 1993; **48**:309 – 326.

43. Cools, R. Monomial cubature rules since "Stroud": a compilation – part 2, *Journal of Computational and Applied Mathematics*, 1999; **112**:21 – 27.

44. Cools, R. An encyclopedia of cubature formulas, *Journal of Complexity*, 2003; **19**:445 – 453.

45. Muthukrishnan, S.N. Mesh generation and adaptive refinement of tetrahedral elements for three-dimensional finite element applications, Doctoral Dissertation, The University of Texas at Arlington 1993.
46. Xin, J., Pinchedez, K., and Flaherty, J.E. Implementation of hierarchical bases in FEMLAB for simplicial elements, *ACM Transactions on Mathematical Software*, 2005; **31**:187–200.
47. Gellert, M., and Harboud, R. Moderate degree cubature formulas for 3-D tetrahedral finite-element approximations, *Communications in Applied Numerical Method*, 1991; **7**:487 – 495.
48. Nambiar, R.V., and Lawrence, K.L. The Zienkiewicz-Zhu error estimator for multiple material problems, *Communcations in Applied Numerical Methods*, 1992; **8**:273 – 277.
49. Peano, A. Short Communications: Gauss-Lobatto integration of high precision tetrahedral elements, , *International Journal for Numerical Methods in Engineering* 1982; **18**:311 – 320.
50. Cook, R.D. *Concepts and Applications of Finite Element Analysis*. New York: John Wiley & Sons, 1974.
51. Sayood, K. *Introduction to Data Compression*. California: Morgan Kaufmann, 2000.
52. Visual Basic Developer Center, "String Data Type (Visual Basic)," MSDN, http://msdn2.microsoft.com/en-us/library/thwcx436(vs.80).aspx (accessed January 22, 2008).

53. Wolfram Mathematica Documentation Center, "Expand," Wolfram, http://reference.wolfram.com/ mathematica/ref/Expand.html (accessed January 16, 2008).
54. Wolfram Mathematica Documentation Center, "Rules," Wolfram, http://reference.wolfram.com/ mathematica/ref/Rule.html (accessed January 16, 2008).
55. Wolfram Mathematica Documentation Center, "Collect," Wolfram, http://reference.wolfram.com/ mathematica/ref/Collect.html (accessed January 16, 2008).
56. Wolfram Mathematica Documentation Center, "FortranForm," Wolfram, http://reference .wolfram.com/mathematica/ref/FortranForm.html (accessed January 16, 2008).
57. Hughes, T.J.R. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. New York: Dover Publications Inc., 1994 (Orig. published 1987).
58. Juvinal, R.C. *Stress, Strain and Strength*. Ohio: McGraw Hill Book Company, 1967.
59. Ugural, A.C., and Fenster, S.K. *Advanced Strength and Applied Elasticity*, 3rd ed., Prentice-Hall PTR: New Jersey, 1995.
60. Roark, R.J., Young, W.C. and Budynas, R.G. *Roark's Formulas for Stress and Strain*, 7th edition. Ohio: McGraw Hill Book Company, 2002.

61. Luo, X., Shephard, M.S., Remacle, J., O'bara, R.M., Beall, M.W., Szabo, B. and Actis, R. P-version mesh generation issues, *11th International Meshing Roundtable*, Sandia National Laboratories, 2002, pp. 343 – 354.
62. Sun Microsystems Documentation, "Sun Studio 12 Collect," Sun, http://docs.sun.com/app/docs/doc/819-5263/6n7c0cbgf?a=view (accessed April 7, 2008).
63. FTN95 Help files, "\TIMING", Silverfrost Ltd., 2006.
64. Scheutze, K.T., Shiakolas, P.S., Muthukrishnan, S.N., Nambiar, R.V., and Lawrence, K.L. A study of adaptively remeshed finite element problems using higher order tetrahedra, *Computers and Structures*, 1994; **54**:279—288.

BIOGRAPHICAL INFORMATION

Sara McCaslin has an AA in engineering from Tyler Junior College, graduating in 1998, summa cum laude. She also has a BSME from the University of Texas at Tyler, where she graduated summa cum laude in 2000. In 2002, she graduated with an MS in computer science, also from the University of Texas at Tyler, with a specialization in artificial intelligence. Sara started work on her PhD in mechanical engineering at the University of Texas at Arlington in 2003. She plans to continue working in research that combines her computer science skills with mechanical engineering applications, and hopes to pursue a career in education. She has worked in research involving hybrid DMSC-MD aerosol simulations, fuzzy neural networks, knowledge discovery with applications in satellite imagery, and, most recently, closed-form solutions in finite element analysis for her dissertation. Since obtaining her master's degree, she has taught at the University of Texas at Tyler for both the mechanical engineering and computer science departments.