1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

THÈSES CANADIENNES

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

Contributions to the Finite Element Solution of the Navier-Stokes Equations

by

(C) Marc Joseph Luc Dulude, B.Eng. (Mech.)

*A thesis submitted to the Faculty of Graduate Studies and Research
in Partial fulfillment of the requirements for the
Degree of Master of Engineering
in Mechanical Engineering*

DEPARTMENT OF MECHANICAL AND AERONAUTICAL ENGINEERING
CARLETON UNIVERSITY
OTTAWA, ONTARIO
CANADA

September 1986

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-35840-8

The undersigned recommend to the faculty of Graduate Studies acceptance of the thesis 'Contributions to the Finite Element Solution of the Navier-Stokes Equations' submitted by Marc J.L. Dulude, B.Eng. (Mech.) in partial fulfillment for the Degree of Master of Mechanical Engineering.

Thesis Supervisor

Chairman

Department of Mechanical
and Aeronautical Engineering

Abstract

Several problems have been overcome to further the solution of the Navier-Stokes equations by means of the finite element technique. The development of a twenty node brick was completed and a fifteen node brick was conceived and implemented. The frontal solver previously adapted from another source was improved upon to dramatically reduce solution time. The FORTRAN routines required for each of the achievements above were imbedded in the ASGARD finite element package. A post-processor for ASGARD output files was developed.

The theory and applications of hierarchical three dimensional elements are discussed. Both the mixed formulation and the penalty method are discussed since the two elements use either one method or the other. The test cases solved are discussed in order to relate theory, finite difference results, or experimental data to the solutions generated. The source code for all routines is supplied as is an operations guide for their use.

Acknowledgements

The modelling of fluid flow phenomena is becoming increasingly practical as available computers and software packages become more powerful. Computational tools are beginning to allow general solutions of equations which have been in existence for decades but were too complex to be solved for the general case. The transient, three dimensional Navier-Stokes equations are the governing equations for all Newtonian flows. The primary difficulty is solving this set of equations.

This report describes a series of small contributions to the modelling of the stationary Navier-Stokes equations and is the result of the help and support of many people. My eternal gratitude goes to both of my thesis supervisors, Dr S.A. Sjolander and Dr J.A. Goldak, for their patient and flawless advice and guidance. Ms Moyra McDill and Mr Alan Oddy aided me throughout the research with answers to my questions regarding the Finite Element Method (FEM) as well as the operation of the FEM program, ASGAR. Dr Andre Fortin of Ecole Polytechnique de Montreal provided valuable information during the development of the 15 node brick and made himself available for questions at my request. Dr Neal Holtz displayed his eternal patience while explaining various aspects of the APOLLO network and the mathematical word processor, TEX, to me. My sister, France, repeatedly prepared some of the charts in this report as a result of my changing opinions. Finally, I would like to express my deepest thanks to both of my parents without whose continuous support and coaxing, this project would never have succeeded.

Table of Contents

Abstract	
Acknowledgements	
Table of Contents	
List of Figures and Tables	
Nomenclature	
1 INTRODUCTION	1
1.1 Computational Fluid Dynamics	1
1.2 Previous Work	3
1.3 Aim & Scope of the Present Work	5
2 THE FINITE ELEMENT METHOD	9
2.1 Introduction	9
2.2 Hierarchical Elements	10
2.3 Rigid Body Mode Analogue	12
2.4 Description of the 20 Node Brick	13
2.5 Description of the 15 Node Brick	14
3 DERIVATION OF WORKING EQUATIONS	15
3.1 Introduction	15
3.2 20 Node Brick Mixed Formulation	16
3.2.1 Solution Algorithm	19
3.3 Constraining Streamwise Acceleration	23
3.4 Modified Usawa Algorithm For The 15 Node Brick	24
3.4.1 Solution Algorithm	26
4 DESCRIPTION OF FORTRAN CODE	36
4.1 20 Node Brick	36
4.1.1 VFL3D20	36
4.2 NOXAXL	37
4.3 15 Node Brick	38
4.3.1 VFL3D14	38
4.4 Frontal Solver	39
4.4.1 FRONTA	40
4.4.2 TOLREL	42
4.5 Post-Processor	44
4.5.1 SEPARATE	44
5 TEST CASES AND RESULTS	46
5.1 20 Node Brick	46

5.1.1 Plane Poiseuille Flow	47
5.1.2 Couette Flow	48
5.1.3 Two Dimensional Diffusing Section	48
5.1.4 Fully Developed Laminar Pipe Flow	49
5.1.5 Turbulent Pipe Flow	50
5.1.6 Three Dimensional Secondary Flow From Curved Duct	51
5.2 15 Node Brick	53
5.2.1 One Element	53
5.2.2 Two Elements In The Streamwise Direction	53
6 CONCLUSIONS & RECOMMENDATIONS	54
6.1 Conclusions	54
6.2 Future Work	55
References	57
Figures and Tables	60

APPENDICES

A DESCRIPTION OF ASGARD STRUCTURE AND USE	81
B VFL3D14	97
C FRONTA	128
D VFL3D20	153
E SEPARATE	181

List of Figures and Tables

Figures

1	Twenty Node Brick Topology	60
2	Fifteen Node Brick Topology	61
3	Reduction in Three Element Solution Time	62
4	One Element Poiseuille Flow Geometry	63
5	NOXAXL Degrees of Freedom	64
6	Diverging Section Geometry	65
7	Three Element Pipe Flow Geometry	66
8	Curved Duct Geometry	67
9	One Element Test - Fifteen Node Brick	68
10	Two Element Test - Fifteen Node Brick	69
11	Poiseuille Flow Velocity Profile	70
12	Diverging Section Velocity Profile	71
13	Laminar Pipe Flow Velocity Profile	72
14	NOXAXL Laminar Pipe Flow Velocity Profile	73
15	Turbulent Pipe Flow Velocity Profile	74
16	Velocities at the Exit Plane of the Curved Duct	75
17	ASGARD Flow Chart	76
18	Curved Duct Pressure Contours	77

Tables

1	Twenty Node Brick Shape Functions	78
2	Twenty Node Brick Shape Function Derivatives	79
3	Fifteen Node Brick Shape Functions	80
4	Fifteen Node Brick Shape Function Derivatives	81

Nomenclature

English

a_k	= constants describing a linear pressure field
$[B]$	= right hand side vector
dof	= degree(s) of freedom
$\bar{e}_i$	= unit vector in the i^{th} direction
F_i	= body force term in i^{th} direction
GP	= Gauss point position in local coordinates as determined by the 14-point non-product rule
h	= half the plate separation height
i, j, k	= equation indices for x_1, x_2, x_3 directions respectively or summation counters
$[J]$	= Jacobian matrix
$[K_{ij}]$	= stiffness matrix entry at row i , column j
$[K]$	= stiffness matrix
$L^2(\Omega)$	= space of square integrable functions
M_i, N_i, S_i	= i^{th} shape, basis, or interpolation functions
n, m	= summation indices for twenty and eight nodes respectively
$(n_i)_j$	= direction cosine
p	= static pressure
q	= Galerkin weighting function used for 15 node brick continuity equation
r	= penalty factor
$R(\bar{u}, p)$	= residual obtained from trial solution with 15 node brick
Re	= Reynolds number
Re_h	= Reynolds number based on plate separation
u_1, u_2, u_3	= velocity components in the x_1, x_2, x_3 directions
u'_i	= velocity perturbation
u^*	= velocity obtained from previous trial solution

$\vec{u}$	= velocity as a vector quantity
$\vec{u}^n$	= velocity vector at iteration number n
$\vec{\delta u}$	= velocity correction vector solved for in the 15 node brick formulation
U	= some known velocity
v_i	= Galerkin weighting function used for 15 node brick Navier-Stokes equations
V_i	= i^{th} elemental discrete divergence interpolation function
$[x]$	= vector of unknowns
$\vec{x}$	= spatial coordinates as a vector quantity
x_1, x_2, x_3	= Cartesian axes for physical domain
X_i, Y_i, Z_i	= conditions for elimination of barycentric dof

Greek

α_i	= dof number i in a dof vector
δ_{ij}	= Kronecker delta ($\delta = 1$ if $i = j$, $\delta = 0$ if $i \neq j$)
Δ	= hierarchical change in a dof value
θ	= over-relaxation parameter
μ	= dynamic viscosity
μ_e	= dynamic turbulent eddy viscosity
μ_t	= dynamic total viscosity
ν	= kinematic viscosity
ν_e	= kinematic turbulent eddy viscosity
ν_t	= kinematic total viscosity
ξ, η, ζ	= Cartesian axes for unit cube domain
ρ	= density
τ_{ij}	= shear stress in $i - j$ plane
ω	= vorticity

Operators

$D_{ij}(\vec{u})$	= strain rate tensor $\left(\frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right)$
$\vec{\nabla}$	= divergence operator $\left(\vec{e}_1 \frac{\partial}{\partial x_1} + \vec{e}_2 \frac{\partial}{\partial x_2} + \vec{e}_3 \frac{\partial}{\partial x_3} \right)$

$\vec{\nabla}_h$ = discrete divergence operator

∇^2 = Laplacian operator $\left(\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2} \right)$

∂ = partial differential operator

$\sum_{k=a}^k$ = sum of terms for $k = a$ to $k = b$

$\int_\Omega$ = integral over domain

$\int_S$ = integral over surface

$\int_{V_e}$ = integral over the element volume

$\int_{\Gamma_e}$ = integral over the element surface

Chapter 1

INTRODUCTION

1.1 Computational Fluid Dynamics

With the continued development of computers capable of storing larger amounts of data and processing this information at increasingly rapid rates, the realm of computations¹ engineering has widened considerably. This proliferation has allowed greater use of computational techniques as a powerful solution tool alongside the more conventional analytical and experimental methods.

The increase in the availability of computational solution packages for fluid dynamics analysis stems from qualities making these methods inherently attractive. Much of the analytical work of the past has been possible because of simplifying assumptions about the physics of the flow. The resulting constraints have restricted the range of applicability of these techniques to a relatively narrow band of problems. These approximations, such as one dimensionality or negligible viscosity, serve to provide estimates in relatively short times but do not provide a detailed picture of the flow behaviour. Once developed, however, the Computational Fluid Dynamics (CFD) approach requires understanding more of the effect of the problem's boundary conditions on the solution of the field than on which simplification method is appropriate to a particular class of problems. This, in essence, leaves the detailed solving of the governing equations to an instrument more suited to the task and allows the analyst to concentrate more on the physics and behaviour of the flow. Also, since any combination of variables may be easily analyzed by modifying input files, the need to interpolate on a series of empirical curves is eliminated. This flexibility is impressive when considering the high costs associated with model development and construction, test rig setup and calibration, as well as data acquisition and reduction which necessarily accompany experimental analysis.

The experimental techniques still in wide use will certainly continue to be required for the foreseeable future since CFD should be used only where results may be confidently

obtained. This means that computational tools should be used within a range bounded by experimentally confirmed results rather than trying to extrapolate from the applicable range. In essence then, the interpolation may be done intelligently using the physics of the problem rather than a series of graphs. In addition, CFD is presently inadequate for evaluating several physical phenomena such as turbulence and complex three dimensional effects. In these cases, experimental techniques must be used to obtain empirical correlations which may then be used as data for the CFD package. The turbulence model of Laufer, which was used during the course of this work to supply eddy viscosity values in pipe flow, is an example of one of these correlations.

Kutler [1985] and Chapman [1980] present lists of "pacing items" in CFD. Although these items have been slightly altered over the five year span separating publication of the two papers, they are very similar. Kutler's critical or "primary" pacing items are listed as:

- 1)three-dimensional grid generation;
- 2)turbulence modelling;
- 3)solution methodology development;
- 4)mainframe computer methodology; and
- 5)architectural advances.

The less critical "secondary" pacing items are:

- 1)algorithm development;
- 2)complex geometry definition; and
- 3)predata and postdata processing.

Recent literature shows that a great deal of effort is being put into overcoming the difficulties arising from these pacing items. The majority of this author's efforts may be classified in the areas of algorithm development as well as data pre- and post-processing.

The decision to develop new CFD tools at Carleton University was the result of a desire to gain more insight into the behaviour of various fluid flow problems. One of the most pressing of these problems was the analysis of weld pools generated by the Finite Element Method (FEM) package, ASGARD, developed and currently maintained here. It was noted

that the inclusion of a viscous fluid flow prediction scheme in this package would also be useful for the solution of other real engineering problems such as the prediction of total pressure losses in wind tunnel and turbomachinery components. The intention was then to develop this new ability using as general an algorithm as possible.

1.2 Previous Work

Although the possibility of using finite elements in the solution of the Navier-Stokes equation was considered in such research as Oden [1971] or Hood [1970] (see, for example, Taylor & Hood [1973]) the first successful efforts into developing a program to solve the $[K][x] = [B]$ equation set generated from the Navier-Stokes and continuity equations were carried out by Taylor & Hood [1973]. The work compared the relative advantages of the primitive variable formulation with the use of the stream function-vorticity approach and concluded that the primitive variable technique was the desirable course of action. This decision was based on several observations. By comparing computer solution times for contained flows in the low to medium Re range, it was found that the primitive variable approach had some speed advantages. The fact that the extension of the two dimensional case to three dimensions was more easily accomplished with the primitive variables operated in its favour. A final important factor was that the treatment of boundary conditions was by no means as evident using the stream function-vorticity approach.

Those authors also chose to investigate the use of a mixed formulation which sought to discretize the continuity equation and include it in the set of resulting equations from which a solution for both the pressure and velocity fields could be derived concurrently. This new method differed from the earlier penalty function technique used by Oden as well as Hood which calculated pressures using a Poisson equation once the momentum equations had been solved to yield a velocity field (a technique already in use in the more familiar structural analysis FEM solutions). The authors felt that the mixed formulation was the desirable course of action. Mixed interpolation reduced the bandwidth of the matrix which is desirable when using band solvers. The solver used in this research was a frontal solver, though, so the reduction in bandwidth was of little value. Another disadvantage of the penalty function approach which these authors quoted was that using the same order element for the pressure and velocity fields, the second derivatives of velocity had to be assumed zero. This was obviously true only in the case of very slow moving fluids

were of little interest. This disadvantage is circumvented by the use of different interpolation orders for the two fields such as was done with the 15 node brick as part of this research.

Researchers began to observe occasional spurious pressures which could not be attributed to mesh inadequacies or ill-posed boundary conditions. Hood & Taylor [1974] noted that the use of the same order interpolation functions in the eight node quadrilateral for both velocities and pressures seemed to be the cause of the problem and showed that a reduction in the order of the pressure interpolation functions from quadratic to linear resulted in a significant improvement in the prediction's accuracy. The corresponding velocity field suffered a small decrease in accuracy attributed to the fact that the imposition of the continuity equation at four nodes was a weaker condition than the previous eight node requirement and thus allowed more freedom for the velocity field to stray from the exact solution.

Hood & Taylor [1974] suggested this reduction in the order of the pressure interpolation functions because of a belief that, in the treatment of coupled equations, the residuals arising from each equation must be weighted (in a Galerkin sense) according to the maximum error occurring in that equation. Since the momentum equations contained second order velocity gradients and the continuity equation contained only first order gradients, they noted that the continuity equation should be weighted with functions one order lower than those used to weight the momentum equation. Not to increase the number of nodes unnecessarily, they decided to weight velocities with quadratics and pressures with linear shape functions.

This line of thought was possibly similar to that which led Babuska and Brezzi to define independently a mathematical condition which, if satisfied, would ensure compatible velocity and pressure fields (see M. Fortin [1981]). The abstract criterion became known as the Babuska-Brezzi condition and essentially says that one cannot mix together any approximations of velocity and pressure. Since the criterion was not explicitly examined by this author during the course of the research, it will not be quoted nor derived here. It should suffice to say that the Babuska-Brezzi condition has become an accepted metric for spurious modes and appears adequate to expose elements which suffer from these modes. M. Fortin [1977] (see, for example, M. Fortin [1981]) furthered the use of this complex mathematical condition by relating it to specific discrete velocity fields. In essence, he showed that if a discrete velocity field could be described which inherently satisfied the same mass balance as the continuous velocity field on an element basis, spurious pressures

could be eliminated. Thus, the problem was reduced to insuring that on any element the divergence, in an integral sense, of a discrete velocity field was equal to that of the continuous field.

For the two dimensional case, the most visibly deficient element was the 4 node quadrilateral which used linear velocity shape functions and a piecewise constant pressure. Detailed investigation into its now well documented "checkerboard" pressure mode began with Sani et al [1981] and Donea et al [1981]. A method of modifying this element to ensure that it implicitly satisfied the Babuska-Brezzi condition was suggested by M Fortin and A Fortin [1984a] which involved the addition of a barycentric node which was used to control the mass flow on an element basis. The word "barycentre" is defined as "pertaining to the centre of gravity" and the use of this term was adequate to describe the additional node since it was placed at the centre of the undistorted element geometry. The node was eventually eliminated from the equation set in a post-processing phase which permitted this improvement to be implemented with a relatively minor penalty in computational cost. This technique forms the basis of the fifteen node brick finally developed as part of this research.

This author began with a 20 node hierarchical brick originally developed by Vemaganti [1984] which used the velocity and pressure primitive variables as dof for the mixed formulation solution of both the Navier-Stokes and continuity equations. The element had been demonstrated to work for laminar, two-dimensional flows at very low Reynolds number (approximately $Re = 1$ to 5). No attempt had been made to incorporate a turbulence model although the structure required for its insertion had been considered. An initial attempt to demonstrate its three dimensional capabilities had been unsuccessful. The element had been developed with the ASGARD format in mind, but often strayed from its philosophy and lacked the generality characteristic of that package. In particular, the element used fixed array dimensions and indices which were either problem dependent or much larger than required which resulted in slow solutions for even simple geometries.

1.3 Aim & Scope of the Present Work

The aim of this research was to continue development of a FEM package for predicting three dimensional, viscous, incompressible flows. It was to be applied in either the laminar

or the turbulent regime but not necessarily in the compressible range. The code was to be used mainly for internal flows owing to the great success of potential methods in the solution of largely inviscid external flows. The routines were to be written as part of the much larger FEM analysis package, ASGARD. The work was not intended to move from one point to a further advanced position in a straight line pattern. Rather, the work solved a number of parallel problems whose solution will facilitate future developments.

As stated earlier, the work began by improving upon the 20 node brick developed by Vemaganti [1984] as part of his M. Eng thesis. Since the development had not been completed, the first priority was to incorporate the existing element into the ASGARD package. In order to improve the element's slow run time, several modifications were made to the frontal solver. These improvements reduced the run times drastically as can be seen by the bar chart included as Figure 3. A separate improvement to the solution time was achieved by allowing an initial guess at the solution to the flow field to be specified. This initial guess could significantly reduce the time required to solve a complex problem and a good guess could often be arrived at using, for example, inviscid flow theory.

As suggested by the program requirements, the next step was to develop a routine providing turbulence values which the element stiffness generator would treat as a "black box". In this way, various turbulence models could be used during the course of development with no changes required to the rest of the program. To begin this development, it was decided to use empirical pipe flow data which would return an eddy viscosity value corresponding to the radial position of the node. The routine created was based upon data collected by Laufer [1954]. The eddy viscosity value at the Gauss point and its gradients are then computed in much the same way as the velocity, for example, by using the interpolation functions.

The element's performance was then to be enhanced by developing the capability to correctly predict the behaviour of three dimensional flow problems. The case selected for testing was a square duct with a 30 degree bend. The reason for this choice is explained in Chapter 5. In order to solve this problem, however, a downstream boundary condition was required. It was decided to approximate the downstream end by imposing zero streamwise acceleration for the final elements. This condition required the development of a special element designed to slave the downstream nodes to their upstream partners by rearranging the element stiffness matrix.

During the course of this development, it was found that the mass of information returned from the FEM program was difficult to interpret and slowed progress considerably. Since the solutions were iterative, a truly nonlinear problem might require some five to fifteen iterations to converge. Given that the problem might consist of hundreds of nodes, many of which were fixed by the imposed boundary conditions, the progress of the solution was difficult to track. To ease the task of development, an output file post-processor was developed which reduced the information to manageable levels. This program, called SEPARATE, is described in more detail later in the report.

In Chapter 5, there is a discussion of the relative orders of magnitude of the terms which comprise the 20 node brick element stiffness matrix. It was found through the course of development that the convection terms, although often summing to zero along a particular row when in a convection free field, would overshadow the contributions of the more important terms. This had adverse effects on the accuracy of the predictions returned by the element and it was decided to minimize this effect. In an effort to do so, it was decided to sum the convection terms separate from the $[K]$ matrix and to subtract these terms from the equation second member. In this way, if the convection effects were non-existent or negligible, they would not affect the accuracy of the solution. Although this technique worked very well for truly convection free fields, it was found to be unstable for even mildly convective fields. The approach may be likened to Taylor & Hughes [1973] when they tried summing into the second member the $\nabla^2 \omega$ term used in the stream function-vorticity technique. In doing so, they also experienced a numerical instability. It may be surmised, then, that this method is not a desirable course of action for future development.

As evidence was amassed which indicated that the 20 node brick suffered from spurious pressure modes, a new element was sought which could be relied upon to return consistently correct pressure values. The work of M. Fortin and A. Fortin was then studied since a large part of their research had been dedicated to developing elements satisfying the Babuska-Brezzi condition. Discussions with A. Fortin revealed that a particular type of fifteen node brick satisfied the Babuska-Brezzi condition and should prove to be cost effective and accurate. Thus, the fifteen node brick was developed to the point of showing its ability to predict simple flows.

Chapter 2 of this report deals with some of the interesting aspects of finite elements which were used during the course of the research. The derivation and subsequent solution

of the working equations for each element is described in Chapter 3. Chapter 4 goes on to describe the FORTRAN code written to solve the equations derived in Chapter 3 as well as the post-processor described above. The test cases used and results obtained are described in Chapter 5. Finally, Chapter 6 details the conclusions reached and proposes some areas of future work.

For someone attempting to continue this research, an introduction to ASGARD is provided as an appendix. This would certainly have reduced the time required for the author to become knowledgeable enough to begin altering aspects of the code. Also included as appendices are listings of the source code written or modified during the work.

Chapter 2

THE FINITE ELEMENT METHOD

2.1 Introduction

In order to model computationally the momentum and mass flow constraints imposed by the governing Navier-Stokes and continuity equations respectively, the FEM employs either: 1) a mixed formulation; or 2) a penalty method formulation. The mixed formulation uses the discretized continuity equation as part of the element stiffness matrix in order to solve for the pressure field. To that end, Taylor & Hood [1973] and Taylor & Hughes [1981] juxtaposed a velocity field with a pressure field using different element types. For example, Taylor & Hughes [1981] solved two dimensional problems using 8 node quadrilaterals to discretize the velocity field and 4 node quadrilaterals for the pressure field. Therefore, the problem space was discretized once for the momentum constraints and a second time for the continuity condition. In this procedure, the coupling of the two fields was the constraint necessary to ensure a unique solution. Since the equation set is elliptic, the cost associated with such a wasteful procedure is high.

To circumvent this redundancy, it was decided to use a hierarchical formulation. Here, the problem is discretized into only one calculation domain and the pressures are coupled to the velocities either by increasing the number of dof at some of the nodes or by using an element level polynomially distributed pressure field. The result is that the velocities at certain nodes take on a new form in order to satisfy the use of a particular set of basis functions.

The 20 node and 15 node bricks considered in this thesis are both hierarchical elements. Although the elements display certain similarities, the formulations used to solve the governing equations were vastly different. At the most obvious level, the dof vector solved for and the treatment of the diffusion terms were different. Also, the 20 node brick used a mixed formulation whereas the 15 node brick used a penalty function to impose the continuity condition.

It is assumed the reader has a working knowledge of the basics of the FEM. The discretization and interpolation procedures resulting from a shape function approach will not be discussed in this report and the interested reader is referred to Irons & Ahmad [1984], Bathe [1982], or Zienkiewicz [1979]. More specific topics such as hierarchical element shape functions or the achieving of desired effects by reorganization of element stiffness matrices will be discussed in greater detail, though, since they are not clearly described in textbooks nor previously documented in the ASGARD code.

2.2 Hierarchical Elements

The hierarchical formulation is a technique used to increase the order of interpolation of an element while preserving the original basis functions. This is accomplished merely by adding extra nodes at appropriate locations. For both the 20 node and 15 node bricks, the technique was used to allow the use of a linear pressure distribution and quadratic velocities.

In order to understand the relationship between the pressure and velocity basis functions using for example, the 20 node element, begin by considering the 8 node brick. The interpolation functions are trilinear which would restrict both pressures and velocities to this order if it were used for the solution of the Navier-Stokes equations. This choice has been shown to produce poor pressure fields which implies either reducing the order of the pressure interpolation function or increasing that of the velocity field.

Discretizing a physical domain into a FEM mesh weakens the strength of the governing equations since they need not be obeyed at all points but rather, obeyed in an integral sense. The choice of which order interpolation to use is an important one since the choice essentially restricts the discrete solution to this order. Thus, although the general solution may allow the velocity field in the case above, for example, to be represented by any order polynomial, the discrete problem constrains it to a linear distribution on an element basis. A clear example of this effect was documented by Gresho & Leone [1984] (see M. Fortin & A. Fortin [1984a]) using the "no-flow test". The element used allowed linear velocity fields and piecewise constant pressures on an element basis. Thus, at an element boundary, the velocity was C^0 continuous. The pressure was C^{-1} continuous and constant on any element. The test problem consisted of a closed body with only body forces present. This should

result in a zero velocity field and a pressure distribution linear in the x_2 direction such as in the case of hydrostatics. Since the element chosen allowed only piecewise constant pressures, this exact solution was not attainable and spurious velocity vortices were seen. Although the field should be an easy one to reproduce as a result of the easily modelled boundary conditions and geometry, the order of the pressure solution did not lie in the discrete solutions allowed and the test failed.

The choice of whether to increase the velocity interpolation order of the 20 node brick or reduce the pressure's is then a simple decision. It is clearly advantageous to increase the velocity interpolation order since a wider solution space reduces the possibility of having a solution lie outside the set of available functions. Thus, twelve midedge nodes are added to the existing 8 node brick and the basis functions associated with these nodes are identically equal to the corresponding 20 node brick midedge functions.

The usual FEM interpolation procedure requires the sum of the basis functions to be unity and the sum of the basis function derivatives to be zero which essentially describes a rigid body motion or its equivalent in the fluid case. Bearing in mind this restriction on the interpolate sums, the midedge dof are found to represent deviations from the linear interpolation prediction at the midedge position. Then, if the velocity at the midedge node is the same as the two corner velocities such as in the case of a bounding wall where the three velocities are zero, the coefficient for this node is zero. Therefore, the midedge dof value can be found by using the bounding corner nodes to calculate a linearly interpolated velocity at the midedge position then subtracting this value from the absolute velocity at that position.

The 15 node brick is constructed in a similar way. Beginning with an 8 node brick, additional nodes are placed at the six midfaces and one at the barycentre. As mentioned earlier, M. Fortin [1981] showed that certain discrete velocity fields could be constructed which implicitly satisfied the Babuska-Brezzi condition. The primary concern with these velocity fields was the presence of a node at the midedge in two dimensions or at the midface in three. This node's normal component of velocity could be manipulated in such a way as to ensure the divergence free condition on each element. To combine this approach with the advantages of the hierarchical method described earlier, the midface nodes' dof were given a very specific significance. These dof were made to be deviations of the velocity normal to the face at the node from a linearly interpolated value using the dof of the four bounding

corner nodes.

It is interesting to note the savings in the number of shape functions which require evaluation when using this hierarchical approach. The three dimensional equivalent of the 8 node and 4 node quadrilateral mixed formulation used by Taylor & Hughes [1981] requires the use of twenty-eight shape functions per element. These come from the nodes of the 20 node and 8 node bricks used for velocity and pressure fields respectively. The corresponding 20 node brick hierarchical formulation requires eight less shape functions since those associated with the element's corner nodes are identical to the eight corner node shape functions used for the pressure formulation. The number of dof for these two cases is unchanged, though, so there is no loss of information with the hierarchical approach. The shape functions and derivatives are listed in Tables 1 & 2 for the 20 node brick and in Tables 3 & 4 for the 15 node brick.

2.3 Rigid Body Mode Analogue

A tool in wide use in the stress analysis aspect of FEM has been the verification of the rigid body modes. In essence, the satisfaction of the rigid body mode conditions has provided a simple, yet infallible, preliminary test of the correctness of newly generated $[K]$ matrices. The fluid flow case has an analogue to this condition and provided the author with a good method to begin testing of elemental $[K]$ matrices.

The condition essentially states that imposing a displacement field corresponding to one of the rigid body modes on an object implies no forces are generated. This corresponds to the case of physically transporting the object as a whole from one point to another or rotating it about some point. These conditions obviously give rise to a stress free state. The usefulness of this technique is in its application to the rows of the $[K]$ matrix. Consider imposing some translation to an object. Since any row in the $[K]$ matrix is multiplied by the dof vector, the sum of the terms along the rows must be zero. In addition, the sum of the terms multiplying the dof representing displacements in the same direction must also be zero since this implies the special case where the object translation is along one axis only.

Thus, for elements with non-hierarchical spatial dof, the sum of all the terms in any row of the elemental $[K]$ matrix must be zero; the sum of the $[K_{ij}]$ terms multiplying the

dof in the i^{th} direction must also be zero. For example, a dof vector for a three dimensional eight node brick element would require:

$$\sum_{j=1}^{24} [K_{i,j}] = 0.0$$

$$\begin{aligned} K_{1,1} + K_{1,4} + K_{1,7} + K_{1,10} + K_{1,13} + K_{1,16} + K_{1,19} + K_{1,22} &= 0.0 \\ K_{1,2} + K_{1,5} + K_{1,8} + K_{1,11} + K_{1,14} + K_{1,17} + K_{1,20} + K_{1,23} &= 0.0 \\ K_{1,3} + K_{1,6} + K_{1,9} + K_{1,12} + K_{1,15} + K_{1,18} + K_{1,21} + K_{1,24} &= 0.0 \end{aligned} \quad (2.1)$$

An analogue to this condition may be realized for the fluid element. Fluid stresses also result from velocity gradients and not from the absolute values of velocity. Thus, giving each node of a fluid element a unit velocity imposes no stresses and the same condition of a zero sum applies. It must be borne in mind, though, that these conditions apply only to the non-hierarchical dof. This is because a unit value for the hierarchical dof does not imply a no-shear condition. The unit value actually represents a shear in relation to the bounding corner nodes. The equivalent rigid body mode is therefore that the sums of the appropriate $[K]$ terms for the corner nodes only must be zero. This condition is equivalent to a unit velocity applied to the entire element.

This check was used extensively in the initial $[K]$ matrix development for both the 20 and 15 node bricks. Chapter 4 refers to the routine which was used to carry out the necessary summations. The routine was called STIFCHEK.

2.4 Description of the 20 Node Brick

The 20 node brick is a three dimensional element consisting of eight corner and twelve midedge nodes as shown in Figure 1. The shape functions used with this particular element are noticeably different from those normally associated with the 20 node brick (see Section 2.2). Each of the corner nodes has four degrees of freedom while the midedge nodes have three. This difference results from the mixed formulation used and more particularly, from the fact that the continuity equation is discretized at only the corner nodes and hence the pressure dof are evaluated only at these nodes. It follows from these definitions of dof that the velocity and pressure fields are C^0 continuous.

The mixed formulation technique was used to solve for the velocity and pressure fields. The size of the element stiffness matrix may be easily shown to be $[68 \times 68]$ of which sixty dof are velocities and eight are pressures. This is a very large set of elemental equations (the largest currently implemented in ASGAR) and, since solution time is inversely proportional to the number of equations and to the square of the frontwidth, obviously results in relatively slow solution of problems. As part of the work carried out, the particular effect of imposing zero streamwise acceleration was achieved by rearranging and reducing the size of the element stiffness matrix to $[60 \times 60]$. This new condition allowed much quicker solution since the solver could then fix many variables to slaved values (see Section 3.3).

2.5 Description of the 15 Node Brick

The 15 node brick has eight corner nodes, one barycentric node, and six midface nodes. The element topology chosen for this element is shown in Figure 2. The velocity is parabolic and C^0 continuous as a result of the hierarchical formulation. The pressure distribution is linear and the pressure is discontinuous (i.e. C^{-1} continuous). The shape functions associated with the corner nodes correspond exactly to those used by the 8 node brick. The barycentric node uses a three dimensional bubble function with hierarchical dof. The midface node basis functions are two dimensional bubble functions on the associated face and linear in the third dimension. The dof associated with these nodes are also hierarchical and represent deviations of velocities normal to the face at the node. From the topology, it may be deduced that the element stiffness matrix dimensions are $[33 \times 33]$ from the twenty four corner node velocity components, three barycentric velocities, and six velocities from the midface nodes.

As mentioned above, the dof of a midface node is the velocity normal to the appropriate face in real space. This velocity may be decomposed into the more conventional Cartesian velocities by using the three direction cosines of the vector normal to the face at the midface node. The resulting components are then equivalent to the three barycentric hierarchical dof and are also velocity deviations from linear interpolation predictions. The direction cosines are calculated once in some pre-processing code by using the cross product of the appropriate rows of the Jacobian matrix of the isoparametric transformation at that node.

Chapter 3

DERIVATION OF WORKING EQUATIONS

3.1 Introduction

In this chapter, the working equations for both elements will be derived to the point of writing a generalized $[K_{ij}]$ submatrix from whence the necessary code could be written. Although the 20 node brick failed to correctly predict accurate pressure fields, the shortcoming was due to its failure to satisfy the Babuska-Brezzi condition and not on the particular discretization and solution procedure used. The equations derived below are therefore of interest for use with a similar element satisfying the Babuska-Brezzi condition.

In the derivations below, the indicial notation is used in a fairly standard way. A quick example may be valuable, though, to ensure the convenience of the notation does not obscure the meaning of the equations. Indicial notation is used to write the sum of many terms in a compact form. The indices are meant to be incremented over the number of axes of interest. Thus, for a two dimensional problem, the indices i and j would take on values 1 and 2; for a three dimensional problem, they would be 1, 2, then 3. The summation described then implies adding all combinations of the variables using the successive indices. As an example, take the first term in the 20 node brick section of this chapter. The term would be expanded in three dimensions as:

$$u_j \frac{\partial u_i}{\partial x_j} = u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2} + u_3 \frac{\partial u_1}{\partial x_3} + u_1 \frac{\partial u_2}{\partial x_1} + u_2 \frac{\partial u_2}{\partial x_2} + u_3 \frac{\partial u_2}{\partial x_3} + u_1 \frac{\partial u_3}{\partial x_1} + u_2 \frac{\partial u_3}{\partial x_2} + u_3 \frac{\partial u_3}{\partial x_3}$$

The author has tried to use common notation for the derivations of the equations used with both elements. Since the methods used in each element are vastly different and the literature surrounding each of these two types use completely different symbols, this has proven difficult. Even more evident than small nomenclature changes between the two derivations is the predominant use of indicial notation for the 20 node brick and vector notation for the 15 node element. Since both of these are equally useful for expressing

vector equations, using either where appropriate to follow the literature surrounding each element seemed justified.

3.2 20 Node Brick Mixed Formulation

Looking first at a force balance on an elemental volume of fluid, for steady flow the Navier-Stokes equations may be derived using indicial notation to give:

$$u_j \frac{\partial u_i}{\partial x_j} = F_i + \frac{1}{\rho} \frac{\partial}{\partial x_j} (\tau_{ij}) \quad (3.1)$$

Now, again on an elemental volume of fluid, performing a mass balance yields the continuity equation in indicial notation:

$$\frac{\partial u_i}{\partial x_i} = 0 \quad (3.2)$$

The shear stress in equation (3.1) may be evaluated by using a transformation to the principal stress space and then taking advantage of certain properties of the space and the direction cosines. The stress can be written as the sum of the pressure term (for the normal stress case only), a Newtonian stress, and the Reynolds stress as follows:

$$\tau_{ij} = -p\delta_{ij} + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \overline{\rho u'_i u'_j} \quad (3.3)$$

A common assumption originally proposed by Boussinesq (see, for example, White [1974]) is that the Reynolds stress terms act in a manner similar to the Newtonian stresses, but rather than using the dynamic viscosity, the multiplying term is that of a variable eddy viscosity. Thus, the Reynolds stress terms can be expressed as:

$$-\overline{\rho u'_i u'_j} = \mu_e \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (3.4)$$

Substituting equation (3.4) into equation (3.3) yields:

$$\tau_{ij} = -p\delta_{ij} + \mu_e \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (3.5)$$

Here, a total viscosity has been defined as a combination of dynamic and eddy viscosities:

$$\mu_t = \mu_e + \mu \quad (3.6)$$

Up to this point, the formulation is not controversial since a total viscosity distribution which satisfies the flow field can always be defined. Substituting equation (3.5) into equation (3.1) allows an expansion from whence the FEM equations may be derived. The newly derived Navier-Stokes equation set is written below:

$$-F_i = \frac{-1}{\rho} \frac{\partial p}{\partial x_i} \delta_{ij} - u_j \frac{\partial u_i}{\partial x_j} + \frac{\partial}{\partial x_j} \left(\nu_t \frac{\partial u_i}{\partial x_j} \right) + \frac{\partial u_i}{\partial x_j} \frac{\partial \nu_t}{\partial x_j} \quad (3.7)$$

Now, this set of equations can be expanded, using $i = 1$ for example, to the following form:

$$\begin{aligned} -F_1 = & \frac{-1}{\rho} \frac{\partial p}{\partial x_1} - \left[u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2} + u_3 \frac{\partial u_1}{\partial x_3} \right] + \\ & + \frac{\partial}{\partial x_1} \left(\nu_t \frac{\partial u_1}{\partial x_1} \right) + \frac{\partial}{\partial x_2} \left(\nu_t \frac{\partial u_1}{\partial x_2} \right) + \frac{\partial}{\partial x_3} \left(\nu_t \frac{\partial u_1}{\partial x_3} \right) + \\ & + \frac{\partial u_1}{\partial x_1} \frac{\partial \nu_t}{\partial x_1} + \frac{\partial u_2}{\partial x_1} \frac{\partial \nu_t}{\partial x_2} + \frac{\partial u_3}{\partial x_1} \frac{\partial \nu_t}{\partial x_3} \end{aligned} \quad (3.8)$$

The total viscosity, ν_t , can be decomposed into eddy and kinematic viscosity contributions in a manner analogous to the total dynamic viscosity term shown in equation (3.6):

$$\nu_t = \nu + \nu_e \quad (3.9)$$

Taking advantage of this fact to expand equation (3.8) yields:

$$\begin{aligned}
 -F_1 = & \frac{-1}{\rho} \frac{\partial p}{\partial x_1} - \left[u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2} + u_3 \frac{\partial u_1}{\partial x_3} \right] + \\
 & + \frac{\partial}{\partial x_1} \left(\nu \frac{\partial u_1}{\partial x_1} \right) + \frac{\partial}{\partial x_2} \left(\nu \frac{\partial u_1}{\partial x_2} \right) + \frac{\partial}{\partial x_3} \left(\nu \frac{\partial u_1}{\partial x_3} \right) + \\
 & + \frac{\partial}{\partial x_1} \left(\nu_e \frac{\partial u_1}{\partial x_1} \right) + \frac{\partial}{\partial x_2} \left(\nu_e \frac{\partial u_1}{\partial x_2} \right) + \frac{\partial}{\partial x_3} \left(\nu_e \frac{\partial u_1}{\partial x_3} \right) + \\
 & + \frac{\partial u_1}{\partial x_1} \frac{\partial \nu_e}{\partial x_1} + \frac{\partial u_2}{\partial x_1} \frac{\partial \nu_e}{\partial x_2} + \frac{\partial u_3}{\partial x_1} \frac{\partial \nu_e}{\partial x_3}
 \end{aligned} \tag{3.10}$$

Using the Chain Rule to further expand the expression yields an equation comprised of several terms. Many of these terms may be eliminated since the kinematic viscosity can be assumed near constant as a function of pressure alone and that no large thermal gradients are permitted. This being assumed, the kinematic viscosity derivatives are zero.

Omitting the appropriate terms and grouping those of similar form, the expression may be simplified to:

$$\begin{aligned}
 -F_1 = & \frac{-1}{\rho} \frac{\partial p}{\partial x_1} - \left[u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2} + u_3 \frac{\partial u_1}{\partial x_3} \right] + \\
 & + \nu \left[\frac{\partial^2 u_1}{\partial x_1^2} + \frac{\partial^2 u_1}{\partial x_2^2} + \frac{\partial^2 u_1}{\partial x_3^2} \right] + \nu_e \left[\frac{\partial^2 u_1}{\partial x_1^2} + \frac{\partial^2 u_1}{\partial x_2^2} + \frac{\partial^2 u_1}{\partial x_3^2} \right] + \\
 & + \left[\frac{\partial u_1}{\partial x_1} \frac{\partial \nu_e}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \frac{\partial \nu_e}{\partial x_2} + \frac{\partial u_1}{\partial x_3} \frac{\partial \nu_e}{\partial x_3} \right] + \\
 & + \left[\frac{\partial u_1}{\partial x_1} \frac{\partial \nu_e}{\partial x_1} + \frac{\partial u_2}{\partial x_1} \frac{\partial \nu_e}{\partial x_2} + \frac{\partial u_3}{\partial x_1} \frac{\partial \nu_e}{\partial x_3} \right]
 \end{aligned} \tag{3.11}$$

In order to shorten the description of the equation, the Laplacian operator is used. Also, to make the grouping of the terms multiplying the viscosities more apparent, the sequence is reordered as:

$$\begin{aligned}
 -F_1 = & \frac{-1}{\rho} \frac{\partial p}{\partial x_1} - \left[u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2} + u_3 \frac{\partial u_1}{\partial x_3} \right] + \\
 & + [\nu + \nu_e] \nabla^2 u_1 + \frac{\partial \nu_e}{\partial x_1} \left[2 \frac{\partial u_1}{\partial x_1} \right] + \\
 & + \frac{\partial \nu_e}{\partial x_2} \left[\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \right] + \frac{\partial \nu_e}{\partial x_3} \left[\frac{\partial u_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1} \right]
 \end{aligned} \tag{3.12}$$

3.2.1 Solution Algorithm

The FEM discretization approximates the value of a variable at a point as the sum of the products of the discrete nodal values and the appropriate shape function magnitude evaluated at the desired point. The hierarchical formulation used specifies that the pressure discretization use only the eight corner nodes of the brick and that each of the velocities use all twenty nodes (see Section 2.3). Therefore, the pressure and velocities are discretized as:

$$\begin{aligned}
 u_1(\xi, \eta, \zeta) &= \sum_{i=1}^8 M_i(\xi, \eta, \zeta) u_{1i} + \sum_{i=9}^{20} S_i(\xi, \eta, \zeta) \Delta u_{1i} \\
 u_2(\xi, \eta, \zeta) &= \sum_{i=1}^8 M_i(\xi, \eta, \zeta) u_{2i} + \sum_{i=9}^{20} S_i(\xi, \eta, \zeta) \Delta u_{2i} \\
 u_3(\xi, \eta, \zeta) &= \sum_{i=1}^8 M_i(\xi, \eta, \zeta) u_{3i} + \sum_{i=9}^{20} S_i(\xi, \eta, \zeta) \Delta u_{3i} \\
 p(\xi, \eta, \zeta) &= \sum_{i=1}^8 M_i(\xi, \eta, \zeta) p_i
 \end{aligned} \tag{3.13}$$

Here, the shape functions corresponding to the corner nodes have been designated by $M_i(\xi, \eta, \zeta)$ and those corresponding to the midedges have been referred to as $S_i(\xi, \eta, \zeta)$ in order to accentuate the difference between the two. The M_i, S_i distinction is an important one but, having made the difference apparent, the notation will be simplified to define the summation over corner nodes only as $\sum M_i$ and over all twenty nodes as $\sum N_i$. Thus, the summations of equation (3.13) may be rewritten as:

$$\begin{aligned}
 u_1(\xi, \eta, \zeta) &= \sum_{i=1}^n N_i(\xi, \eta, \zeta) u_{1i} \\
 u_2(\xi, \eta, \zeta) &= \sum_{i=1}^n N_i(\xi, \eta, \zeta) u_{2i} \\
 u_3(\xi, \eta, \zeta) &= \sum_{i=1}^n N_i(\xi, \eta, \zeta) u_{3i} \\
 p(\xi, \eta, \zeta) &= \sum_{i=1}^m M_i(\xi, \eta, \zeta) p_i
 \end{aligned} \tag{3.14}$$

From the previous discussion, the value of m is clearly eight and the value of n is twenty. Using the summations of equation (3.14) and the fact that equivalent expressions can be written for derivatives of these variables, the Navier-Stokes equations (equation (3.12)) may be re-written as:

$$\begin{aligned}
 -\sum_{j=1}^n N_j F_{1j} = & \quad -\frac{1}{\rho} \sum_{j=1}^m \frac{\partial M_j}{\partial x_1} p_j + 2 \frac{\partial \nu_e}{\partial x_1} \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{1j} - \\
 & - \left[\sum_{k=1}^n N_k u_{1k} \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{1j} + \sum_{k=1}^n N_k u_{2k} \sum_{j=1}^n \frac{\partial N_j}{\partial x_2} u_{1j} + \sum_{k=1}^n N_k u_{3k} \sum_{j=1}^n \frac{\partial N_j}{\partial x_3} u_{1j} \right] + \\
 & + [\nu + \nu_e] \left[\sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_1^2} u_{1j} + \sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_2^2} u_{1j} + \sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_3^2} u_{1j} \right] + \\
 & + \frac{\partial \nu_e}{\partial x_2} \left[\sum_{j=1}^n \frac{\partial N_j}{\partial x_2} u_{1j} + \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{2j} \right] + \frac{\partial \nu_e}{\partial x_3} \left[\sum_{j=1}^n \frac{\partial N_j}{\partial x_3} u_{1j} + \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{3j} \right] \quad (3.15)
 \end{aligned}$$

The Galerkin Weighted Residual technique is used to solve this equation. The technique specifies that the weighting functions used be the nodal shape functions themselves. Thus, equation (3.15) is written as:

$$\begin{aligned}
 \sum_i \int_{V^e} N_i \left[\sum_{k=1}^n N_k u_{1k} \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{1j} + \sum_{k=1}^n N_k u_{2k} \sum_{j=1}^n \frac{\partial N_j}{\partial x_2} u_{1j} + \sum_{k=1}^n N_k u_{3k} \sum_{j=1}^n \frac{\partial N_j}{\partial x_3} u_{1j} \right] - \\
 - \sum_{j=1}^n N_j F_{1j} + \frac{1}{\rho} \sum_{j=1}^m \frac{\partial M_j}{\partial x_1} p_j - 2 \frac{\partial \nu_e}{\partial x_1} \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{1j} - \\
 - [\nu + \nu_e] \left[\sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_1^2} u_{1j} + \sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_2^2} u_{1j} + \sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_3^2} u_{1j} \right] - \\
 - \frac{\partial \nu_e}{\partial x_2} \left[\sum_{j=1}^n \frac{\partial N_j}{\partial x_2} u_{1j} + \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{2j} \right] - \frac{\partial \nu_e}{\partial x_3} \left[\sum_{j=1}^n \frac{\partial N_j}{\partial x_3} u_{1j} + \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{3j} \right] dV^e = 0 \quad (3.16)
 \end{aligned}$$

Using the same discretization and minimization procedure as above, the continuity equation (equation (3.2)) may be expressed as:

$$\sum_i \int_{V^e} M_i \left[\sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{1j} + \sum_{j=1}^n \frac{\partial N_j}{\partial x_2} u_{2j} + \sum_{j=1}^n \frac{\partial N_j}{\partial x_3} u_{3j} \right] dV^e = 0 \quad (3.17)$$

In the two discretized equations above, the summation symbol preceeding the integral sign is used to designate the sum of all contributions from elements which include node i .

In order to reduce the second order diffusion terms in equation (3.16) to first order while simultaneously incorporating a required boundary condition, Green's theorem is invoked. Thus, these diffusion terms may be expanded as shown:

$$\begin{aligned} \int_V N_i \left[\sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_1^2} u_{1j} - \sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_2^2} u_{1j} - \sum_{j=1}^n \frac{\partial^2 N_j}{\partial x_3^2} u_{1j} \right] dV = \int_{\Gamma} N_i \sum_{j=1}^n \frac{\partial N_j}{\partial n} u_{1j} d\Omega - \\ - \int_V \left[\frac{\partial N_i}{\partial x_1} \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} u_{1j} + \frac{\partial N_i}{\partial x_2} \sum_{j=1}^n \frac{\partial N_j}{\partial x_2} u_{1j} + \frac{\partial N_i}{\partial x_3} \sum_{j=1}^n \frac{\partial N_j}{\partial x_3} u_{1j} \right] dV \end{aligned} \quad (3.18)$$

Using equations (3.18) and (3.16) yields a large system of equations which is evaluated using Gaussian quadrature rules. The number of Gauss points required may be determined by methods described in Irons & Ahmad [1984] or Bathe [1982]. It is sufficient here to say that a 14 point non-product rule was found to be adequate for the quadratic velocity field of the element. Then, each term in the integral set was evaluated at the Gauss points, multiplied by the Gaussian weighting factor and the determinant of the Jacobian at that point. In this well established procedure, the sum of the terms is the value of the integral.

The large, discrete equation may be expressed in the convenient matrix form $[K][x] = [B]$. Here, $[x]$ is the dof vector consisting of three velocities for all nodes and a supplementary pressure dof for the corner nodes. The $[B]$ vector is the forcing function which consists only of the body force term. The $[K]$ matrix is the element stiffness matrix and is made up of all terms multiplying the individual dof. A 4×4 submatrix can be written as shown below for each corner node. In the case of a midedge node, the fourth row and column of the submatrix are removed. The $[K_{ij}]$ entry is weighted by functions of the i^{th} node acting on the j^{th} nodal variable.

$$[K_{ij}] \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ p \end{pmatrix} = [B] \quad (3.19)$$

$$[K_{ij}] = \begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & K_{1,4} \\ K_{2,1} & K_{2,2} & K_{2,3} & K_{2,4} \\ K_{3,1} & K_{3,2} & K_{3,3} & K_{3,4} \\ K_{4,1} & K_{4,2} & K_{4,3} & K_{4,4} \end{pmatrix} \quad (3.20)$$

where -

$$[K_{1,1}] = \left(u_1^* N_i \frac{\partial N_j}{\partial x_1} + u_2^* N_i \frac{\partial N_j}{\partial x_2} + u_3^* N_i \frac{\partial N_j}{\partial x_3} \right) + \nu_i \left(\frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_2} \frac{\partial N_j}{\partial x_2} + \frac{\partial N_i}{\partial x_3} \frac{\partial N_j}{\partial x_3} \right) - \left(2N_i \frac{\partial N_j}{\partial x_1} \frac{\partial \nu_i}{\partial x_1} + N_i \frac{\partial N_j}{\partial x_2} \frac{\partial \nu_i}{\partial x_2} + N_i \frac{\partial N_j}{\partial x_3} \frac{\partial \nu_i}{\partial x_3} \right)$$

$$[K_{1,2}] = -N_i \frac{\partial N_j}{\partial x_1} \frac{\partial \nu_i}{\partial x_2}; \quad [K_{1,3}] = -N_i \frac{\partial N_j}{\partial x_1} \frac{\partial \nu_i}{\partial x_3}; \quad [K_{1,4}] = \frac{1}{\rho} N_i \frac{\partial M_j}{\partial x_1}$$

$$[K_{2,1}] = -N_i \frac{\partial N_j}{\partial x_2} \frac{\partial \nu_i}{\partial x_1}$$

$$[K_{2,2}] = \left(u_1^* N_i \frac{\partial N_j}{\partial x_1} + u_2^* N_i \frac{\partial N_j}{\partial x_2} + u_3^* N_i \frac{\partial N_j}{\partial x_3} \right) + \nu_i \left(\frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_2} \frac{\partial N_j}{\partial x_2} + \frac{\partial N_i}{\partial x_3} \frac{\partial N_j}{\partial x_3} \right) - \left(N_i \frac{\partial N_j}{\partial x_1} \frac{\partial \nu_i}{\partial x_1} + 2N_i \frac{\partial N_j}{\partial x_2} \frac{\partial \nu_i}{\partial x_2} + N_i \frac{\partial N_j}{\partial x_3} \frac{\partial \nu_i}{\partial x_3} \right)$$

$$[K_{2,3}] = -N_i \frac{\partial N_j}{\partial x_2} \frac{\partial \nu_i}{\partial x_3}; \quad [K_{2,4}] = \frac{1}{\rho} N_i \frac{\partial M_j}{\partial x_2}$$

$$[K_{3,1}] = -N_i \frac{\partial N_j}{\partial x_3} \frac{\partial \nu_i}{\partial x_1}; \quad [K_{3,2}] = -N_i \frac{\partial N_j}{\partial x_3} \frac{\partial \nu_i}{\partial x_2}$$

$$[K_{3,3}] = \left(u_1^* N_i \frac{\partial N_j}{\partial x_1} + u_2^* N_i \frac{\partial N_j}{\partial x_2} + u_3^* N_i \frac{\partial N_j}{\partial x_3} \right) + \nu_i \left(\frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_2} \frac{\partial N_j}{\partial x_2} + \frac{\partial N_i}{\partial x_3} \frac{\partial N_j}{\partial x_3} \right) - \left(N_i \frac{\partial N_j}{\partial x_1} \frac{\partial \nu_i}{\partial x_1} + N_i \frac{\partial N_j}{\partial x_2} \frac{\partial \nu_i}{\partial x_2} + 2N_i \frac{\partial N_j}{\partial x_3} \frac{\partial \nu_i}{\partial x_3} \right)$$

$$[K_{3,4}] = \frac{1}{\rho} N_i \frac{\partial M_j}{\partial x_3}$$

$$[K_{4,1}] = M_i \frac{\partial N_j}{\partial x_1}; \quad [K_{4,2}] = M_i \frac{\partial N_j}{\partial x_2}; \quad [K_{4,3}] = M_i \frac{\partial N_j}{\partial x_3}$$

$$[K_{4,4}] = 0$$

The summations over index k in equation (3.16) are done using the velocity field obtained from the previous iteration. This essentially determines the trial velocity at the Gauss point in the current iteration. The $*$ notation is used to designate a trial variable.

In the equations above, the velocities u_1^i , u_2^i , and u_3^i are the velocities at the Gauss point. These velocities are found from the velocity field obtained from the previous iteration and are obviously the source of non-linearity in the equation. Thus, a velocity field is originally chosen then updated through successive iterations until the relative change from one iteration to the next is smaller than some chosen tolerance. Also obvious from the equation is the asymmetric nature of the stiffness matrix. Turning attention to the terms of the form $u_k^i N_i \frac{\partial N_j}{\partial x_k}$ shows that transposition of the i and j indices will not result in the same terms. This asymmetric matrix required the use of the frontal solver, FRONTA, which uses a total pivoting scheme to solve this particular type of equation set.

3.3 Constraining Streamwise Acceleration

The problem of constraining the streamwise acceleration resulted from a search for an appropriate boundary condition in the case of the curved duct flow. The constraint was necessary for a unique solution. The reasoning was that at a distance far enough downstream of the bend, the effect of the acceleration could be neglected and that a vortical pattern of fluid would be propagated downstream unaltered.

The implementation of this constraint was a simple matter. Since the condition was that two separate dof must have the same value, the technique entailed adding the rows and columns of the slave node to the corresponding master rows and columns. The slave rows and columns were then deleted which essentially reduced $[K]$ from size $[68 \times 68]$ to $[60 \times 60]$.

The procedure began by adding the entries of the columns of each of dof α_5 , α_9 , α_{21} , α_{25} , α_{36} , α_{48} , α_{51} , and α_{60} to dof α_1 , α_{13} , α_{17} , α_{29} , α_{42} , α_{45} , α_{54} , and α_{66} respectively. These relations are depicted in Figure 5. Notice that the dof numbers above correspond to the streamwise dof of interest because of the element topology chosen. The choice of carrying out column operations first was arbitrary and could have just as easily been done after the row additions. The only difference to bear in mind in the case of the row operations is that the right hand side vector must also be modified. More particularly, the slave $[B]$ entries had to be added to the master values at this stage.

Having completed these operations, it is important to note that the matrix has been changed. Left in its present form, the matrix equation no longer characterizes the same

set of equations as at the onset. Carrying out the column operations does not return the original equation set. This is accomplished by the subsequent removal of the rows and columns of the slave dof.

The removal of the redundant slave values was accomplished using the array pointers. Simply, the sixth array entry was reassigned to the fifth, the seventh to the sixth, and so on until the end of the array was encountered. In this way, the slave dof were compacted out of the resulting matrix. The size of the matrix, LVABZ, was decremented accordingly at the end of each compaction.

3.4 Modified Usawa Algorithm For The 15 Node Brick

Equation (3.1) may be rewritten using vector notation to yield the form below. In the case of the 15 node brick, this notation is preferred since most of the literature surrounding this class of elements has been written in this form.

$$\left[(\vec{u} \cdot \vec{\nabla}) u_i \right] - \frac{1}{\rho} \vec{\nabla} \cdot (r_{ij} \vec{e}_j) - F_i = 0 \quad (3.21)$$

The $\left[(\vec{u} \cdot \vec{\nabla}) u_i \right]$ term is expanded in the same way as M. Fortin and A. Fortin [1984b]. In three dimensions, the term could be written explicitly as:

$$\left[(\vec{u} \cdot \vec{\nabla}) u_i \right] = u_1 \frac{\partial u_i}{\partial x_1} + u_2 \frac{\partial u_i}{\partial x_2} + u_3 \frac{\partial u_i}{\partial x_3} \quad (3.22)$$

Clearly, the bracketed term in entirety is the operator and not just the divergence operator represented by $\vec{\nabla}$.

Recall that the stress term, r_{ij} , may be written in terms of pressure and Newtonian stress contributions (in this case, we are considering only the laminar case and the Reynolds stress contribution is omitted). Specifically, the stress is defined by:

$$r_{ij} = -p\delta_{ij} + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (3.23)$$

Proceeding with the typical Galerkin weighting scheme at the element level, equation (3.21) is multiplied by the element shape functions and the integral over the subdomain is sought. The continuity condition, equation (3.2), is also multiplied by a weighting function

and, as with the 20 node brick, this function is not identical to that used for the Navier-Stokes equation. These weightings yield the following:

$$\int_{\Omega} [(\vec{u} \cdot \vec{\nabla}) u_i] v_i d\Omega - \int_{\Omega} \frac{1}{\rho} \vec{\nabla} \cdot (r_{ij} \vec{e}_j) v_i d\Omega - \int_{\Omega} F_i v_i d\Omega = 0 \quad (3.24)$$

$$\int_{\Omega} \vec{\nabla} \cdot \vec{u} q d\Omega = 0 \quad (3.25)$$

Now, integrate the stress integral by parts. We begin by developing the term:

$$\vec{\nabla} \cdot (r_{ij} \vec{e}_j v_i) = \vec{\nabla} \cdot (r_{ij} \vec{e}_j) v_i + r_{ij} \frac{\partial v_i}{\partial x_j} \quad (3.26)$$

Recall Green's theorem:

$$\int_{\Omega} \vec{\nabla} \cdot (r_{ij} \vec{e}_j v_i) d\Omega = \int_S v_i r_{ij} (n_i)_j dS \quad (3.27)$$

where $(n_i)_j$ = direction cosine between outward normal at the centre of j^{th} face and the i^{th} axis

Combining equations (3.26) and (3.27) yields the equation below:

$$\int_S v_i r_{ij} (n_i)_j dS = \int_{\Omega} \vec{\nabla} \cdot (r_{ij} \vec{e}_j) v_i d\Omega + \int_{\Omega} r_{ij} \frac{\partial v_i}{\partial x_j} d\Omega \quad (3.28)$$

Now, combine equations (3.28) and (3.24) to get:

$$\int_{\Omega} [(\vec{u} \cdot \vec{\nabla}) u_i] v_i d\Omega + \frac{1}{\rho} \int_{\Omega} r_{ij} \frac{\partial v_i}{\partial x_j} d\Omega - \frac{1}{\rho} \int_S v_i r_{ij} (n_i)_j dS - \int_{\Omega} F_i v_i d\Omega = 0 \quad (3.29)$$

Using the form of equation (3.29) and the stress tensor equation (equation(3.23)), the following expression may be derived:

$$\frac{1}{\rho} \int_{\Omega} r_{ij} \frac{\partial v_i}{\partial x_j} d\Omega = -\frac{1}{\rho} \int_{\Omega} p \delta_{ij} \frac{\partial v_i}{\partial x_j} d\Omega + \nu \int_{\Omega} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \frac{\partial v_i}{\partial x_j} d\Omega \quad (3.30)$$

Notice that:

$$-p\delta_{ij}\frac{\partial v_i}{\partial x_j} = -p\frac{\partial v_i}{\partial x_i} = -p\vec{\nabla} \cdot \vec{v} \quad (3.31)$$

and that:

$$\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \frac{\partial v_i}{\partial x_j} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) \quad (3.32)$$

Defining the strain rate tensor as:

$$D_{ij}(\vec{u}) = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \quad (3.33)$$

we obtain:

$$\frac{1}{\rho} \int_{\Omega} \tau_{ij} \frac{\partial v_i}{\partial x_j} d\Omega = -\frac{1}{\rho} \int_{\Omega} p \vec{\nabla} \cdot \vec{v} d\Omega + 2\nu \int_{\Omega} D_{ij}(\vec{u}) D_{ij}(\vec{v}) d\Omega \quad (3.34)$$

Thus, the Navier-Stokes equation shown as equation (3.24) may be expanded to:

$$2\nu \int_{\Omega} D_{ij}(\vec{u}) D_{ij}(\vec{v}) d\Omega + \int_{\Omega} [(\vec{u} \cdot \vec{\nabla}) u_i] v_i d\Omega - \int_{\Omega} \frac{p}{\rho} \frac{\partial v_i}{\partial x_i} d\Omega = \int_{\Omega} F_i v_i d\Omega + \frac{1}{\rho} \int_S v_i \tau_{ij}(n_j) dS \quad (3.35)$$

3.4.1 Solution Algorithm

The algorithm proposed to solve equation (3.35) along with the continuity equation (equation (3.25)) is of a form similar to the modified Usawa algorithm developed by M. Fortin and A. Fortin [1984b]. The iterative procedure actually solves for a velocity correction, $\delta \vec{u}$. This correction is required to reduce the residual (obtained when using a trial velocity field in equation (3.35)) to a minimum value, thus obtaining as accurate a solution as possible in the chosen function space. The velocity terms are treated as known quantities since a solution is originally guessed at then updated by the method shown below. The relation between the dof vector and the nodal velocities is made clear in equation (3.37). Based on equation (3.35), the algorithm equation may be shown to be:

$$2\nu \int_{\Omega} D_{ij}(\bar{\delta}\bar{u}) D_{ij}(\bar{v}) d\Omega + \int_{\Omega} [(\bar{u} \cdot \bar{\nabla}) \bar{\delta}\bar{u}] \bar{v} d\Omega + \int_{\Omega} [(\bar{\delta}\bar{u} \cdot \bar{\nabla}) \bar{u}] \bar{v} d\Omega + \\ + r \int_{\Omega} (\bar{\nabla}_h \cdot \bar{\delta}\bar{u}_h) (\bar{\nabla}_h \cdot \bar{v}_h) d\Omega = R(\bar{u}, p) + r \int_{\Omega} (\bar{\nabla}_h \cdot \bar{u}_h) (\bar{\nabla}_h \cdot \bar{v}_h) d\Omega \quad (3.36)$$

Notice that the surface integral in equation (3.35) has been omitted from equation (3.36). The reason is mostly one of convenience in that its treatment would require more terms to be summed into the $[K]$ matrix. Its omission precludes the use of non-zero traction boundary conditions meaning only Dirichlet conditions may be used. This restriction is not a serious one and is, in fact, the only way that boundary conditions were specified with the 20 node brick. The use of symmetry for the analysis of axisymmetric problems, though, is not eliminated since this represents the special case where the surface integral is identically zero.

The velocities and pressures are updated from one iteration to the next by

$$\bar{u}^{n+1} = \bar{u}^n - \delta \bar{u}^n \quad (3.37)$$

$$p^{n+1} = p^n - r \bar{\nabla}_h \cdot \bar{u}_h^{n+1} \quad (3.38)$$

The vector space in which the trial velocity and pressure fields will be sought requires some discussion. The velocity must be chosen in the field of square integrable functions described below:

$$L^2(\Omega) = \{ u \mid \int_{\Omega} |u(x)|^2 dx \} \quad (3.39)$$

The reason for this selection is to allow a Galerkin weighting which allows an inner product described by:

$$(u, v) = \int_{\Omega} uv \, d\Omega \quad (3.40)$$

The continuity equation uses the pressure field for the weighting in much the same way as for the 20 node brick. This weighting function, q , must also be chosen from the $L^2(\Omega)$ space for the same reasons as above.

The terms in equation (3.36) involving the discrete divergence operator should be explained. The elemental divergence is discrete in the sense that it is obtained by an interpolation procedure in much the same way as an interpolated velocity. In this case, the velocity field is said to be discrete as is the elemental divergence. The condition required to satisfy the Babuska-Brezzi condition mentioned earlier was a discrete velocity field satisfying the same divergence criterion on an elemental basis, in an integral sense, as the continuous velocity distribution. More specifically:

$$\int_{\Omega} \vec{\nabla}_h \cdot \vec{v}_h q d\Omega = \int_K \vec{\nabla} \cdot \vec{v} dK \quad (3.41)$$

A piecewise constant pressure field implies a piecewise constant weighting function, q . Although the barycentric node permits a linear pressure field, these non-constant portions of the pressure will be removed from the calculation as shown later in this chapter. Thus the weighting function, q , may be assumed to be a constant. Using unity as the value of that constant:

$$\int_K \vec{\nabla}_h \cdot \vec{v}_h dK = \int_K \vec{\nabla} \cdot \vec{v} dK \quad (3.42)$$

Notice in equation (3.42) that the discrete divergence integral is now evaluated on an element basis rather than on the domain, Ω . This is because of the choice of weighting function, q .

The discrete divergence is a constant on an element and therefore:

$$\vec{\nabla}_h \cdot \vec{v}_h \cdot \text{volume}(K) = \int_K \vec{\nabla} \cdot \vec{v} dK$$

$$\vec{\nabla}_h \cdot \vec{v}_h = \frac{1}{\text{volume}(K)} \int_K \vec{\nabla} \cdot \vec{v} dK \quad (3.43)$$

The $[K]$ and $[B]$ entries required to solve the system of equations can be found by expansion and discretization of equation (3.36). Doing so in the x_1 -direction to demonstrate the method yields:

$$\begin{aligned}
& \frac{\nu}{2} \int_{\Omega} [AA] d\Omega + \int_{\Omega} [BB] d\Omega + \int_{\Omega} [CC] d\Omega + [DD] \\
& = \frac{\nu}{2} \int_{\Omega} [aa] d\Omega + \int_{\Omega} [bb] d\Omega - \int_{\Omega} \frac{p}{\rho} \frac{\partial v_1}{\partial x_1} d\Omega - \int_{\Omega} v_1 F_{x_1} d\Omega + [cc] \quad (3.44)
\end{aligned}$$

where -

$$\begin{aligned}
AA = 4 & \frac{\partial(\delta u_1)}{\partial x_1} \frac{\partial v_1}{\partial x_1} + \frac{\partial(\delta u_1)}{\partial x_2} \frac{\partial v_1}{\partial x_2} + \frac{\partial(\delta u_1)}{\partial x_2} \frac{\partial v_2}{\partial x_1} + \frac{\partial(\delta u_2)}{\partial x_1} \frac{\partial v_1}{\partial x_2} + \frac{\partial(\delta u_2)}{\partial x_1} \frac{\partial v_2}{\partial x_1} \\
& + \frac{\partial(\delta u_1)}{\partial x_3} \frac{\partial v_1}{\partial x_3} + \frac{\partial(\delta u_1)}{\partial x_3} \frac{\partial v_3}{\partial x_1} + \frac{\partial(\delta u_3)}{\partial x_1} \frac{\partial v_1}{\partial x_3} + \frac{\partial(\delta u_3)}{\partial x_1} \frac{\partial v_3}{\partial x_1}
\end{aligned}$$

$$BB = v_1 \left[u_1 \frac{\partial(\delta u_1)}{\partial x_1} + u_2 \frac{\partial(\delta u_1)}{\partial x_2} + u_3 \frac{\partial(\delta u_1)}{\partial x_3} \right]$$

$$CC = v_1 \left[\delta u_1 \frac{\partial u_1}{\partial x_1} + \delta u_2 \frac{\partial u_1}{\partial x_2} + \delta u_3 \frac{\partial u_1}{\partial x_3} \right]$$

$$DD = \text{penalty}$$

$$\begin{aligned}
aa = 4 & \frac{\partial u_1}{\partial x_1} \frac{\partial v_1}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \frac{\partial v_1}{\partial x_2} + \frac{\partial u_1}{\partial x_2} \frac{\partial v_2}{\partial x_1} + \frac{\partial u_2}{\partial x_1} \frac{\partial v_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \frac{\partial v_2}{\partial x_1} \\
& + \frac{\partial u_1}{\partial x_3} \frac{\partial v_1}{\partial x_3} + \frac{\partial u_1}{\partial x_3} \frac{\partial v_3}{\partial x_1} + \frac{\partial u_3}{\partial x_1} \frac{\partial v_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1} \frac{\partial v_3}{\partial x_1}
\end{aligned}$$

$$bb = v_1 \left[u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2} + u_3 \frac{\partial u_1}{\partial x_3} \right]$$

$$cc = \text{penalty}$$

The penalty terms are treated in a special way and will be dealt with separately later in this chapter. The result of this approach is that the values to be used for these terms may be calculated once in some pre-processing code and the $[K]$ matrix may be post-processed to remove the unwanted rows and columns and add the penalty terms.

A diagram of the brick with the chosen element topology is included as Figure 2. The shape functions and their derivatives are tabulated in Tables 3 and 4. As a result of hierarchical interpolation functions, the velocity component in the i^{th} Cartesian direction at any point (ξ, η, ζ) is :

$$u_i(\xi, \eta, \zeta) = \sum_{j=1}^{j=8} N_j(\xi, \eta, \zeta) u_{i,j} + N_9(\xi, \eta, \zeta) \Delta u_{i,9} + \sum_{j=10}^{j=15} N_j(\xi, \eta, \zeta) \Delta u_{i,j}(n_i), \quad (3.45)$$

Now discretizing and grouping terms multiplying the same δu_i terms in equation (3.44) it can be rewritten as shown below. Note that the v_i terms are velocity trial functions in the i^{th} direction used to orthogonalize the residual with respect to the velocity space. The method for actually doing this requires multiplication by the interpolation function of the appropriate node, N_j . Since these functions are the means of mapping a discrete velocity field into a continuous polynomial field, multiplying by these functions is sufficient to orthogonalize the residual with respect to this continuous trial velocity field. Equation (3.44) is rewritten as :

$$\begin{aligned} & \frac{\nu}{2} \int_{\Omega} [(\Lambda\Lambda)(\delta u_1) + (BB)(\delta u_2) + (CC)(\delta u_3)] d\Omega + \\ & + \int_{\Omega} N_i \left(\left[u_1 \frac{\partial N_j}{\partial x_1} + u_2 \frac{\partial N_j}{\partial x_2} + u_3 \frac{\partial N_j}{\partial x_3} \right] \delta u_1 \right) d\Omega + \\ & + \int_{\Omega} N_i \left(\left[N_j \frac{\partial u_1}{\partial x_1} \right] \delta u_1 + \left[N_j \frac{\partial u_1}{\partial x_2} \right] \delta u_2 + \left[N_j \frac{\partial u_1}{\partial x_3} \right] \delta u_3 \right) d\Omega + \\ & + r \left[\left(\frac{\partial N_i}{\partial x_1} \sum_{j=1}^n \frac{\partial N_j}{\partial x_1} \delta u_1 + \frac{\partial N_i}{\partial x_2} \sum_{j=1}^n \frac{\partial N_j}{\partial x_2} \delta u_2 + \frac{\partial N_i}{\partial x_3} \sum_{j=1}^n \frac{\partial N_j}{\partial x_3} \delta u_3 \right) (\text{volume}) \right] \\ & = \text{eqn(3.44)RHS} \end{aligned} \quad (3.46)$$

where -

$$\begin{aligned} \Lambda\Lambda &= 4 \frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_2} \frac{\partial N_j}{\partial x_2} + \frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_2} + \frac{\partial N_i}{\partial x_3} \frac{\partial N_j}{\partial x_3} + \frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_3} \\ BB &= \frac{\partial N_i}{\partial x_2} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} \\ CC &= \frac{\partial N_i}{\partial x_3} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} \end{aligned}$$

So, the $[K_{i,j}]$ submatrix treating the j^{th} node's contribution to the momentum equations at the i^{th} node may be written as:

$$[K_{i,j}] \begin{pmatrix} \delta u_1 \\ \delta u_2 \\ \delta u_3 \end{pmatrix} = [B] \quad (3.47)$$

$$\{K_{ij}\} = \begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} \\ K_{2,1} & K_{2,2} & K_{2,3} \\ K_{3,1} & K_{3,2} & K_{3,3} \end{pmatrix} \quad (3.48)$$

where -

$$K_{1,1} = 4 \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_2} + \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_3} + \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_2} + \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_3} +$$

$$+ N_1 \left[u_1 \frac{\partial N_1}{\partial x_1} + u_2 \frac{\partial N_1}{\partial x_2} + u_3 \frac{\partial N_1}{\partial x_3} \right] + N_1 N_1 \frac{\partial u_1}{\partial x_1}$$

$$K_{1,2} = \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_1} + N_1 N_1 \frac{\partial u_1}{\partial x_2}$$

$$K_{1,3} = \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_1} + N_1 N_1 \frac{\partial u_1}{\partial x_3}$$

$$K_{2,1} = \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_2} + \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_2} + N_1 N_1 \frac{\partial u_2}{\partial x_1}$$

$$K_{2,2} = 4 \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_2} + \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_3} + \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_3} +$$

$$+ N_1 \left[u_1 \frac{\partial N_1}{\partial x_1} + u_2 \frac{\partial N_1}{\partial x_2} + u_3 \frac{\partial N_1}{\partial x_3} \right] + N_1 N_1 \frac{\partial u_2}{\partial x_2}$$

$$K_{2,3} = \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_2} + \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_2} + N_1 N_1 \frac{\partial u_2}{\partial x_3}$$

$$K_{3,1} = \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_3} + \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_3} + N_1 N_1 \frac{\partial u_3}{\partial x_1}$$

$$K_{3,2} = \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_3} + \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_3} + N_1 N_1 \frac{\partial u_3}{\partial x_2}$$

$$K_{3,3} = 4 \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_3} + \frac{\partial N_1}{\partial x_1} \frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_2} \frac{\partial N_1}{\partial x_2} + \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_3} \frac{\partial N_1}{\partial x_2} +$$

$$+ N_1 \left[u_1 \frac{\partial N_1}{\partial x_1} + u_2 \frac{\partial N_1}{\partial x_2} + u_3 \frac{\partial N_1}{\partial x_3} \right] + N_1 N_1 \frac{\partial u_3}{\partial x_3}$$

$$B_1 = 4 \frac{\partial u_1}{\partial x_1} \frac{\partial N_1}{\partial x_1} + \frac{\partial u_1}{\partial x_2} \left(\frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_2} \right) + \frac{\partial u_1}{\partial x_3} \left(\frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_3} \right) +$$

$$+ \frac{\partial u_2}{\partial x_1} \left(\frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_2} \right) + \frac{\partial u_3}{\partial x_1} \left(\frac{\partial N_1}{\partial x_1} + \frac{\partial N_1}{\partial x_3} \right) +$$

$$+ N_1 \left(u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2} + u_3 \frac{\partial u_1}{\partial x_3} \right) - N_1 F_{1,1}$$

$$\begin{aligned}
B_2 = & \frac{\partial u_2}{\partial x_1} \left(\frac{\partial N_i}{\partial x_2} + \frac{\partial N_i}{\partial x_1} \right) + 4 \frac{\partial u_2}{\partial x_2} \frac{\partial N_i}{\partial x_2} + \frac{\partial u_2}{\partial x_3} \left(\frac{\partial N_i}{\partial x_2} + \frac{\partial N_i}{\partial x_3} \right) + \\
& + \frac{\partial u_1}{\partial x_2} \left(\frac{\partial N_i}{\partial x_2} + \frac{\partial N_i}{\partial x_1} \right) + \frac{\partial u_3}{\partial x_2} \left(\frac{\partial N_i}{\partial x_2} + \frac{\partial N_i}{\partial x_3} \right) + \\
& + N_i \left(u_1 \frac{\partial u_2}{\partial x_1} + u_2 \frac{\partial u_2}{\partial x_2} + u_3 \frac{\partial u_2}{\partial x_3} \right) - N_i F_{i,2}
\end{aligned}$$

$$\begin{aligned}
B_3 = & \frac{\partial u_3}{\partial x_1} \left(\frac{\partial N_i}{\partial x_3} + \frac{\partial N_i}{\partial x_1} \right) + \frac{\partial u_3}{\partial x_2} \left(\frac{\partial N_i}{\partial x_3} + \frac{\partial N_i}{\partial x_2} \right) + 4 \frac{\partial u_3}{\partial x_3} \frac{\partial N_i}{\partial x_3} + \\
& + \frac{\partial u_1}{\partial x_3} \left(\frac{\partial N_i}{\partial x_3} + \frac{\partial N_i}{\partial x_1} \right) + \frac{\partial u_2}{\partial x_3} \left(\frac{\partial N_i}{\partial x_3} + \frac{\partial N_i}{\partial x_2} \right) + \\
& - N_i \left(u_1 \frac{\partial u_3}{\partial x_1} + u_2 \frac{\partial u_3}{\partial x_2} + u_3 \frac{\partial u_3}{\partial x_3} \right) - N_i F_{i,3}
\end{aligned}$$

Once again, the penalty terms are not represented in the summations above. These terms are added only after the barycentric dof contributions are processed out of the $[K]$ matrix. Notice that this $[K_{ij}]$ submatrix only applies for the nodes which have three dof. For the remaining six midface nodes, the submatrix must be compacted to the form:

$$(K) (\delta u) \quad (3.49)$$

where $K = K_{ij}$, $(i, j = 1, 3)$ multiplied by the appropriate direction cosines

Here, the three momentum equations have been summed into one row. The individual dof have also been reduced from three columns to one which means that these entries involve the summation of many terms.

The values of the dof for the barycentre are set in order to implicitly achieve the divergence-free condition. Because of this, these dof are not left as typical variables. They are, in fact, eliminated from the element stiffness matrix in a post-processing phase which reduces the size of $[K]$ from $[33 \times 33]$ to $[30 \times 30]$. The method of satisfying this divergence-free condition by the setting of barycentric dof values is as shown below.

Remember from the discussion preceding equation (3.39) that the continuity equation weighting function, q , is the pressure field itself. This pressure is linear as a result of the addition of the barycentric node. Thus, q could be expressed as:

$$q = a_{0k} + a_{1k}(x_1 - \bar{x}_1) + a_{2k}(x_2 - \bar{x}_2) + a_{3k}(x_3 - \bar{x}_3) \quad (3.50)$$

The decomposition of this term multiplying the divergence of the velocity field shows that four conditions must be met. One of these conditions is the achieving of a divergence free field as specified in equation (3.43). The other three conditions are satisfied by the setting of the barycentric node dof to implicitly do so.

Begin by calculating the values of parameters X_i , Y_i , and Z_i which are used to satisfy the conditions specified by equation (3.50). These terms are set according to the following equations:

$$X_i = \int_K (x_1 - \bar{x}_1) \bar{\nabla} \cdot \bar{v}_i dK$$

$$Y_i = \int_K (x_2 - \bar{x}_2) \bar{\nabla} \cdot \bar{v}_i dK \quad (3.51)$$

$$Z_i = \int_K (x_3 - \bar{x}_3) \bar{\nabla} \cdot \bar{v}_i dK$$

where $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$ = element centroid position

The method of solution is the usual Gaussian integration of the element mapped into the unit cube computational space. Then, evaluate the (ξ, η, ζ) Gauss point coordinate and since the centroid position in the mapped element is the origin, the $\bar{x}_1$, $\bar{x}_2$, and $\bar{x}_3$ terms in equation (3.50) may be dropped. The result is then:

$$X_i = \sum_1^{NGAUS} \left[\xi(GP) \left(\frac{\partial(\bar{v}_i)_\xi}{\partial \xi} + \frac{\partial(\bar{v}_i)_\eta}{\partial \eta} + \frac{\partial(\bar{v}_i)_\zeta}{\partial \zeta} \right) (\text{Weight}(GP)) \det J \right] \quad (3.52)$$

Notice here that the shape functions represent true three dimensional vectors. Each of the subscripted ξ , η , and ζ $\bar{v}_i$ functions are the same polynomially but multiply each of the x_1 , x_2 , and x_3 nodal dof. Clearly:

$$\bar{v}_1 = (N_1, 0, 0)$$

$$\bar{v}_2 = (0, N_1, 0)$$

$$\bar{v}_3 = (0, 0, N_1)$$

and so on for each of the nodes. Thus, there are 45 $\bar{v}_i$ functions for a 15 node brick corresponding to the 45 X_i , Y_i , and Z_i values. Then, the barycentric dof which have been removed from the global equation may be retrieved at the element level using the formulae:

$$\begin{aligned}\alpha_{25} &= -\frac{1}{X_{25}} \left[\sum_{i=1}^{24} \alpha_i X_i + \sum_{i=28}^{45} \alpha_i X_i \right] \\ \alpha_{26} &= -\frac{1}{X_{25}} \left[\sum_{i=1}^{24} \alpha_i Y_i + \sum_{i=28}^{45} \alpha_i Y_i \right] \\ \alpha_{27} &= -\frac{1}{X_{25}} \left[\sum_{i=1}^{24} \alpha_i Z_i + \sum_{i=28}^{45} \alpha_i Z_i \right]\end{aligned}\quad (3.53)$$

$$\begin{aligned}\text{where } -\alpha_{25} &= (u_1)_0 \\ -\alpha_{26} &= (u_2)_0 \\ -\alpha_{27} &= (u_3)_0\end{aligned}$$

The final point which requires clarification is the method used to treat the penalty terms. These penalty terms are evaluated using the discrete divergence integrals and the way the divergence free condition which must be satisfied as a result of the pressure field chosen. Recall equation (3.43) which describes the discrete divergence integral rewritten below for a specific dof i :

$$\bar{\nabla}_h \cdot \bar{v}_{ih} = \frac{1}{\text{volume}(K)} \int_K \bar{\nabla} \cdot \bar{v}_i dK \quad (3.54)$$

The left hand side is the term which has been referred to as the discrete divergence interpolation function. One such term is calculated for each dof and is then used in the same way as the familiar N_i functions. To calculate the element discrete divergence, sum the products of the dof and the interpolation functions:

$$\bar{\nabla}_h \cdot \bar{u}_h = \sum_{i=1}^{30} \alpha_i \cdot (\bar{\nabla}_h \cdot \bar{v}_{ih}) \quad (3.55)$$

This is the term which is used in the convergence test and for the updating of the constant portion of the elemental pressure.

With regards to the summation of the penalty terms into the $[K_{ij}]$ entry, the term may be simplified to using these interpolation functions. Remember from equation (3.36) that the two penalty terms which must be modeled are:

$$\begin{aligned} & r \int_{\Omega} (\vec{\nabla}_h \cdot \delta \vec{u}_h) (\vec{\nabla}_h \cdot \vec{v}_h) d\Omega \\ & r \int_{\Omega} (\vec{\nabla}_h \cdot \vec{u}_h) (\vec{\nabla}_h \cdot \vec{v}_h) d\Omega \end{aligned} \quad (3.56)$$

Realizing that the discrete divergence, $(\vec{\nabla}_h \cdot \vec{u}_h)$, is a constant on an element and using equation (3.54), equation (3.56) is rewritten as:

$$\begin{aligned} r \int_{\Omega} (\vec{\nabla}_h \cdot \delta \vec{u}_h) (\vec{\nabla}_h \cdot \vec{v}_h) d\Omega &= r (\vec{\nabla}_h \cdot \delta \vec{u}_{jh}) (\vec{\nabla}_h \cdot \vec{v}_{ih}) \cdot \text{volume}(K) \\ r \int_{\Omega} (\vec{\nabla}_h \cdot \vec{u}_h) (\vec{\nabla}_h \cdot \vec{v}_h) d\Omega &= r (\vec{\nabla}_h \cdot \vec{u}_h) (\vec{\nabla}_h \cdot \vec{v}_{ih}) \cdot \text{volume}(K) \end{aligned} \quad (3.57)$$

The notation may be simplified by realizing that the discrete divergence interpolation functions are functions of geometry only and may thus be calculated once in a pre-processing phase and stored. Let:

$$V_i = (\vec{\nabla}_h \cdot \vec{v}_{ih}) = \frac{1}{\text{volume}(K)} \int_K \vec{\nabla} \cdot \vec{v}_i dK \quad (3.58)$$

Using this notation shows that the entries required in the $[B_i]$ and $[K_{ij}]$ positions are:

$$\begin{aligned} \text{penalty}[B_i] &= V_i \cdot (\vec{\nabla}_h \cdot \vec{u}_h) \cdot \text{volume}(K) \\ \text{penalty}[K_{ij}] &= V_i \cdot V_j \cdot \text{volume}(K) \end{aligned} \quad (3.59)$$

The procedure to be used for the summation is now clear. Begin by calculating the vector of 30 V_i values at the start of the problem. Formulate the $[K]$ matrix and $[B]$ vector in the way described above. At the end of this procedure, post-process the matrix and vector to remove the barycentric contributions and finally, add the penalty terms into the matrices based on equations (3.58) and (3.59).

Chapter 4

DESCRIPTION OF FORTRAN CODE

4.1 20 Node Brick

4.1.1 VFL3D20

This element stiffness matrix and load vector generator was converted from the program written by Vernaganti [1984]. The nomenclature was adopted from that source as was the logic. The basis for the approach is well documented in Taylor & Hughes [1981]. A copy of the source code for this routine has been included as an appendix.

The routine begins by interpreting the data input by the user which had been stored in the array VPROP. The meanings of the ASGARD variables are easily found either from the appendix included in this report or from the dictionary of variables in the comments at the start of the code itself.

For the first iteration, INIT is called to initialize the COORD array to zero entries except for the nodes with prescribed value. MIDSIDH is then called in order to alter the absolute values of the element geometry to the hierarchical values required and discussed at length in various sections of this report. The element stiffness matrix, $[K]$, and the right hand side vector are then set to zero. Having prepared the various storage areas, the processing begins.

The Gauss point locations and weights are determined in the routine GAUS3D and stored in the vectors XGAUS, YGAUS, and ZGAUS and WEIGP. A local vector storing the appropriate Gauss point coordinates, XITA, is set from the Gauss point coordinate vectors and the shape function and shape function derivative values are evaluated at each of these points. The velocities at the Gauss point are then calculated using the nodal values arrived at in the previous iteration. In the case of a turbulent calculation, the eddy viscosity and the viscosity gradient values are also calculated. The right hand side vector, consisting only of body force terms, is then determined.

The various $[K]$ matrix entries are calculated by setting pointers to various positions in the matrix. The IROWU, IROWV, IROWW, JCOLU, JCOLV, and JCOLW variables are these pointers. They serve to indicate into which row and column the various terms should be summed. Initially, the velocity terms alone are considered in the $[K]$ construction. Thus, each node has three rows and columns into which to sum terms. These pointers are incremented to consider, in turn, each of the three columns associated with a nodal submatrix as described in Chapter 3. Once all the columns for a set of three rows have been completed, the next three rows are used and so on until the end of the matrix. In a similar fashion, the pressure rows are summed near the end of the routine. In this case, only eight of the sixty eight rows need be considered.

In the case of a turbulent flow, the supplemental Reynolds stress values are then calculated. These stresses are added into the off-diagonal terms where they have been shown to belong. The summation for the velocities ends and the pressure summation begins. It proceeds in a manner analogous to the velocity summation.

Near the end of the routine, NSPL is checked to determine whether any special requirements are demanded of the element. If NSPL is one, the element is a no x-acceleration element and the stiffness matrix must be rearranged accordingly (see Sections 3.3 and 4.2). In either case, the element stiffness matrix along with the right hand side vector is written to a disk scratch file for the subsequent read by the frontal solver's reading routine, ABFIND. The routine then returns to ELFILE.

4.2 NOXAXL

The method used to derive the modified element stiffness matrix for the zero streamwise acceleration element (NOXAXL) is described in Section 3.3. Once the composition of the new matrix is identified, the implementation of the modification is clear.

The elemental slaved dof numbers are α_5 , α_9 , α_{21} , α_{25} , α_{36} , α_{48} , α_{51} , and α_{60} which correspond to the x_1 -wise velocity components of each of the downstream nodes. Since these dof are to be set equal to the upstream master dof, they are compacted out of the matrix and their row and column entries are added to the appropriate rows as described earlier. See Figure 5 for a schematic of the relationship between these dof and their masters.

At the start of the routine, the columns of the slaved nodes are added to the corre-

sponding master columns. It is important to realize that at the end of this operation, the matrix is no longer equivalent to what it was originally. It is easily verified that carrying out the analogous row operations will complete the summation processes required prior to deleting the slave dof.

Having completed the summation processes, the redundant rows and columns are erased. This is accomplished by decrementing the array index of each of the rows and columns of those dof subsequent to the dof in question. Thus, to compact out the fifth row of the matrix, the sixth row is reassigned the fifth, the seventh the sixth, and so on to the end of the matrix. The same operations are carried out for the columns. At the end of the compaction of each of the slave dof, the number of elemental dof, LVABZ, is decremented by one to account for the progressively smaller matrix.

4.3 15 Node Brick

4.3.1 VFL3D14

In a manner similar to VFL3D20, VFL3D14 begins by reading the user input stored in the VPROP array. The first iteration calls to INIT and MIDSIDH remain although the routine MIDSIDH is obviously altered. The difference in this initial phase is the call to a new routine called GEOM. GEOM is used to calculate some quantities related to the geometry of the problem such as the direction cosines at the faces of the element in question. It also determines the components of the vector normal to the faces bearing in mind the necessity for a homogeneous inter-element definition of this quantity. Finally, GEOM calculates the values of the X, Y, and Z vectors used to eliminate the internal node contributions to the stiffness matrix. The discrete divergence interpolation values are calculated and stored since they are functions of geometry only and are independent of iterated data.

Upon return from GEOM the modified coordinates, the X, Y, and Z vectors, as well as the discrete divergence interpolation values are written to a scratch file for recovery on successive iterations. Once the $[K]$ matrix and right hand side vector are initialized, the barycentric node dof are recovered. These values are required for the proper summation into the $[K]$ matrix. The elemental discrete divergence is then calculated from the discrete

contributions from which the elemental pressure may be determined using the penalty factor. The $[K]$ matrix summation is now possible.

The loop over the Gauss points proceeds in the same manner as the 20 node brick and the number of Gauss points is also the same (which is apparent since the number of points is based upon the order of the element and the orders of the two elements are the same). The most noticeable difference in the early stages is the presence of the array GRAD. This array stores the gradients of the three Cartesian velocity components in each of the three directions based on the previous iteration's nodal values. The calculation of the velocities themselves proceeds in the same way as the 20 node brick bearing in mind the different summation required as a result of the different hierarchical schemes used.

The remainder of the code results from the equations derived in Chapter 3 and needs little explanation. The only difficulty is in identifying which submatrix is being treated and this is accomplished by the pointers defined earlier in this chapter.

At the end of the first loop summing entries into the $[K]$ matrix, the post-processing procedure used to remove the rows and columns of the barycentric node is carried out. As mentioned earlier, this removal means that the addition of the extra node results in only a minor increase in computational cost while returning a substantially more robust element. Finally, the penalty terms are added to the $[K]$ matrix and the $[B]$ vector. These arrays are then written to the scratch file for the subsequent read by the frontal solver, FRONTA.

4.4 Frontal Solver

The frontal solver used, FRONTA, was originally developed by Hood [1978] and was modified by Vernaganti [1984] to work with ASGARD test cases. The program was designed to provide a practical means of solving an asymmetric equation set. Vernaganti's version used fixed array dimensions and, for some cases, used values characteristic of his test cases only to signal various procedures. It had not been optimized for minimum solution time. Part of the current research was to improve the program's generality in order to solve arbitrary problems and to optimize its performance. Since the solution procedure accounts for approximately 70% to 90% of the total run time, it is obvious that this is one of the most appropriate phases to improve.

Following the transfer of the existing ASGARD code from the CP-6 system to the

APOLLO network, the author's immediate task was to achieve the same level of development which had previously existed. Having reached this level, the author began modifying the asymmetric frontal solver to improve its performance. A graph of solution times versus time of development is shown as Figure 3. The independent variable was determined by samples of run times at various stages of development. The reduction in run time which resulted from this optimization was of the order of about 70% for available test cases.

4.4.1 FRONTA

The solver is a Gaussian elimination, back-substitution routine using a total pivoting scheme to make it more robust. The pivoting scheme represents a compromise between the ideal pivoting scenario and the storage limitations of the computer. More specifically, while it is desirable to store the entire matrix in core at one time and to search this matrix for subsequent pivots, the finite amount of available memory makes this impossible for all but the smallest cases. This solver places a restriction on the size of the portion of the asymmetric matrix kept in core at any time and seeks a pivot in this area.

Having chosen this approach for pivot choice, the solution algorithm becomes clear. Quoting from the original source (Hood [1976]):

"The frontal routine starts by assembling each of the element stiffness matrices in turn into core, until the core area allocated to the solution routine by the programmer is filled. Then, from within the assembled part of the complete matrix, a pivotal search is made to determine the largest entry amongst the rows and columns which are fully summed, i.e. rows and columns to which no further contributions will arise in subsequent assembly of element stiffness matrices. The pivotal row is then used to eliminate all the coefficients in the pivotal column, after which it is placed on the backing store disc. When sufficient coefficients have been eliminated, it is possible to assemble the next element stiffness matrix after which further elimination may take place. When finally all the coefficients have been eliminated the solution is obtained by a back-substitution routine."

Turning now to the frontal solver code, the detailed procedure may be described. A series of flags and counters are initialized at the start of the program including the parameter for the maximum size of the area allowed in core (NCRIT). The pre-front procedures which

provide the information regarding the number of dof for each node in the front follow, as well as whether they have a prescribed value or not. The array used to store the pointer to the first dof of each node in the front is also created.

Having initialized the pointers and counters, the first element stiffness matrix is brought into core by calling subroutine ABFIND. This routine reads the disc file containing the element stiffness matrices and elemental right hand side vectors. This information was originally written at the end of the particular elemental routine (VFL3D20 for the 20 node brick, VFL3D14 for the 15 node brick). Since the global topology need not number nodes in a sequential order, a pointer detailing which nodes are in the current front is required. This pointer, NK, is constructed according to one of the three cases below:

- i) each node has one dof,
- ii) nodes have multiple dof and have no special solution requirements, or
- iii) nodes have multiple dof and some dof are slaved to take on the value of some master node (used to impose the zero streamwise acceleration condition for the 20 node brick).

An important factor in the routine's success lies in its ability to map the front's equation numbers into the global dof and vice versa. These relations are possible through the use of two pointers referred to as the heading and destination vectors. For a detailed description of heading and destination vectors, see Irons & Ahmad [1984]. Briefly, though, the heading vector takes an element equation number and translates it into a global dof number. The destination vector takes a global dof number and returns an element equation number. These two vectors are constructed from the NK array and are then used to direct the element stiffness entries into the correct locations in the global matrix.

Elimination takes place only on the fully summed nodes. This fact places an obvious restriction on the minimum area which can be allotted and the program halts if none of the nodes may be eliminated or when the allocated area is full. The search for fully summed entries produces a list of node numbers which may be removed.

The method used to treat prescribed boundary conditions may vary from one solver to the next. Some programs may multiply the diagonal entry by a very large number to ensure it represents the major contribution to the row. This routine uses a somewhat different

technique which essentially achieves the same result. FRONTA identifies a prescribed value row in the matrix and makes the diagonal entry unity. All other row entries are set to zero and the right hand side entry is given the prescribed value. The end results of the two techniques are clearly similar and are identical if the multiplying factor of the former technique is chosen to be large enough.

The search for the pivot follows. The only entries considered are the fully summed nodes since the pivotal equation will be written to disc following a normalizing operation. The largest entry is the pivot and may be found in any row or column satisfying the fully summed restriction. The pivot row and column are then eliminated by an appropriate Gaussian matrix operation and the normalized pivotal row is written to a disc file for future use by the back-substitution routine.

Since a row has then been removed from the stiffness matrix, the heading and destination vectors must be altered to account for this new numbering scheme. Having done so, the routine checks whether to continue eliminating, begin further assembling, or begin the back-substitution phase. The continuation of the elimination is straight forward and is exactly as described above. If more assembling is required, FRONTA returns to the original call to ABFIND and sums the next element stiffness matrix into the global matrix using the same procedures as described above.

Once the back-substitution procedure is called, the calculation of the solution dof vector, SK, is begun. Because the method used to account for boundary conditions leaves a unity diagonal entry bounded by zero entries on the row, SK need only be given the right hand vector entry of rows with prescribed value. The other values are found by stepping backwards through the pivotal row equations as returned from the scratch file.

Since this a non-linear solver, a convergence check is required. FRONTA calls this checking routine, TOLREL, and passes it the newly calculated dof vector as well as the dof vector from the previous iteration. TOLREL returns a flag denoting either a converged or an unconverged solution. FRONTA returns this flag to ASGARD

4.4.2 TOLREL

The convergence check of a solver is essentially arbitrary. Neither the metric used to establish success, or the tolerance specified for a particular problem is strictly constrained. Because of the different nature of the dof vectors produced by the two elements, different

techniques were used to test whether the solutions were acceptable or not

In the case of the 20 node brick, each of the hierarchical dof is checked with its counterpart from the previous iteration. The percentage change formula below is used to calculate $\% \Delta$ for each dof. If all of these values are less than the input tolerance, the solution is accepted. If this is the case the convergence flag, NCONV, is set to unity and returned to FRONTA for subsequent verification by ASGARD. This value ends the run and causes output of the converged solution. If this is not the case, each dof is altered by use of an over-relaxation parameter which has been shown to improve convergence rates. The convergence check and relaxation scheme are shown in the formulae below

$$\% \Delta = \frac{\text{varb2} - \text{varb1}}{\text{varb2}} \quad (4.1)$$

$$\text{varb2} = \text{varb2} + \theta (\text{varb1} - \text{varb2}) \quad (4.2)$$

where

- varb1 = dof vector from previous iteration
- varb2 = current, then updated dof vector

The convergence check used for the fifteen node brick is vastly different. In this case, the metric used is the elemental discrete divergence. A RMS (root mean square) value for the total discrete divergence is obtained by summing the individual elemental values. The elemental discrete divergences stored in the vector DIVERGE are squared for summation in TOLREL and the positive square root of the total, CANLA, is calculated in order to avoid the effects of sign in the individual values. Then, this RMS total is compared to the tolerance input by the user. If the RMS value calculated is less than the tolerance, the convergence flag, NCONV, is set to unity. If not, the flag is left as zero and the program continues to iterate until the maximum number of iterations is reached or the convergence test returns NCONV equal to unity. A typical value to use for the tolerance, TOLERA, would be approximately 0.001.

If the convergence check of the fifteen node brick shows that further iterations are required, the nodal velocities are updated from the dof corrections before exiting the routine. The update of the pressures is done in VFL3D14 as part of the preparatory phase.

Paying specific attention to the routine TOLREL, the reader may see that the 20

node brick tolerance checking has been bypassed to allow easy modification of the code for any future elements using the routine. Notwithstanding the element used, the converged solution is modified just prior to output to convert from hierarchical to absolute velocities. This facilitates examination and graphical representation of the velocity field. Thus, in the case of the 20 node brick, the converged dof value is added to the average of the two bounding corner node dof values. Similarly, the midface nodes of the fifteen node brick are added to the average of the bounding four corner nodes. The barycentric node is also appropriately modified.

4.5 Post-Processor

4.5.1 SEPARATE

The program is based upon the fact that ASGARD produces output files in a consistent sequence. Since the same information is output from one run to the next with the only difference being the values of the numeric output, some key lines may be sought and used as benchmarks to read the required information. Once all the interesting data has been read, the user may choose to suppress some of the redundant information from the compressed output.

The program begins by calling a routine called `READER` which, as the name implies, is responsible for the reading of the ASGARD output file. To do this, it begins by searching for the line which gives the time of the run. This implies reading over the `TITLEN` lines until the string read corresponds to the format of the time output by ASGARD. Some global information is then read from the appropriate lines of the file. In a similar way, all the necessary data is gleaned from the correct areas of the file.

Having read all this data, `SEPARATE` prompts the user to return the type of output desired. Since different formats could be used to highlight various trends in the iterative procedure, two types of output have been provided for:

- i) Single tables for each of the four dof. Each table is made up of individual nodes, one for each row, with progressive iteration numbers forming the columns.
- ii) Each node given a separate table of information with the four dof forming the columns of the table and the iteration number the rows.

Since the dof fixed through the use of boundary conditions show no change, their output is redundant and distracting. The user may remove all prescribed value dof from the final output. The user may also wish to track only specific dof such as those on the exit face of a complex geometry. Such a demand is accommodated through response to a question posed by SEPARATE.

The method used to accomplish the removals described above is quite simple. Since the dof have been read into an array with the corresponding global node number, a comparison with this index allows the routine REMOVE to condense out the undesired entries before sending the array to be output. The record to be removed is identified either by using the list of nodes input or by comparing with the nodes assigned prescribed values (as read in the routine READER). Once an undesirable record is identified, subsequent record numbers are decremented by one which essentially squeezes out the record in question. These record operations are very simple for formatted files such as those read by READER.

Finally, the dof values are output in the desired format and the user is asked to either end the run or to continue with another form of analysis.

Notice that the format for each type of ASGARD file is the same but the number of parameters read and the elemental topology used are specific to the 20 node brick. For this reason, SEPARATE will not work with the output files generated for the fifteen node brick. Some modifications will be required for adaptation to this element.

Chapter 5

TEST CASES AND RESULTS

5.1 20 Node Brick

The test cases for the 20 node brick were the more comprehensive of the two elements used. Since the skeleton procedure for the formulation had already been written, the purpose was originally to improve its range of applicability. As a result of complications which arose, a full range of test cases was required which finally demonstrated the 20 node brick's inconsistent behaviour in predicting pressure fields.

The Poiseuille and Couette flow tests carried out showed that given a pressure gradient across the problem geometry, the element was capable of returning a velocity profile which was in very good agreement with the exact solution. With this additional confidence, the problem constraints were changed in order to enlarge the range of cases for which solutions could be determined. The initial test cases were again used but this time upstream velocity profiles and pressures were specified and the downstream quantities were sought. Here as well, the element consistently returned near exact solutions for both the downstream velocity distribution and the pressure.

A series of pipe flow tests were also carried out. In order to achieve a typical Re_A value of 2000, a pressure gradient was applied across the element. With the geometry given, a unique solution existed and the result computed was very near the exact solution. Various combinations of boundary conditions were tested in ways similar to the Poiseuille flow tests described above. The pipe flow case was also used to test the NOXAXL routine which constrained the downstream streamwise velocity to be equal to its upstream node partner. This proved to be very efficient at reducing the solution runtime and returning considerably better velocity profiles.

Notice that each of the three test geometries mentioned above yielded convection free fields. It was therefore decided to incorporate the convection terms into the problem by including a total diffusion angle of approximately 6 degrees into the previous Poiseuille

flow problem. This modification essentially transformed the original case into a two dimensional diffuser with a moderate diffusion angle. The results obtained showed the spurious pressures.

The next test case was that of a square duct bent through an angle of approximately 30 degrees giving rise to a pair of secondary flow vortices at the downstream face. A finite difference (FD) solution satisfying only the continuity equation provided an approximate downstream station velocity distribution. It was found that the element converged to a solution with a velocity distribution very near that predicted by the FD scheme. The small deviations were attributed to the inability of the FD model to satisfy the momentum balance through the duct. The point of interest, however, was in the pressure field computed. The pressures were of order 10^{-1} to 10^4 greater than the input upstream pressure. It became apparent after several attempts to correct this anomaly that the element was not functioning properly.

Each of the test cases mentioned above is described in more detail in the following sections.

5.1.1 Plane Poiseuille Flow

This test case was the simplest of all those used. The plane-Poiseuille flow is one of the problems for which an exact solution to the Navier-Stokes equations exists. The pressure gradient exactly balances the wall shear stress and, after applying the no-slip condition at the solid boundaries, produces a parabolic velocity profile defined as:

$$u_1(x_2) = -\frac{1}{8\mu} \frac{\partial p}{\partial x_1} (h^2 - x_2^2) \quad (5.1)$$

The symbols h and x_2 are shown in the diagram of the Poiseuille flow included as Figure 4. Since the order of the solution to the real velocity field is the same as the finite element interpolation order, one element should be sufficient to obtain the exact solution. This was indeed the case as the FEM velocity and pressure solutions agreed with the exact solution to within approximately $10^{-4}\%$. This value as well as all other percentage differences in this chapter were calculated using one of the formulae:

$$\% \text{difference} = \frac{\text{theory} - \text{FEM}}{\text{theory}}$$

$$\% \text{difference} = \frac{\text{experiment} - \text{FEM}}{\text{experiment}}$$

Since convergence must be checked, the solution required two iterations to converge despite the fact that the exact solution had been obtained after just one iteration. The rapidity of convergence is a result of the fact that the velocity field is convection free which means that the source of non-linearity is not present. Thus, the immediate step to the exact solution comes as no surprise.

The test case's exact agreement and its two iteration convergence were documented with each of the various combinations of boundary conditions described above. The comparison of the FEM obtained velocity field with the exact solution is shown in Figure 11.

5.1.2 Couette Flow

The Couette flow is very similar to the Poiseuille flow except one plate has a velocity relative to the plane opposite. Thus, the Poiseuille flow can be considered a special case of the Couette flow where the velocity of the top plate is zero relative to the bottom. The equation representing Couette flow is easily shown to be:

$$u_1(x_2) = \frac{2U}{h} (x_2 + h) - \frac{1}{8\mu} \frac{\partial p}{\partial x_1} (h^2 - x_2^2) \quad (5.2)$$

The agreement with the exact solution was the same as for the Poiseuille flow and requires no further discussion.

5.1.3 Two Dimensional Diffusing Section

This was the simplest test case for which the spurious pressure modes were seen. The test converged after twelve iterations to a an exit plane velocity profile with the correct shape and magnitude. The inlet and exit velocity distributions are shown in Figure 12. The upstream pressure had been assigned a value of two units and the upstream velocities were fixed. The corresponding downstream pressures and velocities were sought. The velocities and pressures applied had been chosen to once again arrive at a Re_h value of approximately 2000.

The surprising aspect of the solution arrived at was the order of magnitude of the downstream pressures. These pressures were of order 10^4 which was obviously incorrect. It was noticed, however, that the incorrect pressures followed a definite pattern and were not

arbitrarily distributed. In fact, each of the downstream pressures recorded at the section boundaries was the same and the centreline pressures were equal to each other.

This symmetry in the pressure field was expected and merely the magnitude of the terms was in error. This is one of the distinct signs of spurious pressures. The symmetric pattern has been postulated to be the result of satisfying a different eigenvalue than the correct one and the pattern of pressures is then, clearly, the corresponding eigenvector. One method of removing spurious pressures has been to filter out this spurious eigenvalue thus retaining only the correct solution. This technique has been used predominantly with regular meshes and was not considered a good method to try and resolve the spurious pressures of the 20 node brick.

5.1.4 Fully Developed Laminar Pipe Flow

This test case was eventually chosen as a benchmark to demonstrate improvements in the frontal solver. It consisted of three elements, thirty nine nodes and was a non-trivial problem. The prediction of pipe flow pressures and velocities is a useful ability but still allowed easy verification of the solution. This test case was also used to test the code for the no x_1 -acceleration element since a constant pipe diameter must produce a constant streamwise velocity.

The streamwise velocity at any radial station in the pipe (making use of polar coordinates) may be shown to be:

$$u_3(r) = - \left(\frac{1}{4\mu} \right) \frac{\partial p}{\partial x_3} (R^2 - r^2) \quad (5.3)$$

Making use of the axisymmetric nature of the problem, x_2 may be used to represent the radial distance and the problem may be rewritten in Cartesian axes as:

$$u_1(x_2) = - \left(\frac{1}{4\mu} \right) \frac{\partial p}{\partial x_1} (R^2 - x_2^2) \quad (5.4)$$

Here, x_1 is the streamwise direction.

Since this is a convection free field it was expected that, as in the case of Poiseuille and Couette flows, convergence could be expected within two iterations to an accuracy of approximately $10^{-4}\%$. This was not the case. In fact, convergence required four iterations

and then the results were incorrect by between -13% and -22% . This was most perplexing considering the similar nature of the Poiseuille flow and its success.

The no x_1 -acceleration element was then developed. The results arrived at using this element were much closer to those expected. The maximum error after convergence in two iterations was of the order of 1% . This was much more understandable and returned focus on the reason why the pipe flow case not using the no x_1 -acceleration elements showed such poor behaviour.

After a great deal of dissection of the solution process, it was found that the problem had, in fact, been numerical roundoff of the terms in the stiffness matrix. The order of magnitude of each of the convection terms was approximately 10^7 larger than all other terms being summed. This was true since the other terms were multiplied by the viscosity ($O(10^{-7})$). Although these convection terms did sum to zero on any given row, they were overshadowing the truly important effects from the other terms. This, in essence, is the equivalent of the depiction of numerical accuracy where a large number such as 10^{10} is added to unity and then subtracted again to show the result is not unity. In both the test case and the example above, the significant contribution had been masked by the larger, though unimportant, terms. Clearly, in an unrealistic environment where infinite precision could be used rather than the finite limitation of the computer, this would not be a problem.

5.1.5 Turbulent Pipe Flow

A turbulent flow calculation was carried out using three elements in the same element configuration as the laminar pipe flow test. The Reynolds number was increased to 10^5 and the source of information for eddy viscosity values at radial positions as well as the empirical velocity values was Laufer's pipe flow data as reported in Hinze [1959].

The values returned by the program showed the fuller velocity profile characteristic of the increased momentum transfer in a turbulent flow. The values compared well with Laufer's data and showed a maximum error of approximately 16% . The comparison was difficult because of the node placement and should not be treated as a detailed analysis. Since velocities near the wall are those which experience the greatest gradients, it is obvious that this is the area of primary concern. Because of the mesh used, however, this area was not exhaustively modelled and the results should be treated as suspect.

This observation shows one of the primary disadvantages of the elements used. The increase in computation time which results from successive mesh refinements is staggering considering that the elements must have all nodes in common and that badly distorted elements tend to give unreliable answers and should, therefore, be avoided. Thus, to double the number of elements close to the wall might require adding up to nine more elements to a problem which began with only three. The associated increase in solution time becomes obvious.

One method presently being used in the FEM analysis of solid mechanics is graded elements. A graded element may possess a variable number of nodes thereby eliminating the requirement for all nodes to be shared by two or more adjacent elements. Rather than requiring nine elements for the mesh refinement discussed above, perhaps only four would be required since the refinement could be made in the region near the wall only without propagating to the area where velocities experience little change. These elements are a reality and could serve in future to allow the mesh accuracy required for detailed analysis of areas with steep velocity gradients.

5.1.6 Three Dimensional Secondary Flow From Curved Duct

This test case was initially created to demonstrate the three dimensional capabilities of the 20 node brick. Upstream pressures of one unit were applied in order to drive the flow through the duct. The upstream fixed velocities represented a bi-parabolic velocity distribution that one could generate by superposition of the two parabolic velocity profiles in the x_2 and x_3 directions (the x_1 direction is the streamwise direction and the x_3 direction is vertical). Figure 8 may be consulted to view the test geometry. The other interesting aspect of the boundary conditions was determining a suitable condition for the downstream end. Fixing an upstream pressure and velocity distribution would not have been sufficient for a unique solution in a finite number of iterations owing to the highly non-linear nature of the problem. It was, therefore, necessary to further define the problem by fixing some downstream condition. It was felt that a condition which essentially determined that the effect of the curvature was no longer felt by the flow would ensure the effects of the bend were examined yet still allowed a reasonable solution time for the problem. Thus, the NOXAXL element was developed to meet these demands. The four elements comprising the cut plane at the downstream end are all NOXAXL elements. From a physical standpoint, the flow

advances towards the bend and secondary vortices begin to be generated as a streamwise component of vorticity is developed. Past the bend, the vortices continue to flow but, since the duct is a fixed size, the streamwise velocity becomes approximately constant which is modelled by the NOXAXL elements.

The velocities at the outlet were verified using a finite difference solution of the same problem which satisfied the continuity equation only. A pattern of contra-rotating secondary vortices was observed at the outlet. This metric for comparison would have been insufficient for detailed comparison of a final solution since it was not experimentally verified and did not consider the effects of pressure changes on the flow. This additional information could have been obtained only if the momentum equations through the duct were also solved. It was sufficient, however, to establish a pattern to be searched for in the FEM solution.

The magnitudes of the pressures through the element varied from 1.0 at the inlet down to a minimum of -4.06×10^6 up to a maximum of 2.03×10^6 . The trend along any line of element edges from the inlet to the exit is highly oscillatory yet is exactly equal along corresponding lines. The pressure variations along these continuous lines of corner nodes are depicted in Figure 18. Because of the symmetry of the pressure solution, only six of the paths were required to describe all the possibilities. In the graph, each station is meant to represent the boundary between adjacent lines of elements where one group of four elements is upstream of the next four.

The pressures once again display a highly symmetric pattern and seem to satisfy some of the qualitative aspects of the expected pressure field. A pressure gradient across a station may also be observed and was expected for a problem of this type. Once again, the only difficulty lies in the excessively large magnitudes of the pressures. These spurious pressures were the cause for stepping back to a simpler problem and trying to establish a reason for the apparent failure of the element. The testing carried out with the diverging section eventually led to the abandoning of this element in favour of the development of the 15 node brick.

5.2 15 Node Brick

The test cases for the 15 node brick were very rudimentary. They were meant only

to test the code for flaws. Since the element was based on several new principles such as the unique midface dof and the elimination of the barycentric node, its development will certainly be a lengthy one. Also, the non-constant part of the elemental pressure was not retrieved so pressure values could only be observed at the element barycentres. This retrieval should form part of the future work as detailed in Chapter 6.

5.2.1 One Element

Having developed the code, the first test was a one element test with a velocity set at one midface and all other nodes, except the opposite downstream node, fixed to zero velocity. Figure 9 shows the geometry and boundary conditions for this problem. Obviously, the result should be a downstream velocity equal to the upstream input. The result was obtained in two iterations. Once again, the two iterations are required because of the convergence check. The check for this element involves the RMS value of elemental discrete divergences. Since the one element begins with the upstream node fixed at some non-zero value and the downstream equivalent fixed at zero, the discrete divergence is non-zero. Once the downstream velocity is provided with the correct velocity by means of the penalization of the pressure, the discrete divergence reduces to within the user input tolerance.

5.2.2 Two Elements In The Streamwise Direction

The next test was an attempt to see the changes in pressure from one element to the next. The first test had given an absolute pressure at the barycentre of the lone element. The two element test case would return two pressures which could be examined qualitatively. This test is premature considering the incomplete nature of the pressure formulation and cannot be expected to provide dependable results.

The problem consisted of two elements aligned in the streamwise direction to allow the flow out of the upstream element to enter the downstream equivalent. The upstream velocity was fixed to a non-zero value and the boundary nodes were fixed to prevent fluid efflux. A sketch of the geometry along with the imposed boundary conditions has been included as Figure 10. The test converged in two iterations as might be expected since it is a linear problem. The velocities obtained were exactly as expected and the magnitudes of the pressures were plausible. The pressure gradient seems to be reversed but this cannot be commented upon until the non-constant portion of the gradient is retrieved. It must

be borne in mind that since the pressure is C^{-1} continuous at the element boundaries, the barycentric values are not sufficient to verify a correct treatment of the pressure terms.

Chapter 6

CONCLUSIONS & RECOMMENDATIONS

6.1 Conclusions

The events described in this thesis represent a series of contributions not necessarily intended to move along a straight path from a beginning to some well defined end. These accomplishments are more clearly defined as solutions to problems along the course of development. The development of the 15 node brick means ASGARD's capability to solve the Navier-Stokes equations is poised ready for the enhancements required to develop it into a powerful tool.

The testing with the 20 node brick clearly demonstrated the feasibility of using finite elements in the prediction of turbulent flows. The weak point in such a procedure is the availability of dependable and easily incorporated turbulence models. Simplicity in such models is often achieved by a reduction in its range of applicability. On the other hand, complex models are often cumbersome to use and are too dependent on correctly modifying constants which depend on the geometry of the problem. These models may be made to yield good agreement with experiment but lack the autonomy required of a computer program. The technique of treating a routine returning nodal or Gauss point turbulence values as a "black box" is advantageous since no modifications to the code must be made to test new models.

The 20 node brick element began originally as an external program using several ASGARD routines. One of the goals at the onset of this research was to incorporate it into the ASGARD package as another element in the list of those available. This was achieved and the element was then replaced with the 15 node brick. The code for the 20 node brick was archived and the element number reassigned to the 15 node brick. Thus, at the conclusion of this work, all code has been returned to the ASGARD program and is available using the methods described in Appendix A.

The reduction in the time required in the frontal solver is dramatic. Since most of the run time is spent in this procedure, the reduction has a very noticeable impact on the solution time of test problems. Attempting further reductions in the time spent in the solver would probably be less productive since the point of diminishing returns has probably been reached.

The research has shown the feasibility of implementing into ASGARD the capacity to solve fluid flow problems. This ability should prove to be a powerful tool which could be used for providing further insight into the physics of many flows. If the analyst bears in mind the restrictions and assumptions inherent to the method as well as those explicitly made in the course of preparing a problem for solution, the results should prove useful. It is hoped that this capability will not be left idle but rather, will be exploited in an attempt to enhance the power and ease of use.

6.2 Future Work

As a first step in that future development, a routine should be added which will recover the non-constant portion of the pressure which was removed as part of the formulation. This merely requires the solution of the Navier-Stokes equation in three dimensions at the element level using the computed velocity field. Clearly, the pressure gradients are then the only three unknowns.

The three dimensional capabilities of the 15 node brick remain to be demonstrated. This verification should be conducted by solving the duct problem (used with the 20 node brick) since the FD solution provides an approximate solution to the field. This test is fully three dimensional and very non-linear and would be a good test for the robustness of the code.

As a result of continued development, the need for an output file post-processor will certainly be seen again. The one written for the 20 node brick, SEPARATE, can be used as a start towards that goal and the changes required for use with the 15 node brick should prove minimal.

Having demonstrated the three dimensionality of the element, the next logical step is the development of the ability to run turbulent flow cases. Many of the problems of interest lie in this regime and its inclusion is a must for a truly general code. Although the user

would almost certainly be required to input whether the flow is laminar or turbulent at the beginning of the development, a test for turbulence could probably be written which would allow a more general approach to the problem. Checking the velocity to obtain a Re value, the code could make the necessary transition to the addition of turbulence terms where appropriate. This level of sophistication is certainly not achievable in the very near future but is not an unreasonable goal.

References .

Bathe K.J.	1982	<i>Finite Element Procedures in Engineering Analysis</i> , Prentice-Hall Inc., Englewood Cliffs, New Jersey, 07632
Chapman D.R.	1980	"Trends and Pacing Items in Computational Aerodynamics", <i>Proceedings of the Seventh International Conference on Numerical Methods in Fluid Dynamics</i> , Lecture Notes In Physics, Vol. 141, pp 2-11
Donea J., Giuliani S., Morgan K., and Quartapelle L.	1981	"The Significance of Chequerboarding in a Galerkin Finite Element Solution of the Navier-Stokes Equations", <i>International Journal For Numerical Methods In, Engineering</i> , Vol. 17, No. 5, pp 799-795
Fortin A.	1985	Private Communications, Carleton University, Ottawa and Ecole Polytechnique de Montreal, Montreal
Fortin A. and Fortin M.	1985	"Experiments With Several Elements for Viscous Incompressible Flows", <i>Ecole Polytechnique de Montreal Report EPM/RT-85-9</i>
Fortin M.	1981	"Old and New Finite Elements for Incompressible Flows", <i>International Journal For Numerical Methods In Fluids</i> , Vol. 1, pp 347-364
Fortin M. and Fortin A.	1984a	"Newer and Newer Elements for Incompressible Flow", <i>Ecole Polytechnique de Montreal Report EPM/RT-84-10</i>
Fortin M. and Fortin A.	1984b	"A Generalization of Uswawa's Algorithm for the Solution of the Navier-Stokes Equations", <i>Ecole Polytechnique de Montreal Report EPM/RT-84-25</i>
Goldak J.	1985	Private Communications, Carleton University, Ottawa
Hallquist J.O. and Flower E.	1986	"Workshop on DYNA3D/NIKE3D", Workshop con- ducted at Carleton University 12-13 May 1986
Hinze J.O.	1959	<i>Turbulence</i> , McGraw-Hill Book Company, Toronto

- Hood P. 1976 "Frontal Solution Program for Unsymmetric Matrices", *International Journal For Numerical Methods In Engineering*, Vol. 10, pp 379-399
- Hood P. 1977 "Note On Frontal Solution Program for Unsymmetric Matrices", *International Journal For Numerical Methods In Engineering*, Vol. 11, pg 1055
- Hood P. and Taylor C. 1974 "Navier-Stokes Equations Using Mixed Interpolation", *Finite Element Methods in Flow Problems*, edited by J.T. Oden, O.C. Zienkewicz, R.H. Gallagher, and C. Taylor, International Symposium on FEM in Flow Problems, January 1974, Swansea, UK, UAH Press, Huntsville, Alabama
- Huebner K.H. and Thornton E.A. 1982 *The Finite Element Method for Engineers*, John Wiley and Sons, New York, Second Edition
- Irons B. and Ahmad S. 1984 *Techniques Of Finite Elements*, Ellis Horwood Ltd., John Wiley and Sons, Toronto
- Knuth D.E. 1984 *The TEXBook*, Addison-Wesley Publishing Company, Don Mills, Ontario
- Kutler P. 1985 "A Perspective of Theoretical and Applied Computational Fluid Dynamics", *AIAA Journal*, Vol. 23, No. 3, March 1985, pp 328-341
- Laufer J. 1954 "The Structure of Turbulence in Fully Developed Pipe Flow", *National Advisory Committee For Aeronautics (NACA) Technical Report No. 1174*
- Rodi W. 1982 "Examples of Turbulence Models for Incompressible Flows", *AIAA Journal*, Vol. 20, No. 7, July 1982, pp 872-879
- Sani R.L., Gresho P.M., Lee R.L., and Griffiths D.F. 1981 "The Cause and Cure of the Spurious Pressures Generated by Certain FEM Solutions of the Incompressible Navier-Stokes Equations: Part I", *International Journal For Numerical Methods In Fluids*, Vol. 1, No. 1, pp 17-43

Schlichting H.	1960	<i>Boundary Layer Theory</i> , McGraw-Hill Book Company Inc., Toronto, Fourth Edition
Sjolander S.A.	1985	Private Communications, Carleton University, Ottawa
Spencer A.J.M., et al.	1977	<i>Engineering Mathematics</i> , Van Nostrand Reinhold Company Limited, Toronto, Vols. I and II
Taylor C. and Hood P.	1973	"A Numerical Solution of the Navier-Stokes Equations Using the Finite Element Technique", <i>Computers and Fluids</i> , Vol. 1, No. 1, January 1973, pp 73-100
Taylor and Hughes T.G.	1981	<i>Finite Element Programming of the Navier-Stokes Equations</i> , Pineridge Press Limited, Swansea, UK
Vernaganti G.R.	1984	<i>Finite Element Solution of the Navier-Stokes Equations</i> , Master of Engineering Thesis, Department of Mechanical and Aeronautical Engineering, Carleton University, Ottawa
White F.M.	1974	<i>Viscous Fluid Flow</i> , McGraw-Hill, Toronto
Zienkewics O.C.	1979	<i>The Finite Element Method</i> , McGraw-Hill Book Company (UK) Limited, Toronto, Third Edition
Zienkewics O.C. and Morgan K.	1983	"Higher Order Finite Element Approximation", <i>Finite Elements and Approximations</i> , John Wiley and Sons, USA, pp 181-192

Unit cube
 Twelve midedge nodes
 Eight corner nodes

Axes

Figure 1: Twenty Node Brick Topology

Unit cube
 Six midface nodes
 Eight corner nodes
 One barycentric node

Axes

Figure 2: Fifteen Node Brick Topology

Machine = APOLLO DN-660

$$\text{Savings} = \frac{(1620 - 390)}{1620} = 75.9\%$$

Figure 3: Reduction in Three Element Solution Time

- = Direction of flow
- = Node fixed at zero velocities
- = Node with non-zero streamwise velocity

Figure 4: One Element Poiseuille Flow Geometry

Unit cube
 Twelve midedge nodes
 Eight corner nodes

Axes

Figure 5: NOXAXL Degrees of Freedom

→ = Direction of flow

Top View

Figure 6: Diverging Section Geometry

20 Node Bricks

2 Planes Of Symmetry

→ = Direction of flow

Figure 7: Three Element Pipe Flow Geometry

16 Elements
4 at each cross section

→ = Direction of flow

Figure 8: Curved Duct Geometry

Figure 9: One Element Test - Fifteen Node Brick

- = Direction of flow
 • = Node fixed at zero velocities
 ○ = Node with non-zero velocity

Figure 10: Two Element Test - Fifteen Node Brick

Figure 11: Poiseuille Flow Velocity Profile

Diverging Section Velocity Profile

Figure 12: Diverging Section Velocity Profile

Laminar Pipe Flow

Figure 13: Laminar Pipe Flow Velocity Profile

Pipe NOXAXL Flow Velocity Profile

Figure 14: NOXAXL Laminar Pipe Flow Velocity Profile

Turbulent Pipe Flow

Figure 15: Turbulent Pipe Flow Velocity Profile

Figure 16: Velocities at the Exit Plane of the Curved Duct

ASGARD

--- START & other statistics routines -- OPEN (files also opened in MATRON)

```

-- NURSE --
--   -- UINOUT
--   -- GETMEM
--   -- (DOCTOR -- WRERR)

-- DOCTOR -- WRERR

-- INPUT -- (DOCTOR -- WRERR)

-- MATRON --
--   -- (DOCTOR -- WRERR)
--   -- SECSYM
--   -- NRECS

-- GETMEM

-- NEWLOD -- RDTEMP -- (DOCTOR -- WRERR)

-- ALGOR

-- DOCTOR -- WRERR
  
```

FLFILE

ASGARD - (NDFBLK)

```

-- FRONT --
--   -- RESIDU
--   -- POSTLN
--   -- NUGNU
--   -- D2STAT --
--   -- NORM --
--   -- TENEXA
--   -- etc.

-- FRONTN --
--   -- INCREM
--   -- VSMA

-- FRONTA --
--   -- RESOL
--   -- BACSUB
--   -- ABFIND
--   -- TOLREL
--   -- MEMBER

-- POSPRO --
--   -- ROTBAC
--   -- VPLOAD
--   -- JACOD
--   -- BLARGE
--   -- LINGNL
--   -- INVVAR
--   -- EL2EFF
--   -- EL3EFF
--   -- JACOB3D
--   -- BLARG3
--   -- INVR3D
--   -- STPOST
--   -- LINESR
--   -- RESIDPL
--   -- UPDACC
--   -- DSPOUT
--   -- UPDACC
--   -- DSPOUT
--   -- JACOB3D
--   -- BLARGE
--   -- LINGNL
--   -- INVVAR
--   -- ADV2D
--   -- RESLOD
--   -- D2PLAS
--   -- INCVD
--   -- D2GAP
--   -- D3PLAS
--   -- D3ETPL
--   -- D3GAP
--   -- DY2DRS
--   -- DY3DRS
--   -- CONOUT
--   -- STRSPL
--   -- NUGNU
  
```

-- ENDING and other statistics routines (note, ENDING called with any internal error in ASGARD)

Figure 17: ASGARD Flow Chart

ASGARD

```
-- SHELL
- BEAM
- BR20F
- TR16F
- Q08F
- QSS2D
- BARIS (2S)(3S)
- BARIF
- FTID --
```

```
-- FT2D --
- MIDSID
- GAUS1D
- SF1D
- SUBDOM2D
- GAUSH2D
- GAUS2D
- SFTRIA
- SFHQD
- SFQ02D -- JACO82D
- THPROP
- NTALPY
- URMATM
```

-- ELFILE -----
(get stiffness
matrix)

```
-- FT3D --
- MIDSID
- FACFID
- GAUS2D
- DAREA
- NTALPY
- SUBDOM3D
- GAUSH3D -- SFBR20
- GAUS3D -- SFBR8
- BRICK3D -- JACO83D
- THPROP -- CARTE3D
- URMATM -- INVJ
- POINTS3D
```

ASGARD

et stiffness
matrix

-- PLAS2D --	--	- TEMPRP	
		- MIDSID	
- DYNAM2D	-	- DMTRIX	
		- GAUS1D	
- CONT2D	--	- SF1D	
		- TMSID	
		- SUBDOM2D	
		- GAUSH2D	
		- GAUS2D	
		- SFTRIA	
		- SFQD2D	
		- SFHQD	
		- TEMPRP	
		- DMTRIX	
		- INVAR	
		- ADV2D	
		- GMSTIF	
	--	- TEMPRP	
		- MIDSID	
		- MIDVAR	
		- DMTRIX	
		- GAUS2D	
		- DAREA	
		- SFQD2D	-- JACOB2D
		- TMSID	
- PLAS3D	--	- TMDVAR	-- SFBR20
		- BRICK3D	- SFBR8
- DYNAM3D	-		- SFHBR
			- JACOB3D
- CONT3D	--		- CARTE3D
			- INVJ
			- POINTS3D
			- SFQCON (for CONT3D)
		- TEMPRP	
		- DMTRIX	
		- INVR3D	
		- YILD3D	
		- FLPL3D	

get stiffness
matrix

- BEAMH
- BMC3D
- SHC3D
- STAT2D
- FTSHEL
- SHELL
- GAP2D
- ETPL3D
- VFL3D14 ---

- INIT	-- SFRHI5	
- MIDSIDH	- JACOB3D	-- SFRHI5
- GEOM -----	- GAUS3D	- JACOB3D
- SFRHI5	- BRICKH ---	- INVJ
- JACOB3D		- CARTE3D
- INVJ		- POINTS3D
- GAUS3D	-- SFRHI5	
- BRICKH ----	- JACOB3D	
- LAUFTURB	- INVJ	
- STIFCHECK	- CARTE3D	
	- POINTS3D	

Figure 18: Curved Duct Pressure Contours

Table 1: 20 Node Brick Shape Functions

Shape Functions	
No.	Shape Function, N
1	$\frac{1}{8}(1-\eta)(1-\zeta)(1-\xi)$
2	$\frac{1}{8}(1+\eta)(1-\zeta)(1-\xi)$
3	$\frac{1}{8}(1+\eta)(1-\zeta)(1+\xi)$
4	$\frac{1}{8}(1-\eta)(1-\zeta)(1+\xi)$
5	$\frac{1}{8}(1-\eta)(1+\zeta)(1-\xi)$
6	$\frac{1}{8}(1+\eta)(1+\zeta)(1-\xi)$
7	$\frac{1}{8}(1+\eta)(1+\zeta)(1+\xi)$
8	$\frac{1}{8}(1-\eta)(1+\zeta)(1+\xi)$
9	$\frac{1}{4}(1-\eta^2)(1-\zeta)(1-\xi)$
10	$\frac{1}{4}(1+\eta^2)(1-\zeta)(1-\xi)$
11	$\frac{1}{4}(1-\eta^2)(1-\zeta)(1+\xi)$
12	$\frac{1}{4}(1-\eta^2)(1-\zeta)(1-\xi^2)$
13	$\frac{1}{4}(1-\eta^2)(1-\zeta)(1+\xi)$
14	$\frac{1}{4}(1+\eta^2)(1-\zeta)(1-\xi)$
15	$\frac{1}{4}(1+\eta^2)(1-\zeta)(1+\xi)$
16	$\frac{1}{4}(1-\eta^2)(1-\zeta)(1+\xi)$
17	$\frac{1}{4}(1-\eta^2)(1+\zeta)(1-\xi)$
18	$\frac{1}{4}(1+\eta^2)(1+\zeta)(1-\xi)$
19	$\frac{1}{4}(1-\eta^2)(1+\zeta)(1+\xi)$
20	$\frac{1}{4}(1-\eta^2)(1+\zeta)(1-\xi^2)$

Table 2: 20 Node Brick Shape Function Derivatives

Shape Functions Derivatives			
No.	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$	$\frac{\partial N}{\partial \zeta}$
1	$-\frac{1}{8}(1-\eta)(1-\zeta)$	$-\frac{1}{8}(1-\zeta)(1-\xi)$	$-\frac{1}{8}(1-\eta)(1-\xi)$
2	$-\frac{1}{8}(1+\eta)(1-\zeta)$	$\frac{1}{8}(1-\zeta)(1-\xi)$	$-\frac{1}{8}(1+\eta)(1-\xi)$
3	$\frac{1}{8}(1+\eta)(1-\zeta)$	$\frac{1}{8}(1-\zeta)(1+\xi)$	$-\frac{1}{8}(1+\eta)(1+\xi)$
4	$-\frac{1}{8}(1-\eta)(1-\zeta)$	$-\frac{1}{8}(1-\zeta)(1+\xi)$	$-\frac{1}{8}(1-\eta)(1+\xi)$
5	$-\frac{1}{8}(1-\eta)(1+\zeta)$	$-\frac{1}{8}(1+\zeta)(1-\xi)$	$-\frac{1}{8}(1-\eta)(1-\xi)$
6	$-\frac{1}{8}(1+\eta)(1+\zeta)$	$\frac{1}{8}(1+\zeta)(1-\xi)$	$\frac{1}{8}(1+\eta)(1-\xi)$
7	$\frac{1}{8}(1+\eta)(1+\zeta)$	$\frac{1}{8}(1+\zeta)(1+\xi)$	$\frac{1}{8}(1+\eta)(1+\xi)$
8	$\frac{1}{8}(1-\eta)(1+\zeta)$	$-\frac{1}{8}(1+\zeta)(1+\xi)$	$\frac{1}{8}(1-\eta)(1+\xi)$
9	$-\frac{1}{4}(1-\eta^2)(1-\zeta)$	$-\frac{1}{4}\eta(1-\zeta)(1-\xi)$	$-\frac{1}{4}(1-\eta^2)(1-\xi)$
10	$-\frac{1}{4}\xi(1+\eta)(1-\zeta)$	$\frac{1}{4}(1-\zeta)(1-\xi^2)$	$-\frac{1}{4}(1+\eta)(1-\xi^2)$
11	$\frac{1}{4}(1-\eta^2)(1-\zeta)$	$-\frac{1}{4}\eta(1-\zeta)(1+\xi)$	$-\frac{1}{4}(1-\eta^2)(1+\xi)$
12	$-\frac{1}{4}\xi(1-\eta)(1-\zeta)$	$-\frac{1}{4}(1-\zeta)(1-\xi^2)$	$-\frac{1}{4}(1-\eta)(1-\xi^2)$
13	$-\frac{1}{4}(1-\eta)(1-\zeta^2)$	$-\frac{1}{4}(1-\zeta^2)(1-\xi)$	$-\frac{1}{4}\zeta(1-\eta)(1-\xi)$
14	$-\frac{1}{4}(1+\eta)(1-\zeta^2)$	$\frac{1}{4}(1-\zeta^2)(1-\xi)$	$-\frac{1}{4}\zeta(1+\eta)(1-\xi)$
15	$\frac{1}{4}(1+\eta)(1-\zeta^2)$	$\frac{1}{4}(1-\zeta^2)(1+\xi)$	$-\frac{1}{4}\zeta(1+\eta)(1+\xi)$
16	$\frac{1}{4}(1-\eta)(1-\zeta^2)$	$-\frac{1}{4}(1-\zeta^2)(1+\xi)$	$-\frac{1}{4}\zeta(1-\eta)(1+\xi)$
17	$-\frac{1}{4}(1-\eta^2)(1+\zeta)$	$-\frac{1}{4}\eta(1+\zeta)(1-\xi)$	$\frac{1}{4}(1-\eta^2)(1-\xi)$
18	$-\frac{1}{4}\xi(1+\eta)(1+\zeta)$	$\frac{1}{4}(1+\zeta)(1-\xi^2)$	$\frac{1}{4}(1+\eta)(1-\xi^2)$
19	$\frac{1}{4}(1-\eta^2)(1+\zeta)$	$-\frac{1}{4}\eta(1+\zeta)(1+\xi)$	$\frac{1}{4}(1-\eta^2)(1+\xi)$
20	$-\frac{1}{4}\xi(1-\eta)(1+\zeta)$	$-\frac{1}{4}(1+\zeta)(1-\xi^2)$	$\frac{1}{4}(1-\eta)(1-\xi^2)$

Table 3: 15 Node Brick Shape Functions

Shape Functions	
No.	Shape Function, N
1	$\frac{1}{8}(1 - \eta)(1 - \zeta)(1 - \xi)$
2	$\frac{1}{8}(1 + \eta)(1 - \zeta)(1 - \xi)$
3	$\frac{1}{8}(1 + \eta)(1 - \zeta)(1 + \xi)$
4	$\frac{1}{8}(1 - \eta)(1 - \zeta)(1 + \xi)$
5	$\frac{1}{8}(1 - \eta)(1 + \zeta)(1 - \xi)$
6	$\frac{1}{8}(1 + \eta)(1 + \zeta)(1 - \xi)$
7	$\frac{1}{8}(1 + \eta)(1 + \zeta)(1 + \xi)$
8	$\frac{1}{8}(1 - \eta)(1 + \zeta)(1 + \xi)$
9	$\frac{1}{8}(1 - \eta^2)(1 - \zeta^2)(1 - \xi^2)$
10	$\frac{1}{8}(1 - \eta)(1 - \zeta^2)(1 - \xi^2)$
11	$\frac{1}{8}(1 + \eta)(1 - \zeta^2)(1 - \xi^2)$
12	$\frac{1}{8}(1 - \eta^2)(1 - \zeta^2)(1 - \xi)$
13	$\frac{1}{8}(1 - \eta^2)(1 - \zeta^2)(1 + \xi)$
14	$\frac{1}{8}(1 - \eta^2)(1 - \zeta)(1 - \xi^2)$
15	$\frac{1}{8}(1 - \eta^2)(1 + \zeta)(1 - \xi^2)$

Table 4: 15 Node Brick Shape Function Derivatives

Shape Functions Derivatives			
No.	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$	$\frac{\partial N}{\partial \zeta}$
1	$-\frac{1}{8}(1-\eta)(1-\zeta)$	$-\frac{1}{8}(1-\zeta)(1-\xi)$	$-\frac{1}{8}(1-\eta)(1-\xi)$
2	$-\frac{1}{8}(1+\eta)(1-\zeta)$	$-\frac{1}{8}(1-\zeta)(1-\xi)$	$-\frac{1}{8}(1+\eta)(1-\xi)$
3	$\frac{1}{8}(1+\eta)(1-\zeta)$	$-\frac{1}{8}(1-\zeta)(1+\xi)$	$-\frac{1}{8}(1+\eta)(1+\xi)$
4	$\frac{1}{8}(1-\eta)(1-\zeta)$	$-\frac{1}{8}(1-\zeta)(1+\xi)$	$-\frac{1}{8}(1-\eta)(1+\xi)$
5	$-\frac{1}{8}(1-\eta)(1+\zeta)$	$-\frac{1}{8}(1+\zeta)(1-\xi)$	$\frac{1}{8}(1-\eta)(1-\xi)$
6	$-\frac{1}{8}(1+\eta)(1+\zeta)$	$-\frac{1}{8}(1+\zeta)(1-\xi)$	$\frac{1}{8}(1+\eta)(1-\xi)$
7	$\frac{1}{8}(1+\eta)(1+\zeta)$	$\frac{1}{8}(1+\zeta)(1+\xi)$	$\frac{1}{8}(1+\eta)(1+\xi)$
8	$\frac{1}{8}(1-\eta)(1+\zeta)$	$-\frac{1}{8}(1+\zeta)(1+\xi)$	$\frac{1}{8}(1-\eta)(1+\xi)$
9	$-2\xi(1-\eta^2)(1-\zeta^2)$	$-2\eta(1-\zeta^2)(1-\xi^2)$	$-2\zeta(1-\eta^2)(1-\xi^2)$
10	$-\xi(1-\eta)(1-\zeta^2)$	$-\frac{1}{2}(1-\zeta^2)(1-\xi^2)$	$-\zeta(1-\eta)(1-\xi^2)$
11	$-\xi(1+\eta)(1-\zeta^2)$	$\frac{1}{2}(1-\zeta^2)(1-\xi^2)$	$-\zeta(1+\eta)(1-\xi^2)$
12	$-\frac{1}{2}(1-\eta^2)(1-\zeta^2)$	$-\eta(1-\zeta^2)(1-\xi)$	$-\zeta(1-\eta^2)(1-\xi)$
13	$\frac{1}{2}(1-\eta^2)(1-\zeta^2)$	$-\eta(1-\zeta^2)(1+\xi)$	$-\zeta(1-\eta^2)(1+\xi)$
14	$-\xi(1-\eta^2)(1-\zeta)$	$-\eta(1-\zeta)(1-\xi^2)$	$\frac{1}{2}(1-\eta^2)(1-\xi^2)$
15	$-\xi(1-\eta^2)(1+\zeta)$	$-\eta(1+\zeta)(1-\xi^2)$	$\frac{1}{2}(1-\eta^2)(1-\xi^2)$

2

Micro-D

Appendix A

DESCRIPTION OF ASGARD STRUCTURE AND USE

STRUCTURE

ASGARD is a Finite Element Method (FEM) analysis package comprised of several subroutines. The main root (also referred to as ASGARD) is responsible for initiating the appropriate calls to the required subroutine in the correct sequence. Each facet of the FEM analysis is contained in a separate subroutine thus allowing efficient debugging of small modules. In general, ASGARD is broken into seven (7) smaller segments where subroutines of similar function are grouped. The number of segments was the result of an observation that various problems made use of particular routines. For instance, a plasticity analysis used the /ASGARD/MAIN directories below (as all ASGARD problems do) but made subsequent use primarily of those routines placed in /ASGARD/NONLINEAR/ELEM_STRESS. Similar observations led to the creation of the seven directories whose naming paths appear below:

```
/USER/ASGARD/MAIN/FIRST  
/USER/ASGARD/MAIN/SECOND  
/USER/ASGARD/NONLINEAR/ELEM_HEAT  
/USER/ASGARD/LINEAR/ELEM_STRESS  
/USER/ASGARD/LINEAR/MISC  
/USER/ASGARD/NONLINEAR/ELEM_STRESS  
/USER/ASGARD/NONLINEAR/MISC
```

The basic philosophy and procedure of ASGARD with regards to the fluid element only will be considered although these procedures vary little with other element types. A diagram showing the chronological sequence of subroutine calls is included in the thesis as Figure 12. The program ASGARD is the first module referenced when the program starts. ASGARD begins by calling START which is used as a job statistics subroutine starting such values as the real time clock and the CPU clock so that at the end of the run, the

run times may be displayed. The next subroutine of importance called is NURSE. This is where the first few lines of the input file are read (see the 'Pre-processing' section of Usage Procedure of this appendix for a detailed description of the input file structure). NURSE begins by reading the list of global parameters from the first line of the file and goes on to read some other data. NURSE also begins the dimensioning of the vector VEC/NVEC and several other vectors and arrays.

The names of the subroutines may, at first, seem confusing. Their roots are well founded, however, and clearly express the subroutine's function. NURSE is meant to perform some initial diagnoses on the 'patient' (the test case in question) and to prescribe some basic 'medications' (the initial dimensioning of VEC/NVEC). If NURSE detects a flaw in the input information, it consults an expert at determining the cause of ailments, DOCTOR. DOCTOR outputs a message describing the ailment and decides whether or not to continue the run (i.e., whether the problem is terminal or not). Once NURSE's level of verification has been satisfied, a new list of information is read in by the subroutine INPUT and control is passed to a more 'knowledgeable' member of the staff, MATRON. Notice that the controlling authority is still ASGARD and that it is responsible for the chronological development of the solution to the test case.

MATRON performs a new series of tests on the information and, like its predecessor NURSE, passes control to the ultimate prognosis authority, DOCTOR, if it finds an error. MATRON is also responsible for the detailed dimensioning of arrays in VEC/NVEC for such quantities as the element stiffness matrices, global stiffness matrices, and other quantities stored in VEC/NVEC. Finally, MATRON opens several scratch files used for storage of information. With this done, ASGARD calls NEWLOD to read from the input file the data characterizing the present load or time increment. This includes the convergence and time iteration data.

The real computation begins at this point by determining the values of several flags which are used to indicate the type of problem being solved. Much of this is done in the subroutine ALGOR. ALGOR sets NALGOR and KRESL which are used to characterize the differential equation and solver, respectively. These NALGOR values would serve to distinguish between, for example, a heat transfer, a stress analysis, or a fluid flow case. This allows several different element types to be used for similar problems such as the use of both 8 node and 20 node bricks for the solution of heat conduction problems.

Given all this preparation, the manipulation of problem values begins by the calculation of the element stiffness matrices. ELFILE is a controller for all the element stiffness matrix and load vector generators. It uses LTYPE as the basis for its choice of generator which, for the 15 node brick fluid element case, is dubbed VFL3D14. The archived 20 node brick's counterpart was called VFL3D20. At each of the sequential calls to VFL3D14 from ELFILE, corresponding to each of the elements in the problem, an element stiffness matrix is created and written to the disc for the subsequent read from the frontal solver. Thus, VFL3D14 is called NELZ times per iteration. With a series of element stiffness matrices created and stored on the appropriate disc, ASGARD calls on FRONTA to assemble the contributions into a global stiffness matrix. This global matrix is then used to calculate the values of the unknown degrees of freedom using a frontal elimination, back-substitution technique.

Since the fluid flow case is a nonlinear phenomenon, the solution must be an iterative one which necessarily entails the use of a convergence limit to check whether the current solution is close enough to the previous solution to be accepted. This verification is carried out in the subroutine TOLREL. If the test is positive (i.e., the solution has converged), a flag is tripped which causes ASGARD to stop the run. If the solution has not converged, ASGARD repeats the solution procedure beginning at the call to ELFILE until either the solution does converge or the maximum number of iterations specified by the user is reached. The subroutine ENDING is then called which outputs the job statistics START began and closes the scratch files opened by MATRON and OPEN before stopping.

ALTERATIONS

The wide diversity of ASGARD's capabilities necessarily entails a great deal of code. The entire ASGARD package consists of some 50,000 lines of FORTRAN code spread over the tens of subroutines grouped into the seven directories listed above. The basic procedures involved in FEM vary little, however, from one problem to the next so they may be used without modification with most new elements incorporated. In essence, this means that a new user of ASGARD need not concern himself with the Gaussian integration procedure, for example, since this procedure has been well written and prepared for general use. Thus, to incorporate the new viscous flow element (element type # 144) required modification of the following subroutines only:

ALGOR

ASGARD
ELFILE
INPUT
MATRON
NDFBLK
NEWLOD
NUGNU
NURSE

Of course, a new element required an element stiffness generator which had to be added to the package and, since the fluid flow problem gives rise to asymmetric element and global stiffness matrices (a phenomena previously unencountered in ASGARD), a frontal solver capable of solving asymmetric matrices was also added to the package. These two subroutines, whose functions are described above, were called VFL3D14 and FRONTA respectively. Although the 20 node brick's element stiffness generator, VFL3D20, was added to ASGARD, it was removed in favour of this new routine, VFL3D14.

USAGE PROCEDURE

To run a particular test case with ASGARD consists of three phases:

pre-processing;
processing; and
post-processing.

Basically, the pre-processing stage involves the preparation of a detailed input file in a very specific sequence which ASGARD then uses to carry out the processing phase of the operation. When ASGARD has completed the calculations to within some user controlled tolerance, the program stops. The post-processing phase of the procedure can then begin. This part allows the user to graph the mesh used for problem solution. Various plotting packages are also available to graph converged data in order to view results in an easier form.

PRE-PROCESSING

As mentioned above, the input file must follow a strict format in order to be accepted for processing. Because of the large volume of information required, many subroutines

- 1) Global parameters: all on first line

NELZ, NFIX, NLOAD, NEXTIF, NROTAT, NSECSM, NEWRHS, MAXRHS,
ITERAT, TITLEN, TEMPLN, NCORD, NALGOR, MINCRE, MITERA

- 2) Title and suitable description: next TITLEN lines

- 3) Element topology: one line for each element (NELZ lines long)

NEL, LTYP, LPOP, LNODS

- 4) Nodal coordinates: one line for each node

NODE NO., XCOORD, YCOORD, ZCOORD

- 5) Prescribed velocities and pressures: NFIX lines long

SEQUENCE NO., NODE NUMBER, KODFIX, PRESCRIBED VALUES

- 6) Property list: one line for each property list

SEQUENCE NO., DYNVIS, RHOIN, BFORX, BFORZ, BFORZ,
LCOUNT, GSCHM, NSPL, PENALTY

- 7) Convergence parameters: one line per increment

THETA, DTINT, TOLERA, FIXSTP

PROCESSING

If a usable rununit is available, the pathname to the unit followed by the name of the input and output files respectively need only be input. For example, the author's computer account structure would allow him to run a test case with input file name TEST and place the output in a file named TEST_OUT by typing the following sequence (note that the dollar sign is a system prompt):

```
$ /user/dulude/navier/rununits/nseqn.run TEST TEST_OUT
```

Of course, if a macro has been prepared to emulate the absolute pathname of the rununit above, it can be used in the same way. For example, the author has one macro for compiling the FORTRAN source code, one to create the rununit, another acting as a link to execute the run by calling on the rununit, and yet another to prepare the debugger available on the APOLLO network. The DOMAIN reference manual should be consulted for a detailed explanation of macros. Since all the FEM code resulting from this research has been returned to the ASGARD directories, the problem may be submitted by using the ASGARD macro:

ASGARD TEST TEST_OUT

Whichever method is used, the user should then see the phrase:

ASGARD run started.

This signals that the run has begun. Since an output file is used, very little data is displayed on the screen. The number of iterations and increments completed should also sequentially appear. The root mean square value of the sum of the elemental discrete divergence magnitudes also appears at the end of each iteration. For numerical results and other program generated messages, the file TEST_OUT (in this case) must be consulted.

POST-PROCESSING

The post-processing provides the means to reduce the large amounts of output into a more comprehensible format. This is accomplished by the use of a post-processor called SEPARATE and a plotting routine called PLOT2D. SEPARATE was written for the 20 node brick fluid element only and will require modification for use with the 15 node brick.

The SEPARATE post-processor uses ASGARD's output file as input and returns the iterated pressure and velocity values in a more readable, tabular form. The type of output is chosen by the user from a program menu. Currently, the menu allows the output to be reorganized to yield either of:

- 1) Four tables (u,v,w, and p) with the iterated values of each forming the subsequent columns of the table and each row representing a unique node.
- 2) NODMAX tables of only four rows length thus allowing the progress of individual nodes to be easily tracked.

The number of columns in each of the output tables has been restricted to correspond to five (5) 'y' values commensurate with the restriction imposed by the plotting routine, PLOT2D.

SEPARATE also has the capability to remove the nodes with prescribed values from the final output or to output only those nodes requested by the user. The choice of output is made by once again responding to a displayed menu. The removal of these nodes is

of obvious benefit when considering the invariant nature of prescribed values or when the progress of a particular set of nodes is desired.

GLOSSARY

Although the glossary presented below is by no means an exhaustive compendium of ASGARD terminology, it does encompass the terms most often dealt with in the use and alteration of the ASGARD subroutines described in the first sections of this appendix. The subroutines most often associated with particular variables have been listed for easy tracing. Where applicable, desired or recommended values have been included and have been annotated as such. In addition, default values have been given where such a condition exists. Where a term is specific to VFL3D20 it has been shown as such. Otherwise, VFL3D14 is given as a reference even if the quantity is common to both elements.

The subroutines listed for each of the terms are the result of the author's experience in dealing with the fluid mechanics routines. This does not mean that each of the terms are restricted to the routines listed. More precisely, as the fluid flow analysis capability of ASGARD is enhanced, it will certainly be found that these terms appear in other routines which perform other ASGARD calculations.

- BFORX - body force term in the x direction
  - input file data
  - subroutines - VFL3D14
- BFORY - body force term in the y direction (see BFORX)
- BFORZ - body force term in the z direction (see BFORX)
- COORD - array of size NODMAX by NCORD containing coordinates
  - of nodes as well as values of u, v, w, and direction cosines
  - generated from input data read in INPUT and calculations carried out in VFL3D14
  - subroutines - most
- DTINT - time increment for nonsteady cases
  - input file data
  - set to 0 for the fluid case

- subroutines - NEWLOD
- DYNVIS - dynamic viscosity, μ
- input file data
- subroutines - VFL3D14
- ELSTIF - element stiffness matrix
- generated in VFL3D14
- subroutines - VFL3D14, ABFIND
- ERHS - element right hand side vector
- generated in VFL3D14
- subroutines - VFL3D14, ABFIND
- FIXSTP - number of increments with fixed time step DTINT
- input file data
- fixed to 0 for steady state problems
- subroutines - NEWLOD
- GSCHEM - Gaussian integration scheme order
- input file data
- set to 3 for 15 node brick
- subroutines - VFL3D14
- IFCOEF - flag used to indicate the re-solution case
- generated in ASGARD for use in MATRON
- set to 0 for the re-solution case, 1 for a normal solution
- subroutines - ASGARD, MATRON
- IFCOF - flag used for dimensioning of arrays to allow more
free core space in the case of a re-solution
- subroutines - MATRON
- INTEG - multiplicative factor to account for different byte lengths
required for real and integer variable storage
- set to 2 in ASGARD
- subroutines - ASGARD, MATRON, NURSE
- ITERAT - a flag for iterative (nonlinear) problems
- input file data
- fixed to 1 for nonlinear problems

- subroutines - NURSE
- KRESL - solution flag set for appropriate solver
  - set in ALGOR
  - subroutines - ALGOR
- KODFIX - array NFIX long containing the node's coded number
  - of prescribed degrees of freedom
  - input file data
  - subroutines - FRONTA
- LCOUNT - flag for identifying the flow type
  - input file data
  - set to 0 for inviscid flow, 1 for laminar, and
    - 2 or more for turbulent flow
  - subroutines - VFL3D14
- LENVEC - length of memory space required for test
  - calculated from TEMPLN in MATRON
  - subroutines - MATRON
- LFRO - element frontwidth
  - calculated for each element in MATRON
  - subroutines - MATRON
- LIMFRO - maximum frontwidth
  - deduced from maximum LFRO value in MATRON
  - subroutines - FRONTA, MATRON
- LNODS - array of size NELZ by LNOMAX used to store the node
  - numbers associated with each of the elements
  - assembled from input file data in NURSE
  - subroutines - most
- LNODZP - array giving the maximum number of nodes in an
  - element of the argument type
  - set in a data statement of NDFBLK
  - subroutines - NDFBLK
- LNOMAX - maximum node number of an element in a mesh
  - determined from input file element topology in NURSE

- subroutines - FRONTA, NURSE

LPOP - property list number
 - input file data
 - subroutines - NURSE, VFL3D14

LTYP - element type (# 144 in this case)
 - input file data
 - subroutines - NDFBLK, NURSE

LTYPE - array containing the element type of each element
           in the mesh
 - generated from input file element topology
 - subroutines - ELFILE, INPUT

LVABZ - number of degrees of freedom in each element
 - calculated in MATRON
 - subroutines - MATRON, FRONTA, VFL3D14

LVMAX - maximum number of elemental degrees of freedom in
           the problem
 - calculated in MATRON
 - subroutines - MATRON, FRONTA, VFL3D14

MAXRHS - maximum number of right hand sides expected for
           the re-splution case
 - input file data
 - default is 1
 - subroutines - MATRON

MINCRE - maximum number of increments in this problem
 - input file data
 - default is 1
 - subroutines - NURSE

MITERA - maximum number of iterations in any time step
 - input file data
 - default is 1
 - subroutines - FRONTA

NALGOR - algorithm number characterizing the particular

- problem being run
  - input file data
  - set to 43 for the Navier-Stokes formulation
  - subroutines - ALGOR, MATRON
- NBUF
  - intermediate buffer size used in calculating NBUFZ
  - calculated in MATRON
  - subroutines - MATRON
- NBUFZ
  - minimum buffer size left in the temporary length of memory allocated to the problem
  - calculated in MATRON
  - subroutines - MATRON
- NCORD
  - number of coordinates in the array COORD
  - set in NURSE for particular NALGOR and given as input file data
  - set to 7 corresponding to 3 dimensions, and either of 3 velocities or 1 velocity and 3 direction cosines
  - subroutines - same as COORD
- NDF
  - number of degrees of freedom of a particular node
  - given as data in the subroutine NDFBLK
  - equal to either 3 for corner and barycentric nodes or 4 for midface nodes
  - subroutines - FRONTA, NDFBLK, NURSE
- NDFPTR
  - pointer to the number of degrees of freedom of a node as stored in the NDF array
  - given as data in NDFBLK
  - subroutines - FRONTA, NDFBLK
- NDIM
  - number of dimensions in the problem
  - set in MATRON
  - set to 3 for NALGOR equal to 43
  - subroutines - MATRON, VFL3D14
- NEL
  - element currently in use
  - temporary variable

- subroutines - FRONTA, VFL3D14, ELFILE
- NELZ - number of elements in the problem
  - input file data
  - subroutines - FRONTA, MATRON, VFL3D14
- NEQUAT - number of equations to be treated in the solution
  - calculated in MATRON
  - subroutines - MATRON, FRONTA
- NEXTIF - used for heat transfer problems. Total number of nodes
  - with prescribed conductances.
  - set to 0 for the fluid case
- NEWRHS - number of new right hand sides expected in the case
  - of a re-resolution problem
  - input file data
  - default is 1
  - subroutines - MATRON
- NFIX - number of nodes with prescribed values
  - input file data
  - subroutines - FRONTA, MATRON
- NITERA - the current iteration number
  - incremented in ASGARD
  - subroutines - FRONTA
- NLOAD - total number of nodes with prescribed shear stresses
  - input file data
  - prescribed 0 for VFL3D14 problems
  - subroutines - MATRON
- NODFIX - array NFIX long containing the node number of
  - nodes with prescribed values
  - input file data
  - subroutines - MATRON, FRONTA
- NODMAX - maximum number of nodes in the problem
  - determined from input file element topology in NURSE
  - subroutines - most

- NROTAT** - number of nodes with rotated or inclined boundary conditions
- input file data
  - subroutines - MATRON
- NSECSM** - number of nodal pairs with sectorial symmetry
- input file data
  - prescribed 0 for VFL3D14 problems
  - subroutines - MATRON
- NSPL** - flag used to incorporate special properties in the element stiffness formulation
- input file data
  - set to 1 to prescribe no acceleration in the x direction for a particular 20 node brick element
  - no routines prepared for 15 node brick
  - subroutines - VFL3D14, VFL3D20
- NVEC** - long integer vector used to store many variables to facilitate passing of variables in subroutine calls
- constructed in MATRON and NURSE
  - subroutines - all
- OFFSET** - integer variable used on CP6 in order to allow dynamic memory management. The value is set to 0 on APOLLO since it is a virtual memory machine.
- RHOIN** - density of the fluid, ρ
- input file data
  - subroutines - VFL3D14
- ROFFSET** - same as OFFSET but for VEC rather than NVEC
- ROFSET** - pointer to the first real double precision variable in the array VEC
- SEQUENCE** - sequential progression of values used to
- NO.** index lists of variables such as a list of nodes with prescribed values.
- input file data

- should begin at 1 and be incremented by 1 for each successive entry in the list
  - subroutines - INPUT, NURSE
- THETA - relaxation parameter used to update the variables between iterations. A value between 0 and 1 corresponds to under-relaxation, between 1 and 2 implies over-relaxation
- used with 20 node brick
  - input file data
  - common example would be 0.5
  - subroutines - TOLREL, NEWLOD
- TITLEN - number of lines taken by the problem description and title in the input file data
- Input file data
- must be exactly the number of lines taken since this provides the benchmark from whence the program begins reading the input data
  - subroutines - NURSE
- TMPLEN - temporary length of memory required for problem solution
- input file data
  - must be smaller than $9.99E+06$. The larger the problem, the larger this value must be made.
  - was more important on CP-6. APOLLO version allows use of $TMPLEN = 0$ and memory requirements are set automatically.
  - set to 0
  - subroutines - NURSE, MATRON
- TOLERA - convergence criteria within which the successive change in variable values must lie in order to signify convergence
- input file data

- common example is 0.1 or 0.01
- subroutines - TOLREL, NEWLOD
- VEC - same as NVEC but for double precision real variables
rather than integer variables
- VISLAM - kinematic laminar viscosity, $\frac{\mu}{\rho}$
  - calculated in VFL3D14
  - subroutines - VFL3D14
- XCOORD - x value of the nodal coordinates
  - input file data
  - subroutines - MATRON, VFL3D14
- XGAUS - x value of the Gauss point coordinate
  - returned from Gaussian integration routine GAUS3D
  - subroutines - GAUS3D, VFL3D14
- YCOORD - y value of the nodal coordinates (see XCOORD)
- YGAUS - y value of the Gauss point coordinate (see XGAUS)
- ZCOORD - z value of the nodal coordinates (see XCOORD)
- ZGAUS - z value of the Gauss point coordinate (see XGAUS)

Appendix B

VFL3D14

SUBROUTINE VFL3D14(COORD,ELSTIF,IPROP,JPROP,LNODS,LNOMAX,
1 LPOP,NDIM,NEL,NELZ,LVABZ,LNODZ,LTYPE,NODMAX,NITERA,VPROP,
2 VFIX,NODFIX,NEWRHS,NDFMAX,NFIX)

* DESCRIPTION: To compute the effective element stiffness matrix
* and effective load vector for the hierarchical
* formulation using a 15 node element for the full
* form of the Navier-Stokes equations. The continuity
* equation is satisfied by a penalty formulation.
* The element uses 8 corner nodes, 6 midface nodes, and
* 1 node at the barycentre. The 15th node is condensed
* out at a postprocessing stage to eliminate the
* non-constant part of the linear pressure distribution.

* DATE: September 1985

* AUTHOR: Marc Dulude

* Passing In : COORD,IPROP,JPROP,LNODS,LNOMAX,LPOP,NDIM,NEL,NELZ,
* LVABZ,LNODZ,LTYPE,NODMAX,NITERA,VPROP,VFIX,NODFIX,
* NEWRHS,NDFMAX,NFIX

* Passing out-via disc: ELSTIF,ERHSU,ERHSV,ERHSW,LVABZ,LNODZ

*****
***** DICTIONARY OF VARIABLES *****
*****

*BFORX : BODY FORCE IN X DIRECTION
*BORY : BODY FORCE IN Y DIRECTION
*BFORZ : BODY FORCE IN Z DIRECTION
*CONVTN : CONVECTION TERMS IN THE MOMENTUM EQUATION
*COSINES : DIRECTION COSINES CALCULATED FOR A FACE NODE (GEOM)
*DERIV : VECTOR OF DERIVATIVES OF THE SHAPE FUNCTIONS
*DERSHX : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. X
*DERSHY : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Y
*DERSHZ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Z
*DERSXI : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. X FOR
* THE 'I'TH ROW
*DERSXJ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. X FOR
* THE 'J'TH COLUMN
*DERSYI : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Y FOR

* THE 'I'TH ROW
 *DERSYJ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Y FOR
 * THE 'J'TH COLUMN
 *DERSZI : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Z FOR
 * THE 'I'TH ROW
 *DERSZJ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Z FOR
 * THE 'J'TH COLUMN
 *DISDIV : ELEMENTAL DISCRETE DIVERGENCE
 *DYNVIS : MOLECULAR DYNAMIC VISCOSITY OF THE FLUID
 *ELXYZ : NODAL COORDINATES OF THE CURRENT ELEMENT
 *ERHS : RIGHT HAND SIDE VECTOR
 *GRAD : 3X3 ARRAY CONTAINING THE VELOCITY GRADIENTS AT THE GAUSS
 * USING THE PREVIOUS ITERATION VALUES.
 *GSCHM : FLAG TO DETERMINE GAUSSIAN THE INTEGRATION SCHEME NUMBER
 *IGAUS : COUNTER FOR THE GAUSS POINT
 *IPROP : NUMBER OF THE ITEM IN THE PROPERTY LIST
 *IROWU : COUNTER FOR THE ROW OF U COMPONENT OF VELOCITY
 *IROWV : COUNTER FOR THE ROW OF V COMPONENT OF VELOCITY
 *IROWW : COUNTER FOR THE ROW OF W COMPONENT OF VELOCITY
 *JCOLU : COUNTER FOR THE COLUMN OF U COMPONENT OF VELOCITY
 *JCOLV : COUNTER FOR THE COLUMN OF V COMPONENT OF VELOCITY
 *JCOLW : COUNTER FOR THE COLUMN OF W COMPONENT OF VELOCITY
 *JPROP : NUMBER OF THE PROPERTY LIST
 *KPOIN : VARIABLE FOR THE GLOBAL NODE NUMBERS
 *LCOUNT : COUNTER FOR THE TYPE OF THE FLOW, I.E. 0 FOR INVISCID FLOW
 * 1 FOR LAMINAR FLOW 2 FOR VISCOUS TURBULENT FLOW
 *LNODS : ARRAY RELATING LOCAL TO GLOBAL NODE NUMBERING
 *LVABZ : TOTAL NUMBER OF DEGREES OF FREEDOM IN AN ELEMENT
 *MIDNOD : NUMBER OF MIDSIDE NODES
 *MIDS : ARRAY OF THE MID-SIDE NODES IN THE ELEMENT
 *ND : A COUNTER FOR THE NODE (MIDSIDH)
 *NDIM : NUMBER OF DIMENSIONS OF THE PROBLEM
 *NDFMAX : MAXIMUM NO. OF DOF FOR A NODE
 *NEL : NUMBER OF THE CURRENT ELEMENT BEING ASSEMBLED
 *NELZ : TOTAL NUMBER OF ELEMENTS IN THE MESH
 *NEW RHS : NO. OF NEW RIGHT HAND SIDES EXPECTED IN A RE-SOLUTION.
 * REQUIRED FOR DIMENSIONING OF VFIX
 *NFI : NO. OF NODES WITH PRESCRIBED VALUES
 *NGAUS : NUMBER OF GAUSS POINTS
 *NITERA : COUNTER FOR THE NUMBER OF ITERATION
 *NODFIX : ARRAY OF NODE NUMBERS FOR PRESCRIBED VALUE NODES
 *PENALTY : PENALTY FACTOR INPUT BY USER
 *PENRHS : PENALTY TERM ADDED TO EACH OF THE RHS ENTRIES
 *POIN : ARRAY GIVING THE GLOBAL COORDINATES OF THE CURRENT GAUSS PT
 *PRES : VECTOR OF ELEMENTAL PRESSURES
 *RHOIN : DENSITY OF THE FLUID
 *SHAPE : VECTOR OF SHAPE FUNCTION VALUES
 *SHAPE : SHAPE FUNCTION VALUE AT THE CURRENT NODE

```

* TURVDX : DERIVATIVE OF THE TURBULENT VISCOSITY W.R.T. X
* TURVDY : DERIVATIVE OF THE TURBULENT VISCOSITY W.R.T. Y
* TURVDZ : DERIVATIVE OF THE TURBULENT VISCOSITY W.R.T. Z
* TURVIS : TURBULENT VISCOSITY
* TVISC : VALUE OF THE TURBULENT VISCOSITY AT A NODAL POINT
* UVELY : U COMPONENT OF VELOCITY
* VFIX : ARRAY OF PRESCRIBED VALUES
* VISLAM : KINEMATIC VISCOSITY OF THE FLUID (DYNVIS/RHOIN)
* VOLK : VOLUME OF THE ELEMENT (DETERMINANT OF JACOBIAN)
* VOLUME : DETERMINANT OF THE JACOBIAN AT THE GAUSS POINT WHICH IS THEN
*          MULTIPLIED BY THE GAUSS WEIGHT FOR THE GAUSSIAN INTEGRATION
* VVELY : V COMPONENT OF VELOCITY
* WVELY : W COMPONENT OF VELOCITY
* XGAUS : THE LOCAL X-COORDINATE AT THE GAUSS POINT
* XITA : THE LOCAL COORDINATES OF THE GAUSS POINTS
* YGAUS : THE LOCAL Y-COORDINATE AT THE GAUSS POINT
* ZGAUS : THE LOCAL Z-COORDINATE AT THE GAUSS POINT
*****

```

IMPLICIT LOGICAL (A-Z)

C

INTEGER

```

* IPROP, JPROP, LCOUNT, LNOMAX, LPOP, NDIM, NEL, NELZ, LVABZ, LNODZ, GSCHM,
* NODMAX, NITERA, INODZ, IGAUS, LTYPE, NGAUS, I, J, K, L, NODE,
* KPOIN, ICON, IROWU, IROWV, IROWW, IROWP, JCOLU, JCOLV, JCOLW, JCOLP, NSPL,
* JCON, IPCON, JPCON, LNODS(LNODZ, NELZ), NEWRHS, NDFMAX, NFIX, OERR,
* NODFIX(NFIX)

```

REAL*8

```

* VISLAM, RHOIN, BFORX, BFORV, BFORZ, VOLUME, UVELY, VVELY, WVELY, DISDIV,
* RNST1, RNST2, RNST3, RNST4, RNST5, RNST6, RNST7, RNST8, RNST9, DIFSON,
* CONVTN, TURVDX, TURVDY, TURVDZ, TURVIS, PRES(10), X(45), Y(45), Z(45),
* COORD(NODMAX, 7), ELSTIF(LVABZ, LVABZ), XITA(3), ERHS(33), JAC(3, 3),
* VPROP(IPROP, JPROP), SHAPE(15), WEIGP(15), ELXYZ(3, 15), JINV(3, 3),
* IGAUS(15), YGAUS(15), ZGAUS(15), VFIX(NEWRHS, NDFMAX, NFIX),
* DERIV(3, 15), DERSHX(15), DERSHY(15), DERSHZ(15), PENRHS, VOLK,
* TVISC(400), DERSXJ, DERSYJ, DERSZJ, DYNVIS, GRAD(3, 3), CONSUM,
* DERSXJ, DERSYJ, DERSZJ, SHAPI, SHAPJ, RADIUS(15), PENALTY, DIV(30)

```

C

SAVE PRES

*

*****

```

* Define the element properties required as originally read
* from the input file

```

*****

*

```

DYNVIS = VPROP(1, LPOP)
RHOIN = VPROP(2, LPOP)
BFORX = VPROP(3, LPOP)

```

```

BFORMY = VPROP(4,LPOP)
BFORMZ = VPROP(5,LPOP)
LCOUNT = VPROP(6,LPOP)
GSCHEN = VPROP(7,LPOP)
NSPL = VPROP(8,LPOP)
PENALTY= VPROP(9,LPOP)
VISLAM = DYNVIS/RHOIN

```

```

*
```

```

*****
```

```

* Assign the nodes and their coordinates for the current element.
* Modify the coordinates for the midside nodes if this is the first
* iteration.

```

```

*****
```

```

*
```

```

IF(NITERA.EQ.1) THEN
  CALL INIT(COORD,NODMAX,NEWRHS,NDFMAX,NFIX,VFIX,NODFIX,PRES,
1      NELZ)
  CALL MIDSIDH(COORD, LNOMAX, LNODS, LNODZ, LTYPE, NEL, NELZ,
1      ELXYZ,NDIM,NODMAX,6)
  CALL GEOM(ELXYZ, LNODZ, COORD, NODMAX, LNODS, NELZ, NEL, GSCHEN
1,      LVABZ, LNOMAX, LTYPE, X, Y, Z, DIV)
  WRITE(7,REC=NEL) ELXYZ, (X(I),Y(I),Z(I), I=1,45), (DIV(J),J=1,30)
  ELSE
  READ(7,REC=NEL) ELXYZ, (X(I),Y(I),Z(I), I=1,45), (DIV(J),J=1,30)
  ENDIF

```

```

*
```

```

*****
```

```

*
```

```

* Initialization of the right hand side vectors and the element
* stiffness matrix

```

```

*
```

```

*****
```

```

*
```

```

DO 10 I = 1, LVABZ
  ERHS(I) = 0.0
  DO 20 J = 1, LVABZ
    ELSTIF(I,J) = 0.0
20  CONTINUE
10  CONTINUE

```

```

*
```

```

*****
```

```

* Retrieve the barycentric node velocities using the appropriate X,
* Y, and Z values.

```

```

*****
```

```

*
```

```

KPOIN = IABS(LNODS(9,NEL))
DO 22 I = 4,6
  COORD(KPOIN,I) = 0.0

```

```

22  CONTINUE
    DO 25 INODZ = 1, LNODZ
      IF (INODZ.EQ.9) GO TO 25
      I = (3*INODZ) - 2
      J = I + 1
      K = I + 2
      NODE = IABS(LNODS(INODZ,NEL))
      IF (INODZ.LE.8) THEN
        COORD(KPOIN,4)=COORD(KPOIN,4)+(COORD(NODE,4)*X(I))
        COORD(KPOIN,5)=COORD(KPOIN,5)+(COORD(NODE,4)*Y(I))
        COORD(KPOIN,6)=COORD(KPOIN,6)+(COORD(NODE,4)*Z(I))
C
        COORD(KPOIN,4)=COORD(KPOIN,4)+(COORD(NODE,5)*X(J))
        COORD(KPOIN,5)=COORD(KPOIN,5)+(COORD(NODE,5)*Y(J))
        COORD(KPOIN,6)=COORD(KPOIN,6)+(COORD(NODE,5)*Z(J))
C
        COORD(KPOIN,4)=COORD(KPOIN,4)+(COORD(NODE,6)*X(K))
        COORD(KPOIN,5)=COORD(KPOIN,5)+(COORD(NODE,6)*Y(K))
        COORD(KPOIN,6)=COORD(KPOIN,6)+(COORD(NODE,6)*Z(K))
      ELSE
        COORD(KPOIN,4)=COORD(KPOIN,4)+(COORD(NODE,4)*X(I))
        COORD(KPOIN,5)=COORD(KPOIN,5)+(COORD(NODE,4)*Y(I))
        COORD(KPOIN,6)=COORD(KPOIN,6)+(COORD(NODE,4)*Z(I))
C
        COORD(KPOIN,4)=COORD(KPOIN,4)+(COORD(NODE,4)*X(J))
        COORD(KPOIN,5)=COORD(KPOIN,5)+(COORD(NODE,4)*Y(J))
        COORD(KPOIN,6)=COORD(KPOIN,6)+(COORD(NODE,4)*Z(J))
C
        COORD(KPOIN,4)=COORD(KPOIN,4)+(COORD(NODE,4)*X(K))
        COORD(KPOIN,5)=COORD(KPOIN,5)+(COORD(NODE,4)*Y(K))
        COORD(KPOIN,6)=COORD(KPOIN,6)+(COORD(NODE,4)*Z(K))
      ENDIF
25  CONTINUE
    COORD(KPOIN,4) = (COORD(KPOIN,4)/X(25))*(-1.0DOO)
    COORD(KPOIN,5) = (COORD(KPOIN,5)/X(25))*(-1.0DOO)
    COORD(KPOIN,6) = (COORD(KPOIN,6)/X(25))*(-1.0DOO)
    WRITE (6, '(A)') 'BARYCENTRE DOFS'
    WRITE(6,1030) (COORD(KPOIN,I),I=4,6)
1030 FORMAT(3(G15.5,2X))
*
*****
*   Update the elemental pressure from the previous iteration to this
*   one if this is not iteration number 1.
*****
*
DISDIV = 0.0
DO 96 NODE = 1, LNODZ
  IF (NODE.EQ.9) GOTO 96

```



```

      KPOIN = IABS(LNODS(NODE,NEL))
      I = (3*NODE) - 2
      J = I+1
      K = I+2
      IF (NODE.LE.8) THEN
        DISDIV = DISDIV + COORD(KPOIN,4)*DIV(I) + COORD(KPOIN,5)*
1          DIV(J) + COORD(KPOIN,6)*DIV(K)
      ELSE
        I = NODE + 15
        DISDIV = DISDIV + COORD(KPOIN,4)*DIV(I)
      ENDIF
96    CONTINUE
      IF (NITERA.GT.1) THEN
        PRES(NEL) = PRES(NEL) - (PENALTY*DISDIV)
      ENDIF
      WRITE(6, '(A)') 'ELEMENT DIVERGENCE'
      WRITE(6,1036) NEL,DISDIV
1036  FORMAT(I3,7X,G18.10)
      *
      *****
      *
      *   Enter the loop for the volume integration for the Usawa
      *   algorithm terms
      *
      *****
      *
      *****
      *   Evaluate the weights and distances for the Gaussian integration
      *****
      *
      CALL GAUS3D(GSCHEM,NGAUS,XGAUS,YGAUS,ZGAUS,WEIGP)
      DO 100   IGAUS = 1,NGAUS
      *
      *****
      *   Assign the values of the local coordinates from the current
      *   Gauss point
      *****
      *
      XITA (1) = XGAUS(IGAUS)
      XITA (2) = YGAUS(IGAUS)
      XITA (3) = ZGAUS(IGAUS)
      *
      *****
      *   Evaluate the shape functions, shape function derivatives, jacobian
      *   of transformation, jacobian inverse, etc. using the BRICKH
      *   subroutine. The subroutine calculates the above values using a
      *   hierarchical 15 node brick which uses linear interpolation for the
      *   corner nodes and parabolic interpolation for the barycentric and

```

* midface nodes.

*****

*

```

      CALL BRICKH(LNODZ, LVABZ, LNOMAX, NEL, NELZ, XITA, SHAPE,
1      ELXYZ, DERIV, VOLUME, LTYPE)
      VOLK = VOLUME
      VOLUME = VOLUME*WEIGP(IGAUS)

```

*

*****

* Assign the values of the shape functions and the derivatives
 * to the analogous terms for the current Gauss point.

*****

*

```

      DO 30 INODZ = 1, LNODZ
        DERSHX(INODZ) = DERIV (1, INODZ)
        DERSHY(INODZ) = DERIV (2, INODZ)
        DERSHZ(INODZ) = DERIV (3, INODZ)
30    CONTINUE

```

*

*****

* Initialize the three velocity components (i.e., u, v, and w), as
 * well as the coefficient of eddy viscosity and its derivatives
 * for the turbulent flow case.

*****

*

```

      UVELY = 0.0
      VVELY = 0.0
      WVELY = 0.0
      TURVIS = 0.0
      TURVDX = 0.0
      TURVDY = 0.0
      TURVDZ = 0.0
      DO 6 ICON = 1, 3
        DO 6 JCON = 1, 3
          GRAD(ICON, JCON) = 0.0
6      CONTINUE
5      CONTINUE

```

*

*****

* Evaluate the velocities and velocity gradients at the Gauss point
 * using the previous nodal velocities for all kinds of flows.
 * The summation over the midface nodes must take into account the
 * direction cosine from the outward normal (midface dof) and the
 * appropriate axis. These are stored in COORD(5,6,7) for x,y,z
 * respectively.

*

* If INODZ <= 9 then the node is a barycentric or corner node. If
 * INODZ is greater than 9, the node is a midface node and the dof is

* the velocity normal to the face.

*****

*

DO 35 INODZ = 1, LNODZ

KPOIN = IABS(LNODS (INODZ, NEL))

IF (INODZ, LE, 9) THEN

UVELY = UVELY + (COORD(KPOIN, 4) * SHAPE(INODZ))

VVELY = VVELY + (COORD(KPOIN, 5) * SHAPE(INODZ))

WVELY = WVELY + (COORD(KPOIN, 6) * SHAPE(INODZ))

*

*****

* Now the velocity gradients.

*****

*

DO 36 ICON = 4, 6

GRAD(ICON-3, 1) = GRAD(ICON-3, 1) +

1 (COORD(KPOIN, ICON) * DERSHX(INODZ))

GRAD(ICON-3, 2) = GRAD(ICON-3, 2) +

1 (COORD(KPOIN, ICON) * DERSHY(INODZ))

GRAD(ICON-3, 3) = GRAD(ICON-3, 3) +

1 (COORD(KPOIN, ICON) * DERSHZ(INODZ))

36 CONTINUE

ELSE

*

*****

* Now the contributions to the velocities & gradients from the face nodes

*****

*

UVELY = UVELY + (COORD(KPOIN, 4) * COORD(KPOIN, 5) * SHAPE(INODZ))

VVELY = VVELY + (COORD(KPOIN, 4) * COORD(KPOIN, 6) * SHAPE(INODZ))

WVELY = WVELY + (COORD(KPOIN, 4) * COORD(KPOIN, 7) * SHAPE(INODZ))

DO 37 ICON = 5, 7

GRAD(ICON-4, 1) = GRAD(ICON-4, 1) + (COORD(KPOIN, 4) *

1 COORD(KPOIN, ICON) * DERSHX(INODZ))

GRAD(ICON-4, 2) = GRAD(ICON-4, 2) + (COORD(KPOIN, 4) *

1 COORD(KPOIN, ICON) * DERSHY(INODZ))

GRAD(ICON-4, 3) = GRAD(ICON-4, 3) + (COORD(KPOIN, 4) *

1 COORD(KPOIN, ICON) * DERSHZ(INODZ))

37 CONTINUE

ENDIF

35 CONTINUE

*

*****

* Check for the type of the flow. In case of the laminar and
* inviscid flows, do not add the eddy viscosity contributions

*

* If LCOUNT = 0 ==> inviscid

* = 1 ==> laminar

```

*           >= 2 ==> turbulent
*****
*
*           IF (LCOUNT.GE.2) THEN
*
*****
*   Call a subroutine to compute the eddy viscosity values at the
*   nodal points of the element.
*****
*
*           IF (LCOUNT.EQ.2) THEN
*               CALL LAUFTURB(COORD, LNODS, LNODZ, NEL, TVISC, NELZ, NODMAX,
1               RADIUS, NITERA)
*           ENDIF
*
*****
*   Evaluate the turbulent viscosity and its derivatives at the
*   Gauss point.
*****
*
*           DO 40 INODZ = 1, LNODZ
*               TURVIS = TURVIS + (TVISC(INODZ) * SHAPE(INODZ))
*               TURVDX = TURVDX + (TVISC(INODZ) * DERSHX(INODZ))
*               TURVDY = TURVDY + (TVISC(INODZ) * DERSHY(INODZ))
*               TURVDZ = TURVDZ + (TVISC(INODZ) * DERSHZ(INODZ))
40          CONTINUE
*           ENDIF
*
*****
*   Velocities and velocity gradients as well as the eddy viscosity
*   and its derivatives (for the turbulent case only) have been
*   calculated at the current Gauss point.
*****
*
*****
*   Start the looping over the element for the left hand side terms
*
*****
*
*           DO 70 ICON=1, LNODZ
*
*****
*   Find the row and column in the stiffness matrix into which the
*   calculated quantities sum. There are three DOFs for barycentric and
*   each of the corner nodes and one for the midface nodes. Rows
*   correspond to the x, y, and z momentum equations (numbered 1, 2, and
*   3 respectively) and the column numbers designate whether to multiply

```

* into the u,v,or w terms (corresponding to 1,2,and 3 respectively).

*****

*

```

      IF(ICON.LE.9) THEN
        IROWU =(3*ICON) - 2
        IROWV = IROWU+1
        IROWW = IROWU+2
      ELSE
        IROWU = ICON + 18
        IROWV = IROWU
        IROWW = IROWU
      ENDIF

```

*

*****

* Evaluate the 'I'th shape function and its derivatives for the
* 'I'th row.

*****

*

```

      SHAPI = SHAPE(ICON)
      DERSXI = DERSHI(ICON)
      DERSYI = DERSHY(ICON)
      DERSZI = DERSHZ(ICON)

```

*

*****

* Counter for the columns as described above

*****

*

```

      DO 80 JCON = 1, LNODZ
        IF(JCON.LE.9) THEN
          JCOLU = (3*JCON) - 2
          JCOLV = JCOLU+1
          JCOLW = JCOLU+2
        ELSE
          JCOLU = JCON + 18
          JCOLV = JCOLU
          JCOLW = JCOLU
        ENDIF

```

*

*****

* Evaluate the 'J'th shape function derivatives for the
* 'J'th column.

*****

*

```

      SHAPJ = SHAPE(JCON)
      DERSXJ = DERSHI(JCON)
      DERSYJ = DERSHY(JCON)
      DERSZJ = DERSHZ(JCON)

```

*

*****

* Evaluate the contributions from each of the two types of convection
 * terms and the diffusion terms. Only include the diffusion terms
 * if this is a viscous flow (ie. LCOUNT=0).

*****

*

```

CONVTN = (UVELY*DERSXJ+VVELY*DERSYJ+WVELY*DERSZJ)
1      *SHAPI*VOLUME
ELSTIF(IROWU,JCOLU) = ELSTIF(IROWU,JCOLU)+CONVTN
ELSTIF(IROWV,JCOLV) = ELSTIF(IROWV,JCOLV)+CONVTN
ELSTIF(IROWW,JCOLW) = ELSTIF(IROWW,JCOLW)+CONVTN

```

*

*****

* Now the convection terms resulting from the Usawa algorithm

*****

*

```

1      ELSTIF(IROWU,JCOLU)=ELSTIF(IROWU,JCOLU)+(SHAPI*SHAPJ*
      GRAD(1,1))*VOLUME
1      ELSTIF(IROWU,JCOLV)=ELSTIF(IROWU,JCOLV)+(SHAPI*SHAPJ*
      GRAD(2,2))*VOLUME
1      ELSTIF(IROWU,JCOLW)=ELSTIF(IROWU,JCOLW)+(SHAPI*SHAPJ*
      GRAD(1,3))*VOLUME
1      ELSTIF(IROWV,JCOLU)=ELSTIF(IROWV,JCOLU)+(SHAPI*SHAPJ*
      GRAD(2,1))*VOLUME
1      ELSTIF(IROWV,JCOLV)=ELSTIF(IROWV,JCOLV)+(SHAPI*SHAPJ*
      GRAD(2,2))*VOLUME
1      ELSTIF(IROWV,JCOLW)=ELSTIF(IROWV,JCOLW)+(SHAPI*SHAPJ*
      GRAD(2,3))*VOLUME
1      ELSTIF(IROWW,JCOLU)=ELSTIF(IROWW,JCOLU)+(SHAPI*SHAPJ*
      GRAD(3,1))*VOLUME
1      ELSTIF(IROWW,JCOLV)=ELSTIF(IROWW,JCOLV)+(SHAPI*SHAPJ*
      GRAD(3,2))*VOLUME
1      ELSTIF(IROWW,JCOLW)=ELSTIF(IROWW,JCOLW)+(SHAPI*SHAPJ*
      GRAD(3,3))*VOLUME

```

*

*****

* Now the diffusion terms

*****

*

```

IF(LCOUNT.NE.0) THEN
1      ELSTIF(IROWU,JCOLU)=ELSTIF(IROWU,JCOLU)+(4*DERSXI*DERSXJ
      +DERSYI*DERSYJ+DERSXI*DERSYJ+DERSZI*DERSZJ+DERSXI*DERSZJ)
2      *VOLUME*VISLAM/2.
1      ELSTIF(IROWU,JCOLV)=ELSTIF(IROWU,JCOLV)+(DERSYI*DERSXJ
      +DERSXI*DERSXJ)*VOLUME*VISLAM/2.
1      ELSTIF(IROWU,JCOLW)=ELSTIF(IROWU,JCOLW)+(DERSZI*DERSXJ
      +DERSXI*DERSXJ)*VOLUME*VISLAM/2.

```

C

```

      ELSTIF(IROWV,JCOLU)=ELSTIF(IROWV,JCOLU)+(DERSXI*DERSYJ
1          +DERSYI*DERSYJ)*VOLUME*VISLAM/2.
      ELSTIF(IROWV,JCOLV)=ELSTIF(IROWV,JCOLV)+(4*DERSYI*DERSYJ
1          +DERSXI*DERSXJ+DERSZI*DERSZJ+DERSYI*DERSXJ+DERSYI*DERSZJ)
2          *VOLUME*VISLAM/2.
      ELSTIF(IROWV,JCOLW)=ELSTIF(IROWV,JCOLW)+(DERSZI*DERSYJ
1          +DERSYI*DERSYJ)*VOLUME*VISLAM/2.
C
      ELSTIF(IROWW,JCOLU)=ELSTIF(IROWW,JCOLU)+(DERSXI*DERSZJ
1          +DERSZI*DERSZJ)*VOLUME*VISLAM/2.
      ELSTIF(IROWW,JCOLV)=ELSTIF(IROWW,JCOLV)+(DERSYI*DERSZJ
1          +DERSZI*DERSZJ)*VOLUME*VISLAM/2.
      ELSTIF(IROWW,JCOLW)=ELSTIF(IROWW,JCOLW)+(4*DERSZI*DERSZJ
1          +DERSXI*DERSXJ+DERSYI*DERSYJ+DERSZI*DERSXJ+DERSZI*DERSYJ)
2          *VOLUME*VISLAM/2.
      ENDIF

```

```

*
*****
*   If the problem is turbulent, the extra contributions are
*   calculated and include in the [K] matrix here.
*****

```

```

      IF(LCOUNT.GE.2) THEN
      ENDIF

```

```

*
*****
*   End the loop for the 'J'th column
*****

```

```

80      CONTINUE

```

```

*****
*   Evaluate the right hand side terms.
*****

```

```

*****
*   From Usawa's algorithm, the RHS terms consist of diffusion,
*   convection, pressure gradient, body force, and penalty terms.
*****

```

```

*
*****
*   Diffusion
*****

```

```

1      ERHS(IROWU)=ERHS(IROWU) + (4*GRAD(1,1)*DERSXI + GRAD(1,2)*
      (DERSXI+DERSYI) + GRAD(1,3)*(DERSXI+DERSZI) + GRAD(2,1)*

```

```

2      (DERSXI+DERSYI) + GRAD(3,1)*(DERSXI+DERSZI))*VOLUME
      ERHS(IROWV)=ERHS(IROWV) + (GRAD(2,1)*(DERSYI+DERSXI) + 4*
1      GRAD(2,2)*DERSYI + GRAD(2,3)*(DERSYI+DERSZI) + GRAD(1,2)*
2      (DERSYI+DERSXI) + GRAD(3,2)*(DERSYI+DERSZI))*VOLUME
      ERHS(IROWW)=ERHS(IROWW) + (GRAD(3,1)*(DERSZI+DERSXI) +
1      GRAD(3,2)*(DERSZI+DERSYI) + 4*GRAD(3,3)*DERSZI + GRAD(1,3)
2      *(DERSZI+DERSXI) + GRAD(2,3)*(DERSZI+DERSYI))*VOLUME
*
*****
*   Convection
*****
*
      ERHS(IROWU)=ERHS(IROWU) + (SHAPI*(UVELY*GRAD(1,1) + VVELY*
1      GRAD(1,2) + WVELY*GRAD(1,3))*VOLUME
      ERHS(IROWV)=ERHS(IROWV) + (SHAPI*(UVELY*GRAD(2,1) + VVELY*
1      GRAD(2,2) + WVELY*GRAD(2,3))*VOLUME
      ERHS(IROWW)=ERHS(IROWW) + (SHAPI*(UVELY*GRAD(3,1) + VVELY*
1      GRAD(3,2) + WVELY*GRAD(3,3))*VOLUME
*
*****
*   Body force
*****
*
      ERHS(IROWU) = ERHS(IROWU)-SHAPI*VOLUME*BFORX
      ERHS(IROWV) = ERHS(IROWV)-SHAPI*VOLUME*BFORY
      ERHS(IROWW) = ERHS(IROWW)-SHAPI*VOLUME*BFORZ
*
*****
*   End the loop for the 'I'th row
*****
*
70  CONTINUE
*
*****
*   End the loop for the Gauss point integration
*****
*
100 CONTINUE
*
*****
*   Postprocess the rows & columns of the [K] matrix to remove the
*   internal node dofs and the nonconstant portion of the elemental
*   pressure.
*****
*
*
*****
*   Do column summation

```



```

*****
*
DO 110 I = 1, LVABZ
*
*****
*   We don't want to change the barycentric dofs
*****
*
IF(I.EQ.25.OR.I.EQ.26.OR.I.EQ.27) GOTO 110
L = 25
DO 112 J = 1, 24
    ELSTIF(I,J) = ELSTIF(I,J) - (1.0/X(25))*(X(J)*ELSTIF(I,25) +
1   Y(J)*ELSTIF(I,26) + Z(J)*ELSTIF(I,27))
112 CONTINUE
*
*****
*   K is used to keep track of the X,Y, and Z values for the rows of the
*   element stiffness matrix. L does the same for the columns. These
*   are required since there are 45 of the X,Y,Z values and only 33 dof
*   in the K matrix (hierarchical formulation).
*****
*
DO 115 J = 28, 33
    L = L+3
    DO 117 JCON = L, L+2
        ELSTIF(I,J) = ELSTIF(I,J) - (1.0/X(25))*(X(JCON)*
1   ELSTIF(I,25) + Y(JCON)*ELSTIF(I,26) + Z(JCON)*
2   ELSTIF(I,27))
117 CONTINUE
115 CONTINUE
110 CONTINUE
*
*****
*   Do row summation
*****
*
DO 120 J = 1, LVABZ
    IF(J.EQ.25.OR.J.EQ.26.OR.J.EQ.27) GOTO 120
    K = 25
    DO 122 I = 1, 24
        ELSTIF(I,J) = ELSTIF(I,J) - (1.0/X(25))*(X(I)*ELSTIF(25,J) +
1   Y(I)*ELSTIF(26,J) + Z(I)*ELSTIF(27,J))
122 CONTINUE
    DO 125 I = 28, 33
        K = K+3
        DO 127 ICON = K, K+2
            ELSTIF(I,J) = ELSTIF(I,J) - (1.0/X(25))*(X(ICON)*
1   ELSTIF(25,J) + Y(ICON)*ELSTIF(26,J) + Z(ICON)*

```

```

      2      ELSTIF(27,J))
127      CONTINUE
125      CONTINUE
120      CONTINUE
*
*****
*   Now, remove the rows and columns of the barycentric dofs.
*   First, remove the rows of the elemental stiffness matrix and the
*   RHS vector.
*****
*
      DO 140 ICON = 28,33
        DO 145 JCON = 1,LVABZ
          ELSTIF(ICON-3,JCON) = ELSTIF(ICON,JCON)
145      CONTINUE
          ERHS(ICON-3) = ERHS(ICON)
140      CONTINUE
*
*****
*   Now the columns of ELSTIF
*****
*
      DO 150 ICON = 1,LVABZ-3
        DO 155 JCON = 28,33
          ELSTIF(ICON,JCON-3) = ELSTIF(ICON,JCON)
155      CONTINUE
150      CONTINUE
*
*****
*   Since we are essentially left with a 14 node brick with 30 dof, reduce
*   the LVABZ value from 33 to 30 to tell the solver.
*****
*
      LVABZ = LVABZ - 3.
*
*****
*   Now add on the penalty terms into the [K] MATRIX and [RHS] vector
*   The pressure gradient terms are also added to the [RHS] vector.
*****
*
      DO 900 I = 1,LVABZ
        DO 910 J = 1,LVABZ
          ELSTIF(I,J) = ELSTIF(I,J) + PENALTY*VOLK*DIV(I)*DIV(J)
910      CONTINUE
          ERHS(I) = ERHS(I) + PENALTY*VOLK*DISDIV*DIV(I)
          ERHS(I) = ERHS(I) - (VOLK*PRES(NEL)*DIV(I))/RHOIN
900      CONTINUE
*

```

*****

* Check NSPL to determine if any special qualities are required
* of the element. None available yet.

*****

*

WRITE(UNIT=1,REC=NEL) NSPL,LVABZ,DISDIV.

1 ((ELSTIF(I,J),I=1,LVABZ),J=1,LVABZ)

WRITE(UNIT=3,REC=NEL+NELZ) LNODZ,LVABZ,(ERHS(I),I=1,LVABZ)

1 PRES(NEL)

RETURN

END

*

*****

*

SUBROUTINE INIT(COORD,NODMAX,NEWRHS,NDFMAX,NFIX,VFIX,NODFIX,PRES,
1 NELZ)

*

*****

* Subroutine to initialize the degrees of freedom of each node to the
* value prescribed in the input file prescribed dof list. Default is 0.0

*****

*

INTEGER NODMAX,I,J,NEWRHS,NFIX,NODFIX(NFIX),NODE,NDFMAX,NELZ

C

REAL*8 COORD(NODMAX,7),VFIX(NEWRHS,NDFMAX,NFIX),PRES(10)

C

*

*****

* Initialize all velocity entries to zero since that is the default

*****

*

DO 5 I = 1,NELZ

PRES(I) = 0.0

5

CONTINUE

DO 10 I=1,NODMAX

COORD(I,4)=0.0

COORD(I,5)=0.0

COORD(I,6)=0.0

COORD(I,7)=0.0

10

CONTINUE

*

*****

* Assign COORD the prescribed dof value where given

*****

*

DO 20 I=1,NFIX

NODE = IABS(NODFIX(I))

COORD(NODE,4) = VFIX(1,1,I)

```

      COORD(NODE,5) = VFIX(1,2,I)
      COORD(NODE,6) = VFIX(1,3,I)
20  CONTINUE
      RETURN
      END
*
*****
*
      SUBROUTINE MIDSIDH(COORD, LNOMAX, LNODS, LNODZ, LTYPE, NEL, NELZ,
1      ELXYZ, NDIM, NODMAX, MIDNOD)
*
*  PURPOSE :   To modify the coordinates of the 15 node brick
*              element for the hierarchical formulation. It modifies
*              the representation of the midface nodes to return
*              the hierarchical values as well as the direction cosines
*              by using the cross product of the Jacobian evaluated at
*              the midface position.
*
      IMPLICIT LOGICAL (A-Z)
C
      INTEGER
1      I, J, LNODZ, LNOMAX, LTYPE, MIDNOD, MIDS15(5,6), ND, NDIM,
2      NEL, NELZ, NOD, NODE, NODMAX, LNODS(LNOMAX, NELZ)
C
      REAL*8
1      COORD(NODMAX,7), ELXYZ(NDIM, LNODZ)
*
*****
*  List of connectivity data for the midface nodes in the 15 node
*  brick. There are 8 corner nodes, 1 barycentric node, and 6 midface
*  nodes.
*****
      DATA MIDS15/10,1,4,8,5, 11,2,3,7,6, 12,1,5,6,2,
1      13,4,8,7,3, 14,1,2,3,4, 15,5,6,7,8/
*
*****
*  Collect the nodal coordinates for the element in ELXYZ(NDIM, LNODZ)
*****
*
      DO 20 NOD=1, LNODZ
          NODE = IABS(LNODS(NOD, NEL))
          DO 10 ND = 1, NDIM
              ELXYZ(ND, NOD) = COORD(NODE, ND)
10          CONTINUE
20      CONTINUE
*
*****

```

* Calculate the coordinates for the faces and barycentre of the 15 node
 * brick. The dof for the midface node is the difference from the average
 * of the four bounding corner nodes. The dof for the barycentric node is
 * the difference from the sum of the averages of the corner and midface
 * nodes. In essence:

* $dof(9) = dof(9)[absolute] - (1/8) * (\text{sum of } dof(1) \text{ to } dof(8)) -$
 * $(1/2) * (\text{sum of } dof(10) \text{ to } dof(15))$

* It modifies the coordinates if specified and makes them zero if not
 * specified.

*****

```

*
DO 40 I = 1,MIDNOD
  NODE = IABS (LNODS(MIDS15(1,I),NEL))
  DO 30 ND = 1,NDIM
    IF (ELXYZ(ND,MIDS15(1,I)) .EQ. 5.5E+55) THEN
      ELXYZ(ND,MIDS15(1,I)) = (ELXYZ(ND,MIDS15(2,I)) +
1      ELXYZ(ND,MIDS15(3,I)) + ELXYZ(ND,MIDS15(4,I)) +
2      ELXYZ(ND,MIDS15(5,I))) * 0.25
    ELSE
      ELXYZ(ND,MIDS15(1,I)) = ELXYZ(ND,MIDS15(1,I)) -
1      ((ELXYZ(ND,MIDS15(2,I)) + ELXYZ(ND,MIDS15(3,I))
2      + ELXYZ(ND,MIDS15(4,I)) + ELXYZ(ND,MIDS15(5,I))) * 0.25)
    ENDIF
  30 CONTINUE
40 CONTINUE

```

*

*****

* Now get the hierarchical value for the barycentric node from the
 * differences of the other nodes.

*****

*

```

DO 45 ND = 1,3
  ELXYZ(ND,9) = ELXYZ(ND,9) - ((ELXYZ(ND,1)+ELXYZ(ND,2)+
1  ELXYZ(ND,3)+ELXYZ(ND,4)+ELXYZ(ND,5)+ELXYZ(ND,6)+
2  ELXYZ(ND,7)+ELXYZ(ND,8)) * 0.125) - ((ELXYZ(ND,10)+
3  ELXYZ(ND,11)+ELXYZ(ND,12)+ELXYZ(ND,13)+ELXYZ(ND,14)+
4  ELXYZ(ND,15)) * 0.5)
45 CONTINUE

```

*

*****

* Place the modified coordinates back in the COORD array

*****

*

```

DO 50 I = 1,MIDNOD
  NODE = IABS(LNODS(MIDS15(1,I),NEL))
  DO 55 ND = 1,NDIM

```

```

      COORD(NODE,ND) = ELXYZ(ND,MIDS15(1,I))
55      CONTINUE
50      CONTINUE
*
*****
*   Check to ensure all coordinates have been assigned.  If not,
*   output an error message and stop.
*****
*
      DO 140 NOD=1, LNODZ
        NODE = IABS(LNODS(NOD,NEL))
        DO 130 ND = 1,NDIM
          IF (ELXYZ(ND,NOD).EQ .5.5E+55) THEN
            PRINT*, 'UNASSIGNED COORDINATE IN MIDSID'
            STOP
          ENDIF
130      CONTINUE
140      CONTINUE
C9000  FORMAT(' NODE',6X,'X',12X,'Y',12X,'Z'./)
C9010  FORMAT(1X,16,3F13.7)
      RETURN
      END

*
*****
*
      SUBROUTINE GEOM(ELXYZ, LNODZ, COORD, NODMAX, LNODS, NELZ, NEL, GSCHM
1.      LVABZ, LNOMAX, LTYPE, X, Y, Z, DIV)
C
      INTEGER
1  LNODZ, I, INOD, ITOPOLG, TEST, AXIS, K, L, ND, MIDS15(5,6), NODMAX
2, LNODS(LNODZ, NELZ), NELZ, NEL, NODE, IGAUS, NGAUS, GSCHM, LVABZ
3, LNOMAX, LTYPE, KPOIN
C
      REAL*8
1  XITA(3), J(3,3), DERIV(3,15), SHAPE(15), NORMAL(3)
2, ELXYZ(3,15), TOL, NEGTOL, LENGTH, COSINES(3), COORD(NODMAX,7)
3, XGAUS(15), YGAUS(15), ZGAUS(15), X(45), Y(45), Z(45), WEIGP(15)
4, VOLUME, VOLK, DIV(30)
C
      DATA TOL/1E-07/
*
*****
*   List of connectivity data for the midface nodes in the 15 node
*   brick. There are 8 corner nodes, 1 barycentric node, and 6
*   midface nodes.
*****
*
      DATA MIDS15/10,1,4,8,5, 11,2,3,7,6, 12,1,5,6,2,

```

```

1          13.4.8.7.3.  14.1.2.3.4.  15.5.6.7.8/
C
  NEGTOI = -1.0*TOL
*
*****
*   Determine the coordinates of the midface nodes based on the local
*   coordinates given in SUBROUTINE MIDISDH (nodes 10 to 15)
*****
*
  DO 1 I = 1,30
    DIV(I) = 0.0
1    CONTINUE
  DO 5 INOD = 1,6
    ITOPOLG = INOD + 9
    IF (INOD.EQ.1.OR.INOD.EQ.2) THEN
      XITA(1) = 1.0
      XITA(2) = 0.0
      XITA(3) = 0.0
    ELSEIF (INOD.EQ.3.OR.INOD.EQ.4) THEN
      XITA(1) = 0.0
      XITA(2) = 1.0
      XITA(3) = 0.0
    ELSEIF (INOD.EQ.5.OR.INOD.EQ.6) THEN
      XITA(1) = 0.0
      XITA(2) = 0.0
      XITA(3) = 1.0
    ENDIF
    TEST = (INOD/2) * 2
    IF (TEST.LT.INOD) THEN
      DO 10 I = 1,3
        XITA(I) = -1.0 * XITA(I)
10     CONTINUE
    ENDIF
*
*****
*   Get the Jacobian matrix by finding the values of the shape function
*   derivatives at the coordinate calculated above.
*****
*
    CALL SFRH15 (XITA,SHAPE,DERIV)
    CALL JACOB3D (DERIV,ELXYZ,J,LNODZ)
*
*****
*   Calculate the cross product vector quantities
*****
*
    AXIS = (INOD+1) / 2
    IF (AXIS.EQ.1) THEN

```

```

        NORMAL (1) = J(2,2)*J(3,3) - J(3,2)*J(2,3)
        NORMAL (2) = J(3,1)*J(2,3) - J(2,1)*J(3,3)
        NORMAL (3) = J(2,1)*J(3,2) - J(3,1)*J(2,2)
    ELSEIF (AXIS.EQ.2) THEN
        NORMAL (1) = J(3,2)*J(1,3) - J(1,2)*J(3,3)
        NORMAL (2) = J(1,1)*J(3,3) - J(3,1)*J(1,3)
        NORMAL (3) = J(3,1)*J(1,2) - J(1,1)*J(3,2)
    ELSEIF (AXIS.EQ.3) THEN
        NORMAL (1) = J(1,2)*J(2,3) - J(2,2)*J(1,3)
        NORMAL (2) = J(2,1)*J(1,3) - J(1,1)*J(2,3)
        NORMAL (3) = J(1,1)*J(2,2) - J(2,1)*J(1,2)
    ENDIF

*
*****
*   Ensure homogeneity of the normal vector directions
*****
*
    IF (NORMAL(1).GT.TOL.OR.NORMAL(1).LT.NEGTOL) THEN
        IF (NORMAL(1).LT.0.0) NORMAL(1) = -1.0 * NORMAL(1)
    ELSEIF (NORMAL(2).GT.TOL.OR.NORMAL(2).LT.NEGTOL) THEN
        IF (NORMAL(2).LT.0.0) NORMAL(2) = -1.0 * NORMAL(2)
    ELSEIF (NORMAL(3).GT.TOL.OR.NORMAL(3).LT.NEGTOL) THEN
        IF (NORMAL(3).LT.0.0) NORMAL(3) = -1.0 * NORMAL(3)
    ENDIF

*
*****
*   Calculate direction cosines
*****
*
    LENGTH = SQRT(NORMAL(1)**2+NORMAL(2)**2+NORMAL(3)**2)
    COSINES (1) = NORMAL(1) /LENGTH
    COSINES (2) = NORMAL(2) /LENGTH
    COSINES (3) = NORMAL(3) /LENGTH

*
*****
*   Put the direction cosines into the COORD array in positions
*   COORD(NODE,5),COORD(NODE,6),COORD(NODE,7) for the x-axis cosine,
*   y-axis cosine, and z-axis cosine respectively.
*****
*
    NODE = IABS(LNODS(ITOPOLG,NEL))
    COORD(NODE,5) = COSINES(1)
    COORD(NODE,6) = COSINES(2)
    COORD(NODE,7) = COSINES(3)
5  CONTINUE

*
*****
*   Calculate the X,Y, and Z values required to remove the internal

```


* velocity dofs and the non-constant portion of the elemental pressure.

*****

*

DO 15 I = 1,45

X(I) = 0.0

Y(I) = 0.0

Z(I) = 0.0

15 CONTINUE

CALL GAUS3D(GSCHEM,NGAUS,XGAUS,YGAUS,ZGAUS,WEIGP)

DO 20 IGAUS = 1,NGAUS

XITA(1) = XGAUS(IGAUS)

XITA(2) = YGAUS(IGAUS)

XITA(3) = ZGAUS(IGAUS)

CALL BRICKH(LNODZ,LVABZ,LNOMAX,NEL,NELZ,XITA,SHAPE,ELXYZ,DERIV,

1 VOLUME,LTYPE)

VOLK = VOLUME

VOLUME = VOLUME * WEIGP(IGAUS)

DO 25 NODE = 1,LNODZ

I = (3 * NODE) - 2

K = I + 1

L = I + 2

*

*****

* The X,Y,Z values are simply the integral of the Gauss point coordinate
 * (x,y,z respectively) times the divergence of the basis functions. The
 * face nodes must be treated in a special way since these basis functions
 * are essentially bubble functions multiplied by the appropriate
 * direction cosine.

*****

*

IF (NODE.LE.9) THEN

X(I) = X(I) + XITA(1)*VOLUME*DERIV(1,NODE)

Y(I) = Y(I) + XITA(2)*VOLUME*DERIV(1,NODE)

Z(I) = Z(I) + XITA(3)*VOLUME*DERIV(1,NODE)

C

X(K) = X(K) + XITA(1)*VOLUME*DERIV(2,NODE)

Y(K) = Y(K) + XITA(2)*VOLUME*DERIV(2,NODE)

Z(K) = Z(K) + XITA(3)*VOLUME*DERIV(2,NODE)

C

X(L) = X(L) + XITA(1)*VOLUME*DERIV(3,NODE)

Y(L) = Y(L) + XITA(2)*VOLUME*DERIV(3,NODE)

Z(L) = Z(L) + XITA(3)*VOLUME*DERIV(3,NODE)

ELSE

KPOIN = IABS(LNODS(NODE,NEL))

X(I) = X(I) + XITA(1)*VOLUME*DERIV(1,NODE)*COORD(KPOIN,6)

Y(I) = Y(I) + XITA(2)*VOLUME*DERIV(1,NODE)*COORD(KPOIN,6)

Z(I) = Z(I) + XITA(3)*VOLUME*DERIV(1,NODE)*COORD(KPOIN,6)

C

```

      X(K) = X(K) + XITA(1)*VOLUME*DERIV(2,NODE)*COORD(KPOIN,6)
      Y(K) = Y(K) + XITA(2)*VOLUME*DERIV(2,NODE)*COORD(KPOIN,6)
      Z(K) = Z(K) + XITA(3)*VOLUME*DERIV(2,NODE)*COORD(KPOIN,6)
C
      X(L) = X(L) + XITA(1)*VOLUME*DERIV(3,NODE)*COORD(KPOIN,7)
      Y(L) = Y(L) + XITA(2)*VOLUME*DERIV(3,NODE)*COORD(KPOIN,7)
      Z(L) = Z(L) + XITA(3)*VOLUME*DERIV(3,NODE)*COORD(KPOIN,7)
      ENDIF
      IF (NODE.EQ.9) GOTO 25
      IF (NODE.LE.8) THEN
        DIV(I) = DIV(I) + VOLUME*DERIV(1,NODE)
        DIV(K) = DIV(K) + VOLUME*DERIV(2,NODE)
        DIV(L) = DIV(L) + VOLUME*DERIV(3,NODE)
      ELSE
        KPOIN = IABS(LNODS(NODE,NEL))
        I = NODE + 15
        DIV(I) = DIV(I) + VOLUME*(DERIV(1,NODE)*COORD(KPOIN,5) +
1      DERIV(2,NODE)*COORD(KPOIN,6) + DERIV(3,NODE)*
1      COORD(KPOIN,7))
      ENDIF
25    CONTINUE
20    CONTINUE
      DO 35 I = 1,30
        DIV(I) = DIV(I)/VOLK
35    CONTINUE
      RETURN
      END
*
*****
*
      SUBROUTINE BRICKH(LNODZ,LVABZ,LNOMAX,NEL,NELZ,XITA,
1      SHAPE,ELXYZ,DERIV,VOLUME,LTYPE)
*
*****
*   This subroutine computes the Jacobian matrix of the transformation
*   as well as its determinant (which is the volume of the
*   transformation), the inverse of this matrix, and the shape
*   functions for this element (15 node brick) in both local and
*   global coordinates at the current Gauss point.
*****
*
      IMPLICIT LOGICAL (A-Z)
C
      INTEGER
1    LNODZ, LNOMAX, LVABZ, NEL, NELZ, OERR, LTYPE
C
      REAL*8
1    DERIV(3,LNODZ), ELXYZ(3,LNODZ), J(3,3), JINV(3,3),

```

```
2 POIN(3), SHAPE(LNODZ), VOLUME,XITA(3)
```

```
*
*****
*   Initialize the error flag
*****
*
      OERR=0
*
*****
*   The subroutine SFRH15 will get the shape functions and the shape
*   function derivatives with respect to all local coordinates at
*   the point XITA in local coordinates.
*****
*
      IF (LTYPE.EQ.144) CALL SFRH15(XITA,SHAPE,DERIV)
*
*****
*   Now the Jacobian, which is the product of ELXYZ and the shape
*   function derivatives (used to transform from local to global
*   coordinates), is found.
*****
*
      CALL JACOB3D(DERIV,ELXYZ,J,LNODZ)
*
*****
*   The Jacobian, is inverted and its determinant, which is the volume
*   of the transformation, is found.
*   If the error flag is equal to 1, we have a negative or zero volume
*   returned from INVJ so output an appropriate error message.
*****
*
      CALL INVJ(J,JINV,VOLUME,OERR)
      IF (OERR.EQ.1) THEN
        WRITE (6,9020)
        RETURN
      ENDIF
*
*****
*   Next, the Cartesian derivatives are found.
*****
*
      CALL CARTE3D (JINV, DERIV,LNODZ)
*
*****
*   Lastly, the global coordinates at the point XITA are found as the
*   matrix product of ELXYZ and the shape functions
*****
*
```

```
CALL POINTS3D(POIN,SHAPE,ELXYZ,LNODZ)
```

```
*
*****
*   Make sure the point XITA is inside the brick by verifying that
*   the local coordinates are all between -1 and +1
*****
```

```
*
  IF ((1.LT.ABS(XITA(1))).OR.(1.LT.ABS(XITA(2)))
1    .OR.(1.LT.ABS(XITA(3)))) THEN
    WRITE(6,9000)
  ENDIF
9000 FORMAT('ERROR - PROGRAM HAS ATTEMPTED TO FIND THE SHAPE',
1         ' FUNCTION OF A POINT OUTSIDE THE BRICK.')
9020 FORMAT ('ERROR - THE DETERMINANT OF THE JACOBIAN FOR THIS',
1         ' ELEMENT IS NEGATIVE OR ZERO')
  RETURN
  END
```

```
*
*****
```

```
SUBROUTINE SFRH15(XITA,SHAPE,DERIV)
```

```
*
*****
*   Purpose: To compute the shape functions and their derivatives
*             for a 15 Node hierarchical brick element with the following
*             local coordinates:
```

NODE	ETA	XI	ZETA	TYPE
----	---	--	----	----
1	-1	-1	-1	Corner
2	1	-1	-1	"
3	1	1	-1	"
4	-1	1	-1	"
5	-1	-1	1	"
6	1	-1	1	"
7	1	1	1	"
8	-1	1	1	"
9	0	0	0	Barycentric
10	-1	0	0	Midface
11	1	0	0	"
12	0	-1	0	"
13	0	1	0	"
14	0	0	-1	"
15	0	0	1	"

```
*****
```

```
INTEGER
```

1 I,J

C

REAL*8

1 XITA(3),ETA,XI,ZETA,ETAMIN,XIMIN,ZETAMIN,ETAPL,XIPL,ZETAPL
2 ETASQ,XISQ,ZETASQ,EIGHTH,HALF,ONEM,TWOM,SHAPE(15),DERIV(3,15)

C

ETA = XITA(1)
XI = XITA(2)
ZETA = XITA(3)

C

ETAMIN = 1 - ETA
XIMIN = 1 - XI
ZETAMIN = 1 - ZETA

C

ETAPL = 1 + ETA
XIPL = 1 + XI
ZETAPL = 1 + ZETA

C

ETASQ = 1 - (ETA**2)
XISQ = 1 - (XI**2)
ZETASQ = 1 - (ZETA**2)

C

EIGHTH = 1.0DOO/8.0DOO
HALF = 1.0DOO/2.0DOO
ONEM = -1.0DOO
TWOM = -2.0DOO

.

*****

• Calculate the shape functions

*****

.

SHAPE(1) = EIGHTH*ETAMIN*XIMIN*ZETAMIN
SHAPE(2) = EIGHTH*ETAPL*XIMIN*ZETAMIN
SHAPE(3) = EIGHTH*ETAPL*XIPL*ZETAMIN
SHAPE(4) = EIGHTH*ETAMIN*XIPL*ZETAMIN
SHAPE(5) = EIGHTH*ETAMIN*XIMIN*ZETAPL
SHAPE(6) = EIGHTH*ETAPL*XIMIN*ZETAPL
SHAPE(7) = EIGHTH*ETAPL*XIPL*ZETAPL
SHAPE(8) = EIGHTH*ETAMIN*XIPL*ZETAPL
SHAPE(9) = ETASQ*XISQ*ZETASQ
SHAPE(10) = HALF*ETAMIN*XISQ*ZETASQ
SHAPE(11) = HALF*ETAPL*XISQ*ZETASQ
SHAPE(12) = HALF*ETASQ*XIMIN*ZETASQ
SHAPE(13) = HALF*ETASQ*XIPL*ZETASQ
SHAPE(14) = HALF*ETASQ*XISQ*ZETAMIN
SHAPE(15) = HALF*ETASQ*XISQ*ZETAPL

.

*****

* Calculate the shape function derivatives with respect to eta
*****

*

```

DERIV(1, 1) = ONEM*EIGHTH*XIMIN*ZETAMIN
DERIV(1, 2) = EIGHTH*XIMIN*ZETAMIN
DERIV(1, 3) = EIGHTH*XIPL*ZETAMIN
DERIV(1, 4) = ONEM*EIGHTH*XIPL*ZETAMIN
DERIV(1, 5) = ONEM*EIGHTH*XIMIN*ZETAPL
DERIV(1, 6) = EIGHTH*XIMIN*ZETAPL
DERIV(1, 7) = EIGHTH*XIPL*ZETAPL
DERIV(1, 8) = ONEM*EIGHTH*XIPL*ZETAPL
DERIV(1, 9) = TWOM*ETA*XISQ*ZETASQ
DERIV(1,10) = ONEM*HALF*XISQ*ZETASQ
DERIV(1,11) = HALF*XISQ*ZETASQ
DERIV(1,12) = ONEM*ETA*XIMIN*ZETASQ
DERIV(1,13) = ONEM*ETA*XIPL*ZETASQ
DERIV(1,14) = ONEM*ETA*XISQ*ZETAMIN
DERIV(1,15) = ONEM*ETA*XISQ*ZETAPL

```

*

*****

* Calculate the shape function derivatives with respect to xi
*****

*

```

DERIV(2, 1) = ONEM*EIGHTH*ETAMIN*ZETAMIN
DERIV(2, 2) = ONEM*EIGHTH*ETAPL*ZETAMIN
DERIV(2, 3) = EIGHTH*ETAPL*ZETAMIN
DERIV(2, 4) = EIGHTH*ETAMIN*ZETAMIN
DERIV(2, 5) = ONEM*EIGHTH*ETAMIN*ZETAPL
DERIV(2, 6) = ONEM*EIGHTH*ETAPL*ZETAPL
DERIV(2, 7) = EIGHTH*ETAPL*ZETAPL
DERIV(2, 8) = EIGHTH*ETAMIN*ZETAPL
DERIV(2, 9) = TWOM*XI*ETASQ*ZETASQ
DERIV(2,10) = ONEM*XI*ETAMIN*ZETASQ
DERIV(2,11) = ONEM*XI*ETAPL*ZETASQ
DERIV(2,12) = ONEM*HALF*ETASQ*ZETASQ
DERIV(2,13) = HALF*ETASQ*ZETASQ
DERIV(2,14) = ONEM*XI*ETASQ*ZETAMIN
DERIV(2,15) = ONEM*XI*ETASQ*ZETAPL

```

*

*****

* Calculate the shape function derivatives with respect to zeta
*****

*

```

DERIV(3, 1) = ONEM*EIGHTH*ETAMIN*XIMIN
DERIV(3, 2) = ONEM*EIGHTH*ETAPL*XIMIN
DERIV(3, 3) = ONEM*EIGHTH*ETAPL*XIPL
DERIV(3, 4) = ONEM*EIGHTH*ETAMIN*XIPL
DERIV(3, 5) = EIGHTH*ETAMIN*XIMIN

```

```

DERIV(3, 6) = EIGHTH*ETAPL*XIMIN
DERIV(3, 7) = EIGHTH*ETAPL*XIPL
DERIV(3, 8) = EIGHTH*ETAMIN*XIPL
DERIV(3, 9) = TWOM*ZETA*ETASQ*XISQ
DERIV(3,10) = ONEM*ZETA*ETAMIN*XISQ
DERIV(3,11) = ONEM*ZETA*ETAPL*XISQ
DERIV(3,12) = ONEM*ZETA*ETASQ*XIMIN
DERIV(3,13) = ONEM*ZETA*ETASQ*XIPL
DERIV(3,14) = ONEM*HALF*ETASQ*XISQ
DERIV(3,15) = HALF*ETASQ*XISQ
RETURN
END

```

.....

```

SUBROUTINE LAUFTURB(COORD, LNODS, LNODZ, NEL, TVISC, NELZ, NODMAX
1      , RADIUS, NITERA)
C
C   INTEGER I, LNODZ, NELZ, NODMAX, NEL, KPOIN, LNODS(LNODZ, NELZ)
1      , NITERA
C
C   REAL*8
1      COORD(NODMAX,7), TVISC(LNODZ), RADIUS(LNODZ), DIAM, XITA2, RAD
C
C   SAVE DIAM
*
* *****
*   Calculates the eddy viscosity at each of the nodal positions.
*   The values are provided by means of a discrete table which
*   approximates Laufer's eddy viscosity curves determined using
*   pipe flow at a Reynolds number of 50,000. The streamwise direction
*   must be aligned in the x - direction.
*
*   NOT TESTED WITH THE 16 NODE BRICK YET.
* *****
*
* *****
*   Find the diameter of the pipe.
* *****
*
IF (NITERA.EQ.1) THEN
    DIAM = 0.0
    DO 80 I=1, NODMAX
        RAD=SQRT((COORD(I,2)**2)+(COORD(I,3)**2))
        IF(DIAM.LT.RAD*2.) THEN
            DIAM = RAD*2.
        ENDIF
80    CONTINUE

```

```

ENDIF
*
*****
*   Initialize the turbulent viscosities to an arbitrary figure
*   of 2.22E+22 in double precision to identify orphan nodes.
*****
*
/   DO 90 I=1,LNODZ
      TVISC(I) = 2.22D+22
90   CONTINUE
*
*****
*   Assign a node a turbulent viscosity value. The viscosity is
*   associated with the radial distance from the centreline to a point.
*****
*
DO 100 I= 1,LNODZ
  KPOIN=IABS(LNODS(I,NEL))
  RADIUS(I)=SQRT((COORD(KPOIN,2)**2)+(COORD(KPOIN,3)**2))
  XITA2=1.-(2.*RADIUS(I)/DIAM)
  IF(XITA2.LE.0.1.AND.XITA2.GE.-0.1) THEN
    TVISC(I)=0.0
  ELSEIF(XITA2.LE.0.22.AND.XITA2.GT.0.10) THEN
    TVISC(I)=0.00806
  ELSEIF(XITA2.LE.0.275.AND.XITA2.GT.0.22) THEN
    TVISC(I)=0.01253
  ELSEIF(XITA2.LE.0.35.AND.XITA2.GT.0.275) THEN
    TVISC(I)=0.01343
  ELSEIF(XITA2.LE.0.475.AND.XITA2.GT.0.35) THEN
    TVISC(I)=0.01307
  ELSEIF(XITA2.LE.0.60.AND.XITA2.GT.0.475) THEN
    TVISC(I)=0.01271
  ELSEIF(XITA2.LE.1.1.AND.XITA2.GT.0.60) THEN
    TVISC(I)=0.01236
  ENDIF
  IF(TVISC(I).EQ.2.22E+22) THEN
    PRINT*,'NO VALUE FOR NODE ',KPOIN,' WITH RADIUS ',RADIUS(I)
    1,' AND XITA2 ',XITA2
  ENDIF
100 CONTINUE
*
*****
*   Change the absolute eddy viscosity values at the midface nodes
*   to differences away from a the average of the four bounding corner
*   nodes to remain commensurate with the hierarchical formulation
*****
*
TVISC(10)=TVISC(10)-((TVISC(1)+TVISC(4)+TVISC(8)+TVISC(5))/4.)

```



```

TVISC(11)=TVISC(11)-((TVISC(2)+TVISC(3)+TVISC(7)+TVISC(6))/4.)
TVISC(12)=TVISC(12)-((TVISC(1)+TVISC(5)+TVISC(6)+TVISC(2))/4.)
TVISC(13)=TVISC(13)-((TVISC(4)+TVISC(8)+TVISC(7)+TVISC(3))/4.)
TVISC(14)=TVISC(14)-((TVISC(1)+TVISC(2)+TVISC(3)+TVISC(4))/4.)
TVISC(15)=TVISC(15)-((TVISC(5)+TVISC(6)+TVISC(7)+TVISC(8))/4.)
TVISC(9) =TVISC(9) -((TVISC(1)+TVISC(2)+TVISC(3)+TVISC(4) +
1      TVISC(5)+TVISC(6)+TVISC(7)+TVISC(8))/8.) -
2      ((TVISC(10)+TVISC(11)+TVISC(12)+TVISC(13) +
3      TVISC(14)+TVISC(15))/2.)

```

```

RETURN

```

```

END

```

```

*****

```

```

SUBROUTINE STIFCHECK(ELSTIF,LVABZ,ERHS)

```

```

C

```

```

INTEGER LVABZ,IROW,JCOL,PTR

```

```

C

```

```

REAL*8 ELSTIF(LVABZ,LVABZ),SUM(33),SUMU(33),SUMV(33),SUMW(33)
1, ERHS(LVABZ),SUM2(33)

```

```

*

```

```

*****

```

```

* Subroutine to sum the individual rows of the element stiffness
* matrix. The sum should be zero for all rows just as it is
* for the rigid body displacement for the stress case.
*
* Only the first 24 rows and columns are considered since they
* are the only ones which actually provide for velocity and
* pressure gradients of zero (remember hierarchical formulation).
*
* Normally this subroutine is used only in the debugging phase
* and the call to it in VFL3D14 is commented out since the information
* output is not required.

```

```

*****

```

```

*

```

```

DO 1 IROW=1,LVABZ

```

```

WRITE(6,FMT='(I3,12G10.2)') IROW,(ELSTIF(IROW,JCOL),JCOL=1,12)

```

```

WRITE(6,FMT='(3I,12G10.2)') (ELSTIF(IROW,JCOL),JCOL=13,24)

```

```

WRITE(6,FMT='(3I,12G10.2)') (ELSTIF(IROW,JCOL),JCOL=25,LVABZ)

```

```

WRITE(6,FMT='(//)')

```

```

1

```

```

CONTINUE

```

```

DO 3 IROW = 1,LVABZ

```

```

WRITE(6,FMT='(I3,1X,3G11.3)') IROW,ERHS(IROW)

```

```

3

```

```

CONTINUE

```

```

DO 5 IROW = 1,LVABZ

```

```

SUM(IROW) = 0.0

```

```

SUM2(IROW) = 0.0

```

```

SUMU(IROW) = 0.0

```

```

      SUMV(IROW) = 0.0
      SUMW(IROW) = 0.0
5    CONTINUE
      DO 10 IROW=1, LVABZ
        DO 20 JCOL=1, 24
          \ SUM(IROW) = SUM(IROW) + ELSTIF(IROW, JCOL)
20    CONTINUE
          IF (ABS(SUM(IROW)) .LT. 1E-06) SUM(IROW)=0.0
10    CONTINUE
          DO 30 IROW=1, LVABZ
            DO 40 JCOL=1, 8
              PTR=3*JCOL - 2
              SUMU(IROW) = SUMU(IROW) + ELSTIF(IROW, PTR)
              SUMV(IROW) = SUMV(IROW) + ELSTIF(IROW, PTR+1)
              SUMW(IROW) = SUMW(IROW) + ELSTIF(IROW, PTR+2)
40    CONTINUE
              IF (ABS(SUMU(IROW)) .LT. 1E-06) SUMU(IROW)=0.0
              IF (ABS(SUMV(IROW)) .LT. 1E-06) SUMV(IROW)=0.0
              IF (ABS(SUMW(IROW)) .LT. 1E-06) SUMW(IROW)=0.0
              SUM2(IROW) = SUM2(IROW) + SUMU(IROW) + SUMV(IROW) + SUMW(IROW)
30    CONTINUE
              WRITE(6, 600)
              DO 100 IROW=1, LVABZ
                WRITE(6, 501) IROW, SUM(IROW), SUM2(IROW), SUMU(IROW), SUMV(IROW)
1,              SUMW(IROW)
100    CONTINUE
500    FORMAT(X, 'THE SUM OF THE ROWS OF THE ELEMENT STIFFNESS MATRIX
1 ARE GIVEN BELOW', /)
501    FORMAT(X, 'SUMS (TOT, 2nd TOT, U, V, W) . ROW ', I2, ' ARE ', 5G15.5)
      RETURN
      END

```

Appendix C

FRONTA

```

SUBROUTINE FRONTA(LNOMAX, LVMAX, NELZ, NODMAX, NEQUAT
1.      LNODS, NDFMAX, VFIX, NODFIX, NFIX, NITERA, COORD
2.      NCHECK, THETA, TOLERA, LTYPE, KODFIX, NCORD
3.      NEWRHS, LIMFRO, NODFRO, LDEST, ELSTIF, EQ, QQ, BC, SK, R1
4.      PVKOL, JMOD, KPIV, LPIV, NOPP, MDF, NCOD, KDEST, KHED
5.      LHED, NK)

```

C

IMPLICIT LOGICAL(A-Z)

C

COMMON/SYSTEM/ NDF(1648), NDFPTR(160)
 SAVE

*

*

FRONTAL ELIMINATION ROUTINE USING FULL PIVOTING

*

*

AUTHOR - P. HOOD (IJNME 1976 VOL. 10)

*

AMENDED- G. VEMAGANTI (1983)

*

- M. DULUDE (DEC. 1984)

*

*

Passing in :

*

Passing out:

*

*****

*

DICTIONARY OF VARIABLES -

*

*

BC (NEQUAT) FOR EACH NODAL DEGREE OF FREEDOM THIS IS ASSIGNED
 ZERO VALUE, UNLESS A BOUNDARY CONDITION IS
 APPLIED, IN WHICH CASE IT IS GIVEN THE VALUE OF
 THE BOUNDARY CONDITION. THE VECTOR FORM OF THE
 ARRAY VFIX.

*

DISDIV ELEMENTAL DISCRETE DIVERGENCE

*

DIVERG VECTOR OF DISDIV FOR EACH ELEMENT

*

ERSHU (LNOMAX) U VELOCITY RIGHT HAND SIDE AS TAKEN FROM ABFIND

*

ERSHV AND ERSW ARE THE SAME AS ERSHU BUT FOR V AND W VELOCITIES

*

EQ (NMAX, NMAX) SUBMATRIX OF MATRIX A IN THE EQUATION AX=B INTO
 WHICH ALL ELEMENT STIFFNESS MATRICES ARE
 INTRODUCED

*

ELSTIF(LVMAX, THE ELEMENT STIFFNESS MATRIX FROM THE CALL TO
 LVMAX) SUBROUTINE ABFIND

*

GASH REAL DUMMY VARIABLE USED FOR TEMPORARILY REQUIRED
 VALUES

*

* IDF NUMBER OF DEGREES OF FREEDOM OF NODE CURRENTLY
 * BEING TREATED
 * JMOD (NMAX) CONTAINS THE ROW NUMBER IN MATRIX EQ. OF THE
 * BOUNDARY CONDITION EQUATIONS
 * KDEST (LVMAX) THE ELEMENT DESTINATION VECTOR, TO INDICATE THE
 * ROWS IN MATRIX EQ WHERE THE ELEMENT STIFFNESS
 * MATRIX ENTRIES ARE TO BE ASSEMBLED
 * KEY RECORD NUMBER FOR THE FORMATTED I/O
 * KHED (NMAX) THE HEADING VECTOR OF THE ROWS OF EQ TO
 * INDICATE WHICH EQUATION IS ASSOCIATED WITH EACH ROW
 * OF EQ
 * KPIVRO THE PIVOTAL ROW NUMBER
 * KPIV (NMAX) THE FULLY SUMMED ROW NUMBERS ARE RECORDED IN THE ARRAY
 * KROW NUMBER OF ROWS OF EQ WHICH ARE OCCUPIED
 * LCOL NUMBER OF COLUMNS OF MATRIX EQ WHICH ARE
 * OCCUPIED (EQUAL TO KROW FOR GAUSSIAN ELIMINATION)
 * LDEST (LVMAX) SAME AS KDEST BUT FOR COLUMN RATHER THAN ROW
 * LHED (NMAX) SAME AS KHED BUT FOR COLUMN RATHER THAN ROW
 * LIMFRO MAXIMUM FRONTWIDTH
 * LNODS (LNOMAX, NODAL NUMBERING ARRAY. SECOND ARGUMENT IS THE
 * NELZ) ELEMENT NUMBER, THE FIRST LOCATES THE NODE NUMBERS
 * OCCURRING IN THE ELEMENT
 * LNODZ NUMBER OF NODES IN THE CURRENT ELEMENT
 * LNOMAX MAXIMUM NUMBER OF NODES IN AN ELEMENT IN THIS PROBLEM
 * LPIV (NMAX) SAME AS KPIV BUT FOR COLUMNS RATHER THAN ROWS
 * LPIVCO THE PIVOTAL COLUMN NUMBER
 * LVABZ TOTAL NUMBER OF DOF IN THE CURRENT ELEMENT
 * LVMAX MAXIMUM NUMBER OF ELEMENT DOF IN THE PROBLEM
 * MDF (NODMAX) NO. OF DEGREES OF FREEDOM FOR EACH NODE IN THE MESH
 * MWGA A ZERO VALUE INDICATES A SINGLE DEGREE OF FREEDOM.
 * A VALUE OF ONE INDICATES MULTIPLE DOFS. A VALUE OF TWO
 * IS A SPECIAL FLAG FOR THE NO X-ACCELERATION CASE.
 * NCOD (NEQUAT) ZERO ENTRY CORRESPONDS TO A NODAL DOF WITH NO
 * APPLIED BOUNDARY CONDITION, AND A UNITY VALUE FOR
 * AN APPLIED BOUNDARY CONDITION
 * NCRIT THE CRITICAL NUMBER OF ROWS OF EQ. MUST BE CHOSEN
 * SMALLER THAN NMAX BY THE NUMBER OF DOF IN AN
 * ELEMENT. MINIMUM IS THE NO. OF DOF IN THE
 * LARGEST FRONTWIDTH. THE LARGER NCRIT IS MADE, THE
 * WIDER THE PIVOTAL CHOICE AND THE SLOWER THE
 * EXECUTION
 * ND1 CHANNEL NUMBER OF THE DISC ON WHICH THE PIVOTAL
 * ROW IS WRITTEN
 * NDFMAX MAXIMUM NUMBER OF DOF ON A NODE. REQUIRED FOR THE
 * DIMENSIONING OF VFIX.
 * NDOF NUMBER OF DEGREES OF FREEDOM OF NODE CURRENTLY
 * BEING TREATED
 * NELZ NUMBER OF ELEMENTS IN THE MESH

* NELL ELEMENT NUMBER OF THE STIFFNESS MATRIX CURRENTLY
 * BEING ASSEMBLED
 * NERROR ERROR CODE WITH INDICATED MEANING
 * NEWRHS NO. OF NEW RIGHT HAND SIDES EXPECTED IN A RE-SOLUTION
 * CASE. REQUIRED FOR THE DIMENSIONING OF VFIX.
 * NFG POINTER FOR NODE WITH PRESCRIBED VALUE CURRENTLY
 * BEING TREATED
 * NFIX NUMBER OF NODES WITH PRESCRIBED VALUES
 * NK (LVMAX) EQUATION NUMBER ASSIGNED TO EACH DOF OF THE
 * ELEMENT CURRENTLY BEING ASSEMBLED
 * NLP CHANNEL NUMBER OF THE LINE PRINTER
 * NMAX CHOSEN TO BE GREATER THAN THE LARGEST FRONTWIDTH
 * PLUS THE NEXT ELEMENT EQUATIONS TO BE ASSEMBLED.
 * THIS ALSO GIVES THE DIMENSIONS OF MATRIX EQ
 * NOD NODE NUMBER OF THE NODE CURRENTLY BEING TREATED
 * NODE CURRENT ELEMENT EQUATION BEING TREATED
 * NODFIX (NEQUAT) NODE NUMBER OF NODE WITH PRESCRIBED VALUE
 * NODFRO (NMAX) SEQUENCE OF NODE NUMBERS CURRENTLY IN THE FRONT
 * NODMAX TOTAL NUMBER OF NODES IN THE MESH
 * NOPP (NODMAX) GIVES THE CODED VALUE OF THE FIRST DOF OF EACH NODE
 * NEQUAT TOTAL NUMBER OF EQUATIONS TO BE SOLVED
 * NSPL FLAG READ FROM THE DISC READ IN ABFIND WHICH
 * INDICATES WHETHER THE ELEMENT IS NO X-ACCELERATION
 * NTRA CODED UNITY BEFORE ENTRY INTO FRONT, AND ZERO ON
 * SUBSEQUENT ENTRY, IN ORDER TO PREVENT PREFRONT
 * PROCESS AFTER THE FIRST CYCLE
 * PIVA TEMPORARY VALUE USED TO FIND THE ABSOLUTE PIVOT
 * PIVOT ENTRY USED AS A PIVOT IN THE ELIMINATION
 * PVKSL (NMAX) CONTAINS THE PIVOT COLUMN
 * QQ (NMAX) CONTAINS THE PIVOTAL ROW NORMALIZED BY DIVISION
 * BY THE PIVOT
 * R1 (NEQUAT) THE RIGHT HAND SIDE VECTOR FOR EACH ELEMENT FROM
 * ABFIND
 * SK (NMAX**2) THE RESULTS ARE OUTPUT IN THIS ARRAY
 * SLAMAS (2, NEQUAT/2) THE ARRAY GIVING THE GLOBAL MASTER NODE NUMBER
 * CORRESPONDING TO THE GLOBAL SLAVE NODE NUMBER.
 * SLANUM NUMBER OF SLAVE NODES THAT HAVE BEEN REMOVED
 * SMNODS (2,20) DATA ARRAY RELATING THE LOCAL MASTER AND SLAVE NODES.
 * A ZERO MASTER VALUE INDICATES THE LOCAL NODE IS NOT
 * A SLAVE. A NON-ZERO VALUE GIVES THE LOCAL MASTER OF
 * THE LOCAL SLAVE.
 * VARB2(NMAX**2) VALUES OF ARRAY SK FROM PREVIOUS ITERATION
 * VFIX (3D ARRAY) VALUE OF BOUNDARY CONDITIONS FOR PRESCRIBED VALUE NOD
 *
 ***TEMPORARY VARIABLES - KRO,LCO,KPIVR,LPIVC
 *
 ***COUNTERS - NEL, INOD, IC, JC, KC, LC, NF, ND, I, J, K, L, LK,
 * NDFG, II, KK, LL, IR, KR IRR, COUNT

C

INTEGER

1 NDF, NDFPTR, LIMFRO, LNOMAX, LVMAX, NELZ, NODMAX, NEQUAT, NMAX

2, MAXEL

PARAMETER (MAXEL=10)

C

INTEGER

1 NLARG, ND1, LPIVC, NCRIT, NLAST, I, J, K, L, M, N, O, LPIVCO, KRW, NODE, LCOL

2, IDF, KPIVR, LCO, NLP, KRO, IRR, NERROR, KPIVRO, NELL, MWGA, NCORD, NEWRHS

3, NDFG, NDOF, NFIX, IC, JC, ND, NF, INOD, NCHECK, KODEL, NFG, NITERA, KEY

4, KC, KROW, LC, II, KH, LH, KK, LK, LL, IR, NN, KR, KT, KOD, NEL, NDFMAX

5, NOD, JMOD(LVMAX+LIMFRO), KPIV(LVMAX+LIMFRO), LPIV(LVMAX+LIMFRO)

6, MDF(NODMAX), NCOD(NEQUAT), NODFRO(LVMAX+LIMFRO), LTYPE(NELZ)

7, KDEST(LVMAX), LDEST(LVMAX), KODFIX(NFIX), NODFIX(NFIX)

8, KHED(LVMAX+LIMFRO), LHED(LVMAX+LIMFRO), LNODS(LNOMAX, NELZ)

9, NOPP(NODMAX), NK(LVMAX), FIXITY, LVABZ, SLANUM, SMNODS(2, 20)

1, SLAMAS(2, 50), NSPL, START, COUNT, LNODZ

C

REAL*8

1 FAC, PIVOT, RHS, PIVA, THETA, TOLERA, PIVOTF, PRES(MAXEL)

2, QQ(LVMAX+LIMFRO), COORD(NODMAX, NCORD), ELSTIF(LVMAX, LVMAX)

2, BC(NEQUAT), EQ(LVMAX+LIMFRO, LVMAX+LIMFRO)

3, VFIX(NEWRHS, NDFMAX, NFIX), R1(NEQUAT), PVKOL(LVMAX+LIMFRO)

4, SK((LVMAX+LIMFRO)*(LVMAX+LIMFRO)), DIVERG(MAXEL)

C

LOGICAL MEMBER, SLAVE

C

DATA SMNODS /1.0, 2.1, 3.4, 4.0, 5.0, 6.5, 7.8, 8.0, 9.0, 10.12
1, 11.0, 12.0, 13.0, 14.13, 15.16, 16.0, 17.0, 18.20, 19.0, 20.0 /

C

KEY=0

SLANUM = 0

NMAX = LIMFRO+LVMAX

MWGA = 0

IF(LVMAX.NE.LNOMAX)MWGA = 1

ND1 = 14

NLP = 6

NLARG = 0.50*NMAX

*

*****

*

Prefront - where the pointer and BC arrays are created

*

*

NDFG = 0

IC = 0

*

```

*****
*   Search sequentially through all the nodes of subsequent elements
*****
*
      DO 101 NEL = 1,NELZ
        DO 101 INOD = 1,LNOMAX
          IF(LTYPE(NEL).EQ.144.AND.INOD.EQ.9) GOTO 101
          NOD = IABS(LNODS(INOD,NEL))
*
*****
*   Find the lowest valued node that has not already been treated from
*   a search through a previous element
*****
*
          IF(NEL.GT.1) THEN
            DO 103 JC=1,IC
              IF(MOD.EQ.MODFRO(JC)) GO TO 101
103          CONTINUE
            ENDIF
            IC = IC+1
            MODFRO(IC) = NOD
*
*****
*   Get number of DOF from the INODth node of element type LTYPE
*   corresponding to element NEL.
*****
*
            NDOF = MDF(MDEPTR(LTYPE(NEL)) + INOD - 1)
            MDF(NOD) = NDOF
            FIXITY = 0
            NPG = 0
*
*****
*   Find when the current node has prescribed value, then assign local
*   values accordingly. Prescribed value nodes get a fixed zero value
*   since no deviation is allowed from the fixity value (remember the
*   dof vector is one of corrections to the nodal dof). Free
*   variables are given a floating zero value
*****
*
            DO 110 NF=1,NFIX
              IF(MOD.EQ.MODFIX(NF)) THEN
                FIXITY = 1
                MOD = MODFIX(NF)
                NPG = NF
                GO TO 121
              ENDIF
110          CONTINUE

```

```

121      DO 106 ND=1,NDOF
*
*****
*   Here is where the fixity code is examined to determine whether node
*   entry in BC should be a prescribed value or not.
*****
*
      NDFG = NDFG+1
      KODEL = 10**(NDOF-ND)
      IF (FIXITY.EQ.1.AND.(KOD.GE.KODEL)) THEN
        KOD = KOD-KODEL
        NCOD(NDFG) = 1
        BC(NDFG) = 0.0
      ELSE
        NCOD(NDFG) = 0
        BC(NDFG) = 0.0
      ENDIF
*
*****
*   Give NOPP the location of the first DOF of each node
*****
*
      IF(ND.EQ.1) NOPP(ND) = NDFG
106      CONTINUE
101      CONTINUE
      DO 102 I = 1,NEQUAT
        R1(I) = 0.0
102      CONTINUE
      NCRIT = NMAX-LVMAX
      IF (NCRIT.LT.LVMAX) NCRIT = LVMAX
      NELL=0
C      WRITE(NLP,400)
      LCOL=0
      KROW=0
      DO 16 I=1,NMAX
        DO 16 J=1,NMAX
          EQ(I,J)=0.0
16      CONTINUE
*
*****
*   Increment to go through all elements
*****
*
18      NELL=NELL+1
      CALL ABFIND(LVABZ,ELSTIF,NELL,R1,LNOMAX,LNODS,NELZ,NOPP,
1          LVMAX,NSPL,MDF,NODMAX,LNODZ,PRES,DIVERG)
      IF(NSPL.EQ.1) MWGA = 2
      N=NELL

```


KC=0

```

*
*****
*   If one DOF, LVABZ=LNOMAX so simply step through nodes to
*   establish the equation numbers of array NK.  If multiple
*   DOFs, must take into account the number of DOF of each node
*   If a zero x-acceleration case, then must take into account that
*   the K matrix is now only a 60x60 matrix rather than the normal
*   68x68.  So, remove the pointer to 8 of the nodes (i.e., the slaves)
*****

```

```

*
      IF(MWGA.EQ.0) THEN
        DO 20 I=1,LNODZ
          NK(I) = LNODS(I,N)
20      CONTINUE

```

```

*
*****
*   In both the MWGA=1 & 2 cases, the procedure is the same:
*   - Find first global dof & the no. of dof for this node
*   - Increment the NK index to store the next value
*   - Find the global dof value corresponding to this local dof
*   - If the local node was at its last occurrence (i.e., a
*     negative node no.), make NK dof negative also
*   - Store the global dof in NK
*   The only difference in the no x-acceleration case is that you
*   neglect the slave u dof pointers since they've been compacted
*   out of the [K] matrix.
*****

```

```

      ELSEIF (MWGA.EQ.1) THEN
        DO 22 J=1,LNODZ
          NOD = LNODS(J,N)
          M=IABS(NOD)
          K=NOPP(M)
          IDF=MOD(M)
          DO 22 L=1,IDF
            KC=KC+1
            II=K+L-1
            IF(NOD.LT.0) II=-II
            NK(KC)=II

```

```

22      CONTINUE

```

```

      ELSEIF (MWGA.EQ.2) THEN
        MWGA = 1
        DO 21 J=1,LNODZ
          NOD = LNODS(J,N)
          M=IABS(NOD)
          K=NOPP(M)
          IDF=MOD(M)

```

SLAVE = MEMBER(J)

*

*****

- * If a slave node, only point to the v,w,& p dofs (if p exists
- * for this node) since u has been compacted out.

*****

*

IF(SLAVE) THEN

START = 2

*

*****

- * Store the global slave and master node numbers of this pair
- * The local slave dof number is from SMNODS(1,J) The master is from
- * SMNODS(2,J). Only do this part of the exercise if this is the last
- * appearance of this slave node (i.e., node no. is negative)

*****

*

IF(MOD.LI,0) THEN

SLANUM = SLANUM + 1

SLAMAS(1,SLANUM) = IABS(LNODS(SMNODS(1,J),N))

SLAMAS(2,SLANUM) = IABS(LNODS(SMNODS(2,J),N))

*

*****

- * Add the value of the slave's right hand side to the master's

*****

*

R1(NOPP(SLAMAS(2,SLANUM)))=R1(NOPP(SLAMAS(2,SLANUM))) +

1

R1(NOPP(SLAMAS(1,SLANUM)))

ENDIF

ELSE

*

*****

- * Case of a node which is not a slave node

*****

*

START = 1

ENDIF

DO 23 L=START, IDF

KC=KC+1

II=K+L-1

IF(MOD.LI,0) II=-II

WK(KC)=II

23 CONTINUE

21 CONTINUE

ENDIF

*

.....

- * Set up heading and destination vectors. Heading vectors give the

* dof number of the argument equation number in EQ. Destination *
 * vectors give the equation number in EQ of the argument dof number.*
 *****

```

      DO 52 LK=1,LVABZ
        NODE=NK(LK)
        IF(LCOL.EQ.0)GO TO 28
        DO 24 L=1,LCOL
          LL=L
          IF(IABS(NODE).EQ.IABS(LHED(L)))GO TO 32
24      CONTINUE
28      CONTINUE
        LCOL=LCOL+1
        LDEST(LK)=LCOL
        LHED(LCOL)=NODE
        GO TO 36
32      CONTINUE
        LDEST(LK)=LL
        LHED(LL)=NODE
36      CONTINUE
        IF(KROW.EQ.0)GO TO 44
        DO 42 K=1,KROW
          KK=K
          IF(IABS(NODE).EQ.IABS(KHED(K)))GO TO 48
42      CONTINUE
44      CONTINUE
        KROW=KROW+1
        KDEST(LK)=KROW
        KHED(KROW)=NODE
        GO TO 52
48      CONTINUE
        KDEST(LK)=KK
        KHED(KK)=NODE
52      CONTINUE
C      WRITE(NLP,420)KROW,LCOL
C      WRITE(NLP,424)
C      WRITE(NLP,428) (KHED(K),LHED(K),K=1,LVABZ)
C      WRITE(NLP,432)
C      WRITE(NLP,428) (KDEST(K),LDEST(K),K=1,LVABZ)
        IF(KROW.LE.NMAX.AND.LCOL.LE.NMAX)GO TO 54
        NERROR=2
        WRITE(NLP,417)NERROR
        STOP
54      CONTINUE

```

*****

* Sum the element entries into the matrix EQ
 *****

```

      DO 56 L=1,LVABZ
        LL=LDEST(L)
        DO 56 K=1,LVABZ
          KK=KDEST(K)
          EQ(KK,LL)=EQ(KK,LL)+ELSTIF(K,L)
56    CONTINUE
C      WRITE(NLP,436)NELL
C      WRITE(NLP,440)((EQ(I,J),J=1,NMAX),I=1,NMAX)
      IF(KROW.LT.NCRIT.AND.NELL.LT.NELZ)GO TO 18
*
*****
*   Begin the elimination procedure
*****
*
*****
*   Find out which matrix elements are fully summed
*****
*
60    CONTINUE
      LC=0
*
*****
*   Find which columns in the heading vector have been fully summed
*****
*
      DO 64 L=1,LCOL
        IF(LHED(L).GE.0)GO TO 64
        LC=LC+1
        LPIV(LC)=L
64    CONTINUE
      IR=0
      KR=0
*
*****
*   Find which rows in the heading vector have been fully summed
*****
*
      DO 68 K=1,KROW
        KT=KHED(K)
        IF(KT.GE.0)GO TO 68
        KR=KR+1
        KPIV(KR)=K
        KRO=IABS(KT)
*
*****
*   Identify those rows which have an associated prescribed value
*   by changing its NCOD value to 2 from 1. IR gives the current

```

* number of those rows.

*****

*

IF(NCOD(KRO).NE.1)GO TO 68

IR=IR+1

JMOD(IR)=K

NCOD(KRO)=2

R1(KRO)=BC(KRO)

68 CONTINUE

*

*****

* Modify the equations with applied boundary conditions.

*****

*

C WRITE(NLP,448)LC,KR

C WRITE(NLP,428) (LPIV(K),KPIV(K),K=1,LVABZ)

IF(IR.EQ.0)GO TO 71

C WRITE(NLP,466)

DO 70 IRR=1,IR

K=JMOD(IRR)

C WRITE(NLP,428)K

*

*****

* Change the EQ row of a prescribed value node to all zeroes except

* for a 1.0 entry on the diagonal. The RHS is then given the value

* of the prescribed value.

*****

*

KH=IABS(KHED(K))

DO 69 L=1,LCOL

EQ(K,L)=0.C

LH=IABS(LHED(L))

IF(LH.EQ.KH) EQ(K,L)=1.0

69 CONTINUE

70 CONTINUE

71 CONTINUE

IF(LC.GT.5) GO TO 72

IF(LC.GT.0.AND.WELL.EQ.WELZ) GO TO 72

NCRIT=NCRIT+5

IF(NCRIT.LE.NLARG)GO TO 18

NERROR=3

WRITE(NLP,418)NERROR

STOP

72 CONTINUE

C WRITE(NLP,464) (I,R1(I),I=1,NEQUAT)

*

*****

* Search for absolute pivot. Use all fully assembled columns for each

```

*   fully assembled row.
*****
*
      PIVOT=0.0
      DO 76 L=1,LC
        LPIVC=LPIV(L)
        DO 74 K=1,KR
          KPIVR=KPIV(K)
          PIVA=EQ(KPIVR,LPIVC)
          IF(ABS(PIVA).LT.ABS(PIVOT))GO TO 74
          PIVOT=PIVA
          LPIVCO=LPIVC
          KPIVRO=KPIVR
74      CONTINUE
76      CONTINUE
*
*****
*   Normalize the pivotal row:
*****
*
      KRO=IABS(KHED(KPIVRO))
      LCO=IABS(LHED(LPIVCO))
C      WRITE(NLP,452)KRO,LCO,PIVOT
*
*****
*   Check for ill conditioning of element matrices
*****
*
      IF(ABS(PIVOT).LT.1E-16) WRITE(NLP,476)
*
*****
*   Calculate the normalized pivotal row entries
*****
*
      DO 80 L=1,LCOL
        QQ(L)=EQ(KPIVRO,L)/PIVOT
80      CONTINUE
      RHS=R1(KRO)/PIVOT
      R1(KRO)=RHS
      PVKOL(KPIVRO)=PIVOT
C      WRITE(NLP,468)
C      WRITE(NLP,440)(QQ(L),L=1,LCOL)
*
*****
*   Eliminate then delete pivotal row and column
*****
*
      IF(KPIVRO.EQ.1)GO TO 104

```

```

KPIVR=KPIVRO-1
DO 100 K=1,KPIVR
  KRW=IABS(KHED(K))
  FAC=EQ(K,LPIVCO)
C  WRITE(NLP,480)FAC
  PVKOL(K)=FAC
  IF(LPIVCO.EQ.1.OR.FAC.EQ.0.)GO TO 88
  LPIVCO=LPIVCO-1
  DO 84 L=1,LPIVC
    EQ(K,L)=EQ(K,L)-FAC*QQ(L)
84  CONTINUE
88  IF(LPIVCO.EQ.LCOL)GO TO 96
  LPIVC=LPIVCO+1
  DO 92 L=LPIVC,LCOL
    EQ(K,L-1)=EQ(K,L)-FAC*QQ(L)
92  CONTINUE
96  R1(KRW)=R1(KRW)-FAC*RHS
100 CONTINUE
104 IF(KPIVRO.EQ.KROW)GO TO 128
  KPIVR=KPIVRO+1
  DO 124 K=KPIVR,KROW
    KRW=IABS(KHED(K))
    FAC=EQ(K,LPIVCO)
C  WRITE(NLP,480)FAC
    PVKOL(K)=FAC
    IF(LPIVCO.EQ.1)GO TO 112
    LPIVC=LPIVCO-1
    DO 108 L=1,LPIVC
      EQ(K-1,L)=EQ(K,L)-FAC*QQ(L)
108 CONTINUE
112 IF(LPIVCO.EQ.LCOL)GO TO 120
  LPIVC=LPIVCO+1
  DO 116 L=LPIVC,LCOL
    EQ(K-1,L-1)=EQ(K,L)-FAC*QQ(L)
116 CONTINUE
120 R1(KRW)=R1(KRW)-FAC*RHS
124 CONTINUE
128 CONTINUE
*
*****
*   Write the pivotal equation on the disc
*****
*
  KEY=KEY+1
  WRITE(UNIT=ND1,REC=KEY)
  1. KRO,LCOL,LPIVCO,(LHED(L),QQ(L),L=1,LCOL)
  2. KROW,PIVOT,KPIVRO,(PVKOL(K),KHED(K),K=1,KROW)
  DO 129 K=1,KROW

```

```

      EQ(K,LCOL)=0.0
129  CONTINUE
      DO 130 L=1,LCOL
        EQ(KROW,L)=0.0
130  CONTINUE
C     WRITE(NLP,436)NELL
C     WRITE(NLP,440)((EQ(I,J),J=1,NMAX),I=1,NMAX)
C     WRITE(NLP,460)
C     WRITE(NLP,464)(I,R1(I),I=1,NEQUAT)
*
*****
*   Rearrange the heading vectors
*****
*
      LCOL=LCOL-1
      IF(LPIVCO.EQ.LCOL+1)GO TO 136
      DO 132 L=LPIVCO,LCOL
        LHED(L)=LHED(L+1)
132  CONTINUE
136  KROW=KROW-1
      IF(KPIVRO.EQ.KROW+1)GO TO 144
      DO 140 K=KPIVRO,KROW
        KHED(K)=KHED(K+1)
140  CONTINUE
144  CONTINUE
C     WRITE(NLP,420)KROW,LCOL
C     WRITE(NLP,424)
C     WRITE(NLP,428)(KHED(K),LHED(K),K=1,LVABZ)
*
*****
*   Determine whether to assemble, eliminate, or backsubstitute
*****
*
      IF(KROW.GT.NCRIT)GO TO 60
      IF(NELL.LT.NELZ)GO TO 18
      IF(KROW.GT.1)GO TO 60
      LCO=IABS(LHED(1))
      KPIVRO=1
      PIVOT=EQ(1,1)
      KRO=IABS(KHED(1))
      LPIVCO=1
      QQ(1)=1.0
C     WRITE(NLP,452)LCO,KRO,PIVOT
      IF(ABS(PIVOT).LT.1E-16) WRITE(NLP,476)
      R1(KRO)=R1(KRO)/PIVOT
      KEY=KEY+1
      WRITE(UNIT=ND1,REC=KEY)
1     KRO,LCOL,LPIVCO,LHED(1),QQ(1)

```



```

2, KROW,PIVOT,KPIVRO,PVKOL(1),KHED(1)
  CALL BACSUB(NEQUAT,SK,BC,R1,NCOD,NODMAX,NOPP,MDF,ND1
1,      NMAX,KEY,KHED,LHED,PVKOL,QQ,SLANUM,SLAMAS)
  CALL TOLREL(NITERA,NEQUAT,THETA,TOLERA,SK,LNODS,NOPP,NELZ,
1      LNOMAX,NODMAX,NCHECK,MDF,COORD,NCORD,NMAX,LNODZ,
2      PRES,DIVERG,LTYPE)
C400  FORMAT(5H NODE,6H NLAST)
C4004 FORMAT(1X,2I5)
C4008 FORMAT(//16H NODAL NUMBERING/)
C413  FORMAT(9I5)
C415  FORMAT(/8H NERROR=,I5//I5,
C      1 62H THE ELEMENT HAS MORE THAN ONE NODE WITH THE
C      1/ 62H SAME NODAL NUMBER
C      1/)
417  FORMAT(/8H NERROR=,I5//
1 62H THE DIFFERENCE NMAX-NLARG IS NOT SUFFICIENTLY LARGE
1 /62H TO PERMIT THE ASSEMBLY OF THE NEXT ELEMENT---
1 /62H EITHER INCREASE NMAX OR LOWER NLARG
1 /)
418  FORMAT(/8H NERROR=,I5//
1 62H THERE ARE NO MORE ROWS FULLY SUMMED,THIS MAY BE DUE TO---
1 /62H (1)INCORRECT CODING OF LNODS OR NK ARRAYS
1 /62H (2)INCORRECT VALUE OF NLARG. INCREASE NLARG TO PERMIT
1 /62H WHOLE FRONT TO BE ASSEMBLED
1 /)
C420  FORMAT(/6H KROW=,I5,6H LCOL=,I5/)
C424  FORMAT(/6H KHED,6H LHED/)
C428  FORMAT(2I6)
C432  FORMAT(/6H KDEST,6H LDEST/)
C436  FORMAT(/22H EQ MATRIX ELEMENT NO=,I3/)
C440  FORMAT(/20F5.2)
C48.  FORMAT(/4H LC=,I3,/4H KR=,I3/6H LPIV,6H KPIV/)
C452  FORMAT(13H PIVOTAL ROW=,I4,16H PIVOTAL COLUMN=,I4,7H,PIVOT=,E20.10
C      1)
C456  FORMAT(/5H JMOD/)
C460  FORMAT(/18H RIGHT HAND VECTOR/)
C464  FORMAT(I5,E20.10)
C468  FORMAT(/12H PIVOTAL ROW/)
476  FORMAT(43H WARNING-MATRIX SINGULAR OR ILL CONDITIONED)
C480  FORMAT(5H FAC=,I20.10)
      RETURN
      END

```

*****

*

```

SUBROUTINE RESOL(NMAX,KEY,NEQUAT,LVMAX,NODMAX,LNOMAX,NELZ,
1      MDF,LNODS,KHED,LHED,NOPP,NCOD,PVKOL,ELSTIF,R1,

```

```

2          BC,EQ,QQ)
C
*
*****
*   The case of a resolution only means that only the right hand
*   side terms need be changed. This subroutine supposedly
*   does this but it has not been tested.
*****
*
      INTEGER NMAX,NEQUAT,LVMAX,NODMAX,MAXEL
      PARAMETER (MAXEL=10)
C
      INTEGER
1  ND1,KROW,NELZ,KRW,LCOL,NELL,NLL,LNOMAX,KEY,KPIVRO,I,K,L
2  LPIVCO,NLP,KRO,LVABZ,NSPL,MDF(NODMAX),LNODS(LNOMAX,NELZ)
3  KHED(NMAX),LHED(NMAX),NOPP(NODMAX),NCOD(NEQUAT),LNODZ
C
      REAL*8
1  PIVOT,RHS,PVKOL(NMAX),ELSTIF(LVMAX,LVMAX),R1(NEQUAT)
2  BC(NEQUAT),EQ(NMAX,NMAX),QQ(NMAX),ERSHU(LNOMAX)
3  ERSHV(LNOMAX),ERSHW(LNOMAX),PRES(MAXEL),DIVERG(MAXEL)
C
      DO 2 NELL=1,NELZ
        CALL ABFIND(LVABZ,ELSTIF,NELL,R1,LNOMAX,LNODS,NELZ,NOPP,
1          LVMAX,NSPL,MDF,NODMAX,LNODZ,PRES,DIVERG)
2  CONTINUE
C  WRITE(NLP,476)ITIME
  WRITE(NLP,472)
  DO 4 I=1,NEQUAT
    WRITE(NLP,464)NCOD(I),BC(I)
    IF(NCOD(I).LT.1)GO TO 4
    R1(I)=BC(I)
4  CONTINUE
  DO 12 NLL=1,NEQUAT
    IF(KEY.LE.0) GO TO 14
    READ(UNIT=ND1,REC=KEY)
1  KRO,LCOL,LPIVCO,(LHED(L),QQ(L),L=1,LCOL)
2, KROW,PIVOT,KPIVRO,(PVKOL(K),KHED(K),K=1,KROW)
    KEY=KEY-1
    PVKOL(KPIVRO)=0
    WRITE(NLP,408)KROW,PIVOT,KPIVRO
    WRITE(NLP,412)(PVKOL(K),K=1,KROW)
    WRITE(NLP,416)(KHED(K),K=1,KROW)
    RHS=R1(KRO)/PIVOT
    R1(KRO)=RHS
    DO 8 K=1,KROW
      KRW=IABS(KHED(K))
      IF(NCOD(KRW).GT.0. AND .KHED(K).GT.0)GO TO 8

```

```

      R1(KRW)=R1(KRW)-PVKOL(K)*RHS
8      CONTINUE
C      WRITE(NLP,400)
      WRITE(NLP,404) (I,R1(I),I=1,NEQUAT)
12     CONTINUE
14     CONTINUE
      CALL BACSUB(NEQUAT,EQ,BC,R1,NCOD,NODMAX,NOPP,MDF,ND1)
C400   FORMAT(/18H RIGHT HAND VECTOR/)
404    FORMAT(I5,E20.10)
408    FORMAT(14H DISC CONTENTS/6H KROW=,I5,7H PIVOT=,E20.10
1, 8H KPIVRO=,I5)
412    FORMAT(5E20.10)
416    FORMAT(10I5)
464    FORMAT(I5,E20.10)
472    FORMAT(20H BOUNDARY CONDITIONS)
C476   FORMAT(/28H RIGHT HAND SIDE COMPLETE AT,I5,6H SECS.//)
      RETURN
      END

```

```

*
*****
*

```

```

      SUBROUTINE BACSUB(NEQUAT,SK,BC,R1,NCOD,NODMAX,NOPP,MDF,ND1
1,      NMAX,KEY,KHED,LHED,PVKOL,QQ,SLANUM,SLAMAS)
C
      INTEGER NMAX
C
      INTEGER
1  LCOL,NEQUAT,LCO,NLP,KRO,IDF,L,IV,ND1,KROW,NODMAX,MASTER
2,  I,J,K,LPIVCO,KPIVRO,KEY,KHED(NMAX),LHED(NMAX)
3,  MDF(NODMAX),NCOD(NEQUAT),NOPP(NODMAX),SLANUM
4,  SLAMAS(2,50)
C
      REAL*8
1  PIVOT,GASH,PVKOL(NMAX),R1(NEQUAT)
2,  BC(NEQUAT),SK(NMAX*NMAX),QQ(NMAX)

```

```

*
*****
*
* Subroutine to perform the back-substitution portion of the solution
* technique. Since the diagonals of the [K] matrix have been given
* the value of unity and the right hand vector the prescribed value
* (where applicable) and all other row entries have been set to zero,
* the dof vector, SK,is first fed the boundary conditions (from BC)
* and then further stacked using the disc stored pivotal rows.
*****
*
*
*****

```

* Give the SK vector the boundary condition values.

*****

*

DO 4 I=1, NEQUAT

SK(I)=BC(I)

4 CONTINUE

*

*****

* Step backwards through records until back at the start of the

* WRITES; then go to output current results.

*****

*

DO 32 IV=1, NEQUAT

IF (KEY.LE.0) GO TO 34

READ (UNIT=ND1, REC=KEY)

1 KRO, LCOL, LPIVCO, (LHED(L), QQ(L), L=1, LCOL)

2, KROW, PIVOT, KPIVRO, (PVKOL(K), KHED(K), K=1, KROW)

KEY=KEY-1

C WRITE(NLP, 404)

C WRITE(NLP, 408) KRO, LCOL, LPIVCO

C WRITE(NLP, 408) (LHED(L), L=1, LCOL)

C WRITE(NLP, 412) (QQ(L), L=1, LCOL)

LCO=IABS(LHED(LPIVCO))

IF (NCOD(LCO).GT.0) GO TO 24

GASH=0.0

QQ(LPIVCO)=0.0

DO 16 L=1, LCOL

GASH=GASH-QQ(L)*SK(IABS(LHED(L)))

16 CONTINUE

SK(LCO)=R1(KRO)+GASH

GO TO 32

24 CONTINUE

NCOD(LCO)=1

32 CONTINUE

C WRITE(NLP, 416)

34 CONTINUE

*

*****

* SK now has the values of the unknown dofs. Output the results.

*****

*

DO 36 L=1, NODMAX

J=NOPP(L)-1

IDF=MDF(L)

DO 40 I=1, SLANUM

*

*****

* If the current node is a slave, then give its SK entry the SK

* entry of its master

*****

```

      IF (L.EQ.SLAMAS(1,I)) THEN
        MASTER = SLAMAS(2,I)
        SK(J+1) = SK(NOPP(MASTER))
        GO TO 45
      ENDIF
40      CONTINUE
45      CONTINUE
      WRITE(6,420) L,(SK(J+1),I=1,IDF)
36      CONTINUE
C404     FORMAT(14H TAPE CONTENTS)
C408     FORMAT(10I5)
C412     FORMAT(5E20.10)
C416     FORMAT(/8H RESULTS/)
420      FORMAT(15.6E18.10)
      RETURN
      END

```

*****

```

      SUBROUTINE ABFIND(LVABZ,ELSTIF,WELL,R1,LNOMAX,LNODS,WELZ,NOPP,
1          LVMAX,NSPL,MDF,NODMAX,LNODZ,PRES,DIVERG)

```

```

      INTEGER MAXEL
      PARAMETER (MAXEL=10)

```

```

      INTEGER
1      LNODS(LNOMAX,WELZ),NOPP(LNOMAX),KPOIN,INODZ,LNOMAX,ITOTU
2.  LVMAX,WELL,WELZ,LVABZ,I,J,NSPL,NDOF,MDF(NODMAX),POINT
3.  NODMAX,LNODZ

```

```

C      REAL*8

```

```

1      ELSTIF(LVMAX,LVMAX),R1(LVMAX+WELZ),ERHS(30),PRES(MAXEL)
2.  DISDIV,DIVERG(MAXEL)

```

C

*

*****

* This subroutine calls the [k] matrices from the disc where the
 * element stiffness generator has written them. It also calls the
 * element loads (i.e., the right hand side terms) and places these
 * loads into the global right hand side vector.

*****

*

```

      READ(1,REC=WELL) NSPL,LVABZ,DISDIV,
1          ((ELSTIF(I,J),I=1,LVABZ),J=1,LVABZ)

```

*

*****

* Read the element level loads from the disc.

*****

*

READ(S,REC=WELL+WELZ) LNODZ, LVABZ, (ERHS(I), I=1, LVABZ), PRES(WELL)

*

*****

* Assemble these loads into the global right hand side vector.

* POINT essentially gives the first dof location in the elemental RHS

* vector of the current node whereas ITOTU gives the first dof of the

* node in the global RHS vector.

*****

*

DIVERG(WELL) = DISDIV

DO 500 INODZ = 1, LNODZ

IF (INODZ.EQ.9) GOTO 500

KPOIN = IABS(LNODS(INODZ, WELL))

ITOTU = NOPP(KPOIN)-1

NDOF = MDF(KPOIN)

IF (INODZ.LE.8) POINT = (3+INODZ) - 2

IF (INODZ.GT.9) POINT = INODZ + 15

DO 505 I = 1, NDOF

—KI(ITOTU+I) = R1(ITOTU+I)+ERHS(POINT - 1 + I)

505 CONTINUE

500 CONTINUE

RETURN

END

*

*****

*

SUBROUTINE TOLREL(NITERA, NEQUAT, THETA, TOLERA, VARB1, LNODS,

1 NOPP, NELZ, LNOMAX, NODMAX, NCHECK, MDF, COORD, NCORD, NMAX, LNODZ,

2 PRES1, DIVERG, LTYPE)

IMPLICIT LOGICAL (A-Z)

SAVE

C

INTEGER MAXEL

PARAMETER (MAXEL=10)

INTEGER

1 NITERA, NEQUAT, NCONV, LTOLA, ITOTV, LNOMAX, NELZ, INOD, IC, NCORD

2 , NCHECK, NOD, I, NODMAX, KOUNT, NDOF, LNODS(LNOMAX, NELZ), NOPP(NODMAX)

3 , MDF(NODMAX), MIDS15(5,6), J, K, L, N, KPOIN, P, IDF, NMAX, LNODZ

4 , M1, M2, M3, M4, LTYPE(NELZ)

C

REAL*8

1 THETA, TOLERA, CANLA, CANGE, VARB1(NMAX+NMAX), DIVERG(MAXEL),

1 COORD(NODMAX, NCORD), VARB2(2000), PRES1(MAXEL), PRES2(MAXEL)

C

DATA MIDS15/10,1,4,8,5. 11,2,3,7,6. 12,1,5,6,2.
 1 13,4,8,7,3. 14,1,2,3,4. 15,5,6,7,8/

*****

- This subroutine checks for the convergence of the variables to
- within a specified tolerance limit. If the solutions are not
- converged, it updates the nodal DOFs using the corrections
- stored as the solution dof vector.

•

• VARB1 = current dof vector
 • VARB2 = previous dof vector

*****

•

CANLA = 0.0
 LTOLA = 0
 NCONV = 1
 NCHECK = 1

•

*****

- If this is a 15 node brick for Navier-Stokes, check that the
- rms value of the sum of elemental discrete divergences is less
- than the given tolerance.

*****

•

IF (LTYPE(NELZ).EQ.144) THEN
 DO 5 I = 1,NELZ
 CANLA = CANLA + DIVERG(I)**2

5

CONTINUE
 CANLA = SQRT(CANLA)
 IF (CANLA.GT.TOLERA) NCONV = 0
 WRITE(6,1000) CANLA
 PRINT*, ' RMS VALUE OF SUM OF ELEMENTAL DISCRETE DIVERGENCE = '

1 CANLA

ELSE

•

*****

- If this is the first time through, no previous variables exist to
- use in the relaxation procedure so simply set the new dof
- vector to the dofs just calculated.

*****

•

IF(NITERA.EQ.1) THEN
 NCONV = 0
 DO 50 ITOTV=1,NEQUAT
 VARB2(ITOTV)=VARB1(ITOTV)
 50 CONTINUE
 DO 66 ITOTV = 1,NELZ
 PRES2(ITOTV) = PRES1(ITOTV)

```

56      CONTINUE
      ELSE
      .
      .....
      * Check to see how different the new solution is from the old.
      * If any of the dof changes more than the tolerance limit set, then
      * trip the non-convergence flag to signify an unacceptable solution
      .....
      .
      DO 10 ITOTV=1,NEQUAT
      IF (ABS(VARB1(ITOTV)).GT.0.0001) THEN
      CANGE=ABS((VARB2(ITOTV)-VARB1(ITOTV))/VARB1(ITOTV))
      IF (CANGE.GT.TOLERA)NCONV=0
      IF (CANLA.LT.CANGE) THEN
      CANLA=CANGE
      LTOLA=ITOTV
      ENDIF
      ENDIF
      CONTINUE
      WRITE(6,2000)LTOLA,CANLA,NITERA
      PRINT=,'LARGEST CHANGE OCCURS AT DOF NUMBER',LTOLA,
      'CHANGE =',CANLA,'AT ITERATION =',NITERA
      ENDIF
      ENDIF
      .
      .....
      * Update the nodal dof using the newly calculated velocity corrections
      * by the equation:
      .
      *  $U(n+1) = U(n) - \text{delta } U(n+1)$ 
      .
      * where n = iteration number
      .....
      .
      DO 30 NOD = 1,NODMAX
      KOUNT = NOPP(NOD)
      NDOF = NDF(NOD)
      IF (NDOF.EQ.3) THEN
      DO 35 I = 4,6
      COORD(NOD,I) = COORD(NOD,I) - VARB1(KOUNT+I-4)
      CONTINUE
      ELSE
      COORD(NOD,4) = COORD(NOD,4) - VARB1(KOUNT)
      ENDIF
      CONTINUE
      IF(NCONV.EQ.1) THEN

```



```

*   Signifies convergence and output of final solution   *
*****
*
*
*****
*   Since the 15 node brick is a hierarchical one, the value actually
*   being solved for in the velocity calculations (at the barycentre
*   and midface nodes) is the difference between a quadratic and a
*   linear interpolation. Rearrange COORD accordingly.
*****
*
      DO 12 I = 1,NELZ
        DO 13 J = 1,LNODZ
          IF (J.EQ.9) GOTO 13
          KPOIN = LNODS(J,I)
          IF(J.LE.8.OR.KPOIN.GT.0) GO TO 12
          K = J-9
          KPOIN = (-1)*KPOIN
          L = NOPP(KPOIN)-1
          M1 = NOPP(IABS(LNODS(MIDS15(2,K),I)))-1
          M2 = NOPP(IABS(LNODS(MIDS15(3,K),I)))-1
          M3 = NOPP(IABS(LNODS(MIDS15(4,K),I)))-1
          M4 = NOPP(IABS(LNODS(MIDS15(5,K),I)))-1
          VARB1(L) = VARB1(L)+((VARB1(M1)+VARB1(M2)+VARB1(M3)
1          +VARB1(M4))*0.25)
13      CONTINUE
12      CONTINUE
*
*****
*   Output the converged solution.
*****
*
      WRITE(6,3000)
      WRITE(6,3002)
      DO 20 L = 1,NELZ
        WRITE(6,430)L,PRES1(L)
20      CONTINUE
      WRITE(6,3001)
      DO 15 L=1,NODMAX
        IDF=MOD(L)
        IF (IDF.EQ.3) THEN
          WRITE(6,420) L,(COORD(L,I),I=4,6)
        ELSE
          WRITE(6,425) L,(COORD(L,I),I=4,7)
        ENDIF
15      CONTINUE
      NCHECK = 0

```

*****

* The calls to NUGNUQ should probably be done here when that routine
* is made to recognize this element.

*****

*

ELSE

WRITE(6,3002)

DO 21 L = 1,NELZ

WRITE(6,430)L,PRES1(L)

21 CONTINUE

ENDIF

*

*****

* Not converged so return with flag for more iterations *

*****

*

1990 FORMAT(/'RMS VALUE OF SUM OF ELEMENTAL DISCRETE DIVERGENCE = '

1 G10.4)

2000 FORMAT(/'LARGEST CHANGE OCCURS AT DOF NUMBER'

1 I3,'CHANGE =',F10.4,'AT ITERATION =',I4)

420. FORMAT(I5,3E18.10)

425 FORMAT(I5,4E18.10)

430 FORMAT(I5,7X,1E18.10)

3000 FORMAT (///'**** FINAL CONVERGED SOLUTION FOLLOWS****'//)

3001 FORMAT (2X,'NODE',8X,'U_VEL',12X,'2ND DOF',13X

1,'3RD DOF',13X,'4TH DOF'///)

3002 FORMAT(//.2X,'ELEMENT',8X,'PRESSURE',/)

RETURN

END

*

*****

*

LOGICAL FUNCTION MEMBER(J)

*

*****

* This function returns a Boolean True value if the argument node

* number is a slave node according to the local nodal numbering

* scheme. It returns a False for a non-slave node.

* The function is used when stacking the NK array for the

* particular case of the no x-acceleration element used with the

* 20 node brick.

*****

*

INTEGER J

C

MEMBER = .FALSE.

IF ((J.EQ.2).OR.(J.EQ.3).OR.(J.EQ.6).OR.(J.EQ.7).OR.

1 (J.EQ.10).OR.(J.EQ.14).OR.(J.EQ.15).OR.(J.EQ.18))THEN

MEMBER = .TRUE.
ENDIF
RETURN
END

Appendix D

VFL3D20

SUBROUTINE VFL3D20(COORD,ELSTIF,IPROP,JPROP,LNODS,LNOMAX,
1 LPOP,NDIM,NEL,NELZ,LVABZ,LNODZ,LTYPE,NODMAX,NITERA,VPROP,
2 VFIX,NODFIX,NEWRES,NDFMAX,NFIX)

- NAME: VFL3D20
- DESCRIPTION: TO COMPUTE THE EFFECTIVE ELEMENT STIFFNESS MATRIX
AND EFFECTIVE LOAD VECTOR FOR THE HIERARCHICAL
FORMULATION USING A 20 AND 8 NODE BRICK ELEMENT
FOR THE FULL FORM OF NAVIER-STOKES EQUATIONS.
NOTATION FAVOURS VISCOUS STEADY THREE DIMENSIONAL
FLOW PROBLEMS. TURBULENCE EFFECTS ARE INCORPORATED
- DATE: 3 RD OCTOBER 1983
- AUTHOR: GURU VEMAGANTI
- AMENDED: MARC DULUDE (DEC 1984)
- Passing In : COORD,IPROP,JPROP,LNODS,LNOMAX,LPOP,NDIM,NEL,NELZ,
LVABZ,LNODZ,LTYPE,NODMAX,NITERA,VPROP,VFIX,NODFIX,
NEWRES,NDFMAX,NFIX
- Passing out-via disc- ELSTIF,ERHSU,ERHSV,ERHSW,LVABZ,LNODZ

***** DICTIONARY OF VARIABLES *****

*****

*BFORX : BODY FORCE IN X DIRECTION
 *BFORY : BODY FORCE IN Y DIRECTION
 *BFORZ : BODY FORCE IN Z DIRECTION
 *CONVTN : CONVECTION TERMS IN THE MOMENTUM EQUATION
 *DERG : GLOBAL DERIVATIVE OF THE SHAPE FUNCTION
 *DERL : LOCAL DERIVATIVE OF THE SHAPE FUNCTION
 *DERSHX : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. X
 *DERSHY : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Y
 *DERSHZ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Z
 *DERSXI : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. X FOR
 THE 'I'TH ROW
 *DERSXJ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. X FOR
 THE 'J'TH COLUMN
 *DERSYI : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Y FOR
 THE 'I'TH ROW
 *DERSYJ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Y FOR
 THE 'J'TH COLUMN

*DERSZI : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Z FOR
 * THE 'I'TH ROW
 *DERSZJ : DERIVATIVE OF THE SHAPE FUNCTION W.R.T. Z FOR
 * THE 'J'TH COLUMN
 *DIFSON : DIFFUSION TERM IN THE MOMENTUM EQUATION
 *DVOLUM : RATIO OF THE VOLUME OF AN ELEMENT TO THAT OF AN UNIT BRICK
 *DYNVIS : MOLECULAR DYNAMIC VISCOSITY OF THE FLUID
 *ELXYZ : LOCAL COORDINATES OF THE CURRENT ELEMENT
 *ELXYZT : GLOBAL COORDINATES OF THE CURRENT ELEMENT
 *ERHSU : RIGHT HAND SIDE TERM FOR U MOMENTUM EQUATION
 *ERHSV : RIGHT HAND SIDE TERM FOR V MOMENTUM EQUATION
 *ERHSW : RIGHT HAND SIDE TERM FOR W MOMENTUM EQUATION
 *IGAUS : COUNTER FOR THE GAUSS POINT
 *IPCON : COUNTER FOR THE ROW IN THE PRESSURE LOOP
 *IPROP : NUMBER OF THE ITEM IN THE PROPERTY LIST
 *IROWP : COUNTER FOR THE ROW OF THE CONTINUITY EQUATION
 *IROWU : COUNTER FOR THE ROW OF U COMPONENT OF VELOCITY
 *IROWV : COUNTER FOR THE ROW OF V COMPONENT OF VELOCITY
 *IROWW : COUNTER FOR THE ROW OF W COMPONENT OF VELOCITY
 *JCOLP : COUNTER FOR THE COLUMN OF THE PRESSURE TERMS
 * IN THE MOMENTUM EQUATIONS
 *JCOLU : COUNTER FOR THE COLUMN OF U COMPONENT OF VELOCITY
 *JCOLV : COUNTER FOR THE COLUMN OF V COMPONENT OF VELOCITY
 *JCOLW : COUNTER FOR THE COLUMN OF W COMPONENT OF VELOCITY
 *JPCOM : COUNTER FOR THE COLUMN IN COMPUTING THE PRESSURE TERMS
 *JPROP : NUMBER OF THE PROPERTY LIST
 *KPGIN : FLAG FOR THE GLOBAL NODE NUMBERING
 *LCOUNT : COUNTER FOR THE TYPE OF THE FLOW, I.E. 0 FOR INVISCID FLOW
 * 1 FOR LAMINAR FLOW 2 FOR VISCOUS TURBULENT FLOW
 *LNODS : ARRAY RELATING LOCAL TO GLOBAL NODE NUMBERING
 *LVABZ : TOTAL NUMBER OF DEGREES OF FREEDOM IN AN ELEMENT
 *MIDNOD : NUMBER OF MIDSIDE NODES
 *MIDS : ARRAY OF THE MID-SIDE NODES IN THE ELEMENT
 *ND : A COUNTER FOR THE NODE
 *NDIM : NUMBER OF DIMENSIONS OF THE PROBLEM
 *NDFMAX : MAXIMUM NO. OF DOF FOR A NODE
 *NEL : NUMBER OF THE CURRENT ELEMENT BEING ASSEMBLED
 *NELZ : TOTAL NUMBER OF ELEMENTS IN THE MESH
 *NEWRES : NO. OF NEW RIGHT HAND SIDES EXPECTED IN A RE-SOLUTION.
 * REQUIRED FOR DIMENSIONING OF VFIX
 *NFIX : NO. OF NODES WITH PRESCRIBED VALUES
 *NGAUS : NUMBER OF GAUSS POINTS
 *NITERA : COUNTER FOR THE NUMBER OF ITERATION
 *NODFIX : ARRAY OF NODE NUMBERS FOR PRESCRIBED VALUE NODES
 *P : TEMPORARY ARRAY OF SHAPE FUNCTIONS
 *POIN : ARRAY GIVING THE GLOBAL COORDINATES OF THE CURRENT GAUSS PT
 *PRCONT : COUNTER TO LOCATE THE CONTINUITY EQUATION IN THE F.S.M.
 *PSCHEN : FLAG TO DETERMINE THE INTEGRATION SCHEME FOR THE PRESSURE

```

*          TERMS
*RHOIN    : DENSITY OF THE FLUID
*RNST(1-9): ADDITIONAL TERMS IN THE MOMENTUM EQUATIONS
*          BECAUSE OF THE REYNOLDS STRESSES
*SHAFUN   : SHAPE FUNCTION VALUE
*SHAPI    : SHAPE FUNCTION VALUE AT THE CURRENT NODE
*TURVDX   : DERIVATIVE OF THE TURBULENT VISCOSITY W.R.T. X
*TURVDY   : DERIVATIVE OF THE TURBULENT VISCOSITY W.R.T. Y
*TURVDZ   : DERIVATIVE OF THE TURBULENT VISCOSITY W.R.T. Z
*TURVIS   : TURBULENT VISCOSITY
*TVISC    : VALUE OF THE TURBULENT VISCOSITY AT A NODAL POINT
*UMAX     : MAXIMUM VELOCITY IN THE FLOW DOMAIN
*UVELY    : U COMPONENT OF VELOCITY
*VFIX     : ARRAY OF PRESCRIBED VALUES
*VISLAM   : MOLECULAR KINEMATIC VISCOSITY OF THE FLUID
*VSCHM    : FLAG TO DETERMINE THE INTEGRATION SCHEME FOR THE VELOCITY
*          TERMS
*VVELY    : V COMPONENT OF VELOCITY
*WVELY    : W COMPONENT OF VELOCITY
*IGAUS    : THE LOCAL X-COORDINATE AT THE GAUSS POINT
*XITA     : THE LOCAL COORDINATES OF THE GAUSS POINTS
*YGAUS    : THE LOCAL Y-COORDINATE AT THE GAUSS POINT
*ZGAUS    : THE LOCAL Z-COORDINATE AT THE GAUSS POINT

```

*****

IMPLICIT LOGICAL (A-Z)

INTEGER

```

& IPROP, JPROP, LCOUNT, LNOMAX, LPOP, NDIM, NEL, NELZ, LVABZ, LNODZ, VSCHM,
& NODMAX, NITERA, INODZ, IEVAB, JEVA, IGAUS, LTYPE, NGAUS, I, J,
& KPOIN, ICON, IROWU, IROWV, IROWW, IROWP, JCOLU, JCOLV, JCOLW, JCOLP, NSPL,
& PSCHM, JCON, IPCON, JPCON, LNODS (LNODZ, NELZ), NEWRHS, NDFMAX, NFIX,
& NODFIX(NFIX)

```

REAL*8

```

& VISLAM, RHOIN, BFORX, BFORV, BFORZ, DVOLUM, UVELY, VVELY, WVELY,
& RNST1, RNST2, RNST3, RNST4, RNST5, RNST6, RNST7, RNST8, RNST9, DIFSON,
& CONVTN, TURVDX, TURVDY, TURVDZ, TURVIS, VOLUME, UMAX,
& COORD(NODMAX, 7), ELSTIF(LVABZ, LVABZ), XITA(3), ERHSU(20),
& ERHSV(20), ERHSW(20), VPROP(IPROP, JPCP), SHAFUN(20), WEIGP(30),
& XGAUS(20), YGAUS(20), ZGAUS(20), ELXYZI(3, 20), P(20),
& DERG(3, 20), DERSHX(20), DERSHY(20), DERSHZ(20),
& TVISC(400), DERSXI, DERSYI, DERSZI, DYNVIS,
& DERSXJ, DERSYJ, DERSZJ, SHAPI, RADIUS(20), DRTRVC(20),
& VFIX(NEWRHS, NDFMAX, NFIX), HIGHRAT, LOWRAT, RAT

```

*****

```

*   Define the element properties required as originally read
*   from the input file

```

*****

```

HIGHROT = 0.0
LOWROT = 0.0
DYNVIS = VPROP(1,LPOP)
RHOIN = VPROP(2,LPOP)
BFORX = VPROP(3,LPOP)
BFORY = VPROP(4,LPOP)
BFORZ = VPROP(5,LPOP)
LCOUNT = VPROP(9,LPOP)
VSCHM = VPROP(10,LPOP)
PSCHM = VPROP(11,LPOP)
VISLAM = DYNVIS/RHOIN
NSPL = VPROP(12,LPOP)

```

```

*****

```

```

* Assign the nodes and their coordinates for the current
* element. Modify the coordinates for the midside nodes
*****

```

```

IF(NITERA.EQ.1) CALL INIT(COORD,NODMAX,NEWRHS,NDFMAX,NFIX,
1 VFIX,NODFIX,LNODS,LNODZ,NELZ)
CALL MIDSIDH(COORD, LNOMAX, LNODS, LNODZ,LTYPE,NEL,NELZ,
1 ELXYZT,NDIM,NODMAX,12)

```

```

*****

```

```

* Initialization of the right hand side vectors and the element
* stiffness matrix
*****

```

```

DO 10 INODZ = 1, LNODZ
  ERHSU(INODZ) = 0.0
  ERHSV(INODZ) = 0.0
  ERHSW(INODZ) = 0.0
10 CONTINUE
DO 20 IEVAB = 1, LVABZ
  DO 20 JEVAB = 1, LVABZ
    ELSTIF(IEVAB,JEVAB) = 0.0
20 CONTINUE

```

```

*****

```

```

* Enter the loop for the volume integration for the velocity
* Gaussian scheme
*****

```

```

*****
*   Evaluate the weights and distances for the Gaussian integration
*****
*
      CALL GAUS3D(VSCHEM,NGAUS,XGAUS,YGAUS,ZGAUS,WEIGP)
      DO 100   IGAUS = 1,NGAUS
*
*****
*   Assign the values of the local coordinates of the current
*   Gauss point
*****
*
      XITA (1) = XGAUS(IGAUS)
      XITA (2) = YGAUS(IGAUS)
      XITA (3) = ZGAUS(IGAUS)
*
*****
*   Evaluate the shape fuctions, shape function derivatives, jacobian
*   of transformation, jacobian inverse, etc. using the BRICKH
*   subroutine. The subroutine calculates the above values using a
*   Hierarchical 20 node brick which uses linear interpolation for the
*   corner nodes (which the pressure formulation uses exclusively) and
*   parabolic interpolation for the velocities (which use all 20 nodes)
*****
*
      CALL BRICKH(LNODZ,LVABZ,LNOMAX,NEL,NELZ,XITA,P.
1      ELXYZT,DERG,VOLUME)
      DVOLUM = VOLUME*WEIGP(IGAUS)
*
*****
*   Assign the values of the shape functions and the derivatives
*   to the analagous terms for the current Gauss point.
*****
*
      DO 30 INODZ = 1,LNODZ
        SHAFUN(INODZ) = P (INODZ)
        DERSHX(INODZ) = DERG (1,INODZ)
        DERSHY(INODZ) = DERG (2,INODZ)
        DERSHZ(INODZ) = DERG (3,INODZ)
30    CONTINUE
*
*****
*   Check for the type of the flow. In case of the laminar and
*   inviscid flows, do not add the eddy viscosity contributions
*
*   If LCOUNT = 0 ==> inviscid
*               = 1 ==> laminar
*               >= 2 ==> turbulent

```



```

*****
*
      IF(LCOUNT.LT.2) THEN
*
*****
*   Initialize the three components of velocity i.e., u,v,and w
*   for the laminar and inviscid flows.
*****
*
      UVELY = 0.0
      VVELY = 0.0
      WVELY = 0.0
*
*****
*   Evaluate the velocities at the Gauss point using the previous
*   nodal velocities for the inviscid and laminar flows.
*****
*
      DO 35 INODZ = 1, LNODZ
          KPOIN = IABS(LNODS (INODZ,NEL))
          UVELY = UVELY+(COORD(KPOIN,5)*SHAFUN(INODZ))
          VVELY = VVELY+(COORD(KPOIN,6)*SHAFUN(INODZ))
          WVELY = WVELY+(COORD(KPOIN,7)*SHAFUN(INODZ))
35      CONTINUE
      ELSE
*
*****
*   Call a subroutine to compute the eddy viscosity values at the
*   nodal points of the element.
*****
*
          IF (LCOUNT.EQ.2) THEN
              CALL LAUFTURB(COORD, LNODS, LNODZ, NEL, TVISC, NELZ, NODMAX,
1              RADIUS, NITERA)
          ENDIF
*
*****
*   Initialize the three velocity components (i.e., u,v,and w),
*   the coefficient of eddy viscosity and its derivatives
*   for the turbulent flow case
*****
*
      UVELY = 0.0
      VVELY = 0.0
      WVELY = 0.0
      TURVIS = 0.0
      TURVDX = 0.0
      TURVDY = 0.0

```

TURVDZ = 0.0

```

*
*****
*   Evaluate the velocities, the turbulent viscosity and its
*   derivatives at the Gauss point using the previous
*   nodal velocities for the turbulent flow cases
*****

```

```

*
      DO 40 INODZ = 1, LNODZ
        KPOIN = IABS(LNODS (INODZ, NEL))
        UVELY = UVELY + (COORD(KPOIN, 5) * SHAFUN(INODZ))
        VVELY = VVELY + (COORD(KPOIN, 6) * SHAFUN(INODZ))
        WVELY = WVELY + (COORD(KPOIN, 7) * SHAFUN(INODZ))
        TURVIS = TURVIS + (TVISC(INODZ) * SHAFUN(INODZ))
        TURVDX = TURVDX + (TVISC(INODZ) * DERSHX(INODZ))
        TURVDY = TURVDY + (TVISC(INODZ) * DERSHY(INODZ))
        TURVDZ = TURVDZ + (TVISC(INODZ) * DERSHZ(INODZ))
40    CONTINUE
      ENDIF

```

```

*
*****
*   Velocities as well as the eddy viscosity and its derivatives
*   have been calculated at the current Gauss point
*****

```

```

*
*****
*   Include the body forces on the right hand side
*****

```

```

*
      DO 55 INODZ=1, LNODZ
        ERHSU(INODZ) = ERHSU(INODZ) + SHAFUN(INODZ) * DVOLUM * BFORX
        ERHSV(INODZ) = ERHSV(INODZ) + SHAFUN(INODZ) * DVOLUM * BFORY
        ERHSW(INODZ) = ERHSW(INODZ) + SHAFUN(INODZ) * DVOLUM * BFORZ
55    CONTINUE

```

```

*
*****
*   Start the looping over the element for the left hand side terms
*
*****

```

```

      DO 70 ICON=1, 20

```

```

*
*****
*   Find the row and column in the stiffness matrix into which the
*   calculated quantities sum. There are four DOFs for each of the
*   corner nodes and three for the midside nodes. Rows correspond
*   to the x, y, and z momentum equations and the continuity equation.

```

- * (numbered 1,2,3, and 4 respectively) and the column numbers
- * designate whether to multiply into the u,v,w, or p terms
- * (corresponding to 1,2,3 and 4 respectively)

*****

```

      IF(ICON.LE.8) THEN
        IROWU = (4*ICON)-3
        IROWV = IROWU+1
        IROWW = IROWU+2
      ELSE
        IROWU = (3*ICON)+6
        IROWV = IROWU+1
        IROWW = IROWU+2
      ENDIF

```

*****

- * Evaluate the 'I'th shape function and its derivatives for the
- * 'I'th row.

*****

```

      SHAPI = SHAFUN(ICON)
      DERSXI = DERSHX(ICON)
      DERSYI = DERSHY(ICON)
      DERSZI = DERSHZ(ICON)

```

*****

- * Counter for the columns as described above

*****

```

      DO 80 JCON = 1,20
        IF(JCON.LE.8) THEN
          JCOLU = (4*JCON)-3
          JCOLV = JCOLU+1
          JCOLW = JCOLU+2
        ELSE
          JCOLU = (JCON+3)+6
          JCOLV = JCOLU+1
          JCOLW = JCOLU+2
        ENDIF

```

*****

- * Evaluate the 'J'th shape function derivatives for the
- * 'J'th column.

*****

```

      DERSIJ = DERSHX(JCON)
      DERSYJ = DERSHY(JCON)

```

DERSZJ = DERSHZ(JCON)

*

*****

* Exclude diffusion terms in the case of an inviscid flow. For the
* laminar case, evaluate quantities using only dynamic viscosities.
* For the turbulent case, use eddy viscosities as well.

*****

*

IF(LCOUNT.EQ.0) THEN

DIFSON = 0.0

ELSE

DIFSON = (DERSXI*DERSXJ + DERSYI*DESYJ + DERSZI*DERSZJ)

1 *VISLAM*DVLUM

ENDIF

CONVTN = (UVELY*DERSXJ + VVELY*DESYJ + WVELY*DERSZJ)

1 *SHAPI*DVLUM

*

*****

* Include the diffusion and convection terms in the momentum
* equations (x,y, and z) for all types of flow.

*****

*

C

RAT = ABS(CONVTN/DIFSON)

IF(RAT.GT.HIGHRAT) THEN

HIGHRAT = RAT

ELSEIF (RAT.LT.LOWRAT) THEN

LOWRAT = RAT

ENDIF

C

ELSTIF(IROWU,JCOLU) = ELSTIF(IROWU,JCOLU) + DIFSON + CONVTN

ELSTIF(IROWV,JCOLV) = ELSTIF(IROWV,JCOLV) + DIFSON + CONVTN

ELSTIF(IROWW,JCOLW) = ELSTIF(IROWW,JCOLW) + DIFSON + CONVTN

IF(LCOUNT.GE.2) THEN

*

*****

* Evaluate the Reynolds stress terms which arise because of
* turbulence.

*****

*

1 RNST1 = SHAPI*DVLUM*((TURVDX*2.*DERSXJ)+(TURVDY*DESY
J*1.)+(TURVDZ*DERSZJ*1.))

RNST2 = SHAPI*DESYJ*TURVDY*DVLUM

RNST3 = SHAPI*DESYJ*TURVDZ*DVLUM

RNST4 = SHAPI*DESYJ*TURVDX*DVLUM

1 RNST5 = SHAPI*DVLUM*((TURVDX*DESYJ*1.)+(TURVDY*2.*DE
RSTJ)+(TURVDZ*DERSZJ*1.))

RNST6 = SHAPI*DESYJ*TURVDZ*DVLUM

```

RNST7 = SHAPI*DERSZJ*TURVDX*DVOLUM
RNST8 = SHAPI*DERSZJ*TURVDY*DVOLUM
RNST9 = SHAPI*DVOLUM*((TURVDX+DERSXJ*1.)+(TURVDY+DERSY
1          J*1.)+(TURVDZ+DERSZJ*2.))

```

```

*
*****
*   Include the Reynolds stress terms in the element stiffness matrix.
*****

```

```

ELSTIF(IROWU,JCOLU) = ELSTIF(IROWU,JCOLU)-RNST1
ELSTIF(IROWU,JCOLV) = ELSTIF(IROWU,JCOLV)-RNST2
ELSTIF(IROWU,JCOLW) = ELSTIF(IROWU,JCOLW)-RNST3
ELSTIF(IROWV,JCOLU) = ELSTIF(IROWV,JCOLU)-RNST4
ELSTIF(IROWV,JCOLV) = ELSTIF(IROWV,JCOLV)-RNST5
ELSTIF(IROWV,JCOLW) = ELSTIF(IROWV,JCOLW)-RNST6
ELSTIF(IROWW,JCOLU) = ELSTIF(IROWW,JCOLU)-RNST7
ELSTIF(IROWW,JCOLV) = ELSTIF(IROWW,JCOLV)-RNST8
ELSTIF(IROWW,JCOLW) = ELSTIF(IROWW,JCOLW)-RNST9

```

```

*
*****
*   Evaluate the diffusion terms arising from the eddy viscosity
*   then include it in the element stiffness matrix.
*****

```

```

1          DIFSON=(DERSXI*DERSXJ+DERSYI*DERSYJ+DERSZI*DERSZJ)*
              (TURVIS)*DVOLUM
ELSTIF(IROWU,JCOLU) = ELSTIF(IROWU,JCOLU)+DIFSON
ELSTIF(IROWV,JCOLV) = ELSTIF(IROWV,JCOLV)+DIFSON
ELSTIF(IROWW,JCOLW) = ELSTIF(IROWW,JCOLW)+DIFSON
ENDIF

```

```

*
*****
*   End the loop for the 'J'th column
*****

```

```

80      CONTINUE

```

```

*****
*   End the loop for the 'I'th row
*****

```

```

70      CONTINUE

```

```

*****
*   End the loop for the velocity Gauss point integration
*****
100     CONTINUE

```

```

*****
*   Enter the loop for the volume integration for the pressure   *
*   Gaussian scheme                                              *
*****
*
      CALL GAUSS3D(PSCHEM,NGAUS,XGAUS,YGAUS,ZGAUS,WEIGP)
      DO 140   IGAUS = 1,NGAUS
*
*****
*   Assign the weights and distances for the Gaussian integration
*   for the current Gauss point.
*****
*
      XITA (1) = XGAUS(IGAUS)
      XITA (2) = YGAUS(IGAUS)
      XITA (3) = ZGAUS(IGAUS)
      CALL BRICKH(LNODZ,LVABZ,LNOMAX,NEL,NELZ,XITA,P,
1      ELXYZT,DERG,VOLUME)
      DVOLUM = VOLUME*WEIGP(IGAUS)
*
      ENDIF
*
*****
*   Assign the values of the shape functions and the derivatives
*   to the analogous terms for the current Gauss point.
*****
*
      DO 150 INODZ = 1,LNODZ
      SHAFUN(INODZ) = P (INODZ)
      DERSHX(INODZ) = DERG (1,INODZ)
      DERSHY(INODZ) = DERG (2,INODZ)
      DERSHZ(INODZ) = DERG (3,INODZ)
150  CONTINUE
*
*****
*   Start the looping over rows and columns for the pressure terms
*   in the momentum equations.
*****
*
      DO 160 IPCON = 1,20
*
*****
*   Set row and column counters as in the velocity integration
*****
*
      IF(IPCON.LE.8) THEN
      IROWU =(4*IPCON)-3
      IROWV = IROWU+1
      IROWW = IROWU+2

```

```

      IROWP = IROWU+3
    ELSE
      IROWU = (3*IPCON)+6
      IROWV = IROWU+1
      IROWW = IROWU+2
    ENDIF

```

```

    .....
    * Evaluate the 'I'th shape function for the 'I'th row.
    .....

```

```

      SHAPI = SHAFUN(IPCON)
      DO 170 JPCON = 1,20
        IF(JPCON.LE.8) THEN
          JCOLU = (4+JPCON)-3
          JCOLV = JCOLU+1
          JCOLW = JCOLU+2
          JCOLP = JCOLU+3
        ELSE
          JCOLU = (JPCON+3)+6
          JCOLV = JCOLU+1
          JCOLW = JCOLU+2
        ENDIF

```

```

    .....
    * Evaluate the 'J'th shape function and its derivatives for the
    * 'J'th column.
    .....

```

```

      DERSIJ = DERSHJ(JPCON)
      DERSYJ = DERSHY(JPCON)
      DERSZJ = DERSHZ(JPCON)

```

```

    .....
    * Include the pressure terms in the momentum equations.
    .....

```

```

      IF(JPCON.LE.8) THEN
        ELSTIF(IROWU,JCOLP)=ELSTIF(IROWU,JCOLP)+SHAPI*DERSIJ*DVOLUM/RHOIN
        / ELSTIF(IROWV,JCOLP)=ELSTIF(IROWV,JCOLP)+SHAPI*DERSYJ*DVOLUM/RHOIN
        ELSTIF(IROWW,JCOLP)=ELSTIF(IROWW,JCOLP)+SHAPI*DERSZJ*DVOLUM/RHOIN
      ENDIF

```

```

    .....
    * Evaluate the continuity equation terms and include them in the
    * element stiffness matrix
    .....

```

```

      IF(IPCON.LE.8) THEN
        ELSTIF(IROWP,JCOLU)=ELSTIF(IROWP,JCOLU)+SHAPI*DVOLUM*DERSIJ
        ELSTIF(IROWP,JCOLV)=ELSTIF(IROWP,JCOLV)+SHAPI*DVOLUM*DERSYJ
        ELSTIF(IROWP,JCOLW)=ELSTIF(IROWP,JCOLW)+SHAPI*DVOLUM*DERSZJ
      ENDIF

*
*****
*   End the loop for the 'J'th column
*****
*
170      CONTINUE
*
*****
*   End the loop for the 'I'th row
*****
*
180      CONTINUE
*
*****
*   End the loop for the pressure Gauss point integration
*****
*
140 CONTINUE
*
*****
*   Check NSPL to determine if any special qualities are required
*   of the element. Available are:
*       NSPL = 1 : Acceleration in the x-wise direction is fixed
*                   to zero
*       = 2 : Cylindrical coordinates taken into account in
*                   the element stiffness matrix entries
*****
*
      IF(NSPL.EQ.1) THEN
        CALL NOIAXL(ELSTIF,LVABZ,NSPL,NEL)
      ELSEIF (NSPL.EQ.2) THEN
        CALL RAGHAVENDER(ELSTIF,NEL,LVABZ)
      ENDIF
C    CALL STIFCHECK(ELSTIF,LVABZ,ERHSU,ERHSV,ERHSW)
      PRINT*,'FOR NEL=',NEL,'HIGHRAT=',HIGHRAT,'LOWRAT=',LOWRAT
      IF(NSPL.NE.1) THEN
        WRITE(UNIT=1,REC=NEL) NSPL,LVABZ,
          ((ELSTIF(I,J),I=1,LVABZ),J=1,LVABZ)
      ENDIF
      WRITE(UNIT=3,REC=NEL+NELZ) LNODZ,ERHSU,ERHSV,ERHSW
      RETURN
      END

```



```

*****
*
*   SUBROUTINE BRICKH(LNODZ,LVABZ,LNOMAX,NEL,NELZ,XITA,
1      SHAPE,ELXYZ,DERIV,VOL)
*
*****
*   This subroutine computes the Jacobian matrix of the transformation
*   as well as its determinant (which is the volume of the
*   transformation), the inverse of this matrix, and the shape
*   functions for this element (20 node brick) in both local and
*   global coordinates.
*****
*
*   IMPLICIT LOGICAL (A-Z)
C
*   INTEGER
1      LNODZ, LNOMAX, LVABZ, NEL, NELZ, OERR
C
*   REAL*8
*   XITA(3)
*
*****
*   Declarations for the local variables
*****
*
*   REAL*8
1      DERIV(LNODZ,3), ELXYZ(3,LNODZ), J(3,3), JINV(3,3),
2      POIN(3), SHAPE(LNODZ), VOL
*
*****
*   Initialize the error flag
*****
*
*   OERR=0
*
*****
*   The subroutine SFRH will get the shape functions and the shape
*   function derivatives with respect to all local coordinates at
*   the point XITA in local coordinates.
*****
*
*   CALL SFRH(XITA,SHAPE,DERIV)
*
*****
*   Now the Jacobian, which is the product of ELXYZ and the shape
*   function derivatives and is used to transform from local global
*   coordinates, is found.

```

```

*****
*
      CALL JACOBS3D(DERIV,ELXYZ,J,LNODZ)
*
*****
*   The Jacobian is inverted and its determinant, which is the volume
*   of the transformation, is found.
*   If the error flag is equal to 1, we have a negative or zero volume
*   returned from INVJ so output an appropriate error message.
*****
*
      CALL INVJ(J,JINV,VOL,OERR)
      IF (OERR.EQ.1) THEN
        WRITE (6,9020)
        RETURN
      ENDIF

*
*****
*   Next, the Cartesian derivatives are found.
*****
*
      CALL CARTE3D (JINV, DERIV,LNODZ)
*
*****
*   Lastly, the global coordinates at the point XITA are found as the
*   matrix product of ELXYZ and the shape functions
*****
*
      CALL POINTS3D(POIN,SHAPE,ELXYZ,LNODZ)
*
*****
*   Make sure the point XITA is inside the brick by verifying that
*   the local coordinates are all between -1 and +1.
*****
*
      IF ((1.2.LT.ABS(XITA(1))).OR.(1.2.LT.ABS(XITA(2))))
1      .OR.(1.2.LT.ABS(XITA(3)))) THEN
        WRITE(6,9000)
      ENDIF
9000 FORMAT('O','PROGRAM HAS ATTEMPTED TO FIND THE SHAPE FUNCTION',
1      ' OF A POINT OUTSIDE THE BRICK.')
9020 FORMAT ('O','THE DETERMINANT OF THE JACOBIAN FOR THIS ',
1      ' ELEMENT IS NEGATIVE OR ZERO')
      RETURN
      END
*
*****

```

SUBROUTINE SFRH(XITA,SHAPE,DERIV)

*****

* Purpose: To compute the shape functions and their derivatives
* for a 20 node hierarchical brick element.

* Date: 3rd October 1983

*****

REAL*8

1 XITA(3),SHAPE(20),DERIV(3,20),R,S,T,R1,R2,R3,S1,S2,S3
2 T1,T2,T3

C

R=XITA(1)

S=XITA(2)

T=XITA(3)

R1=(1.-R)/2.

NM=20

S1=(1.-S)/2.

T1=(1.-T)/2.

R2=1.-R+R

S2=1.-S+S

T2=1.-T+T

R3=(1.+R)/2.

S3=(1.+S)/2.

T3=(1.+T)/2.

*****

* Calculate the shape functions for the current element.

*****

SHAPE(1)=R1*S1*T1

SHAPE(2)=R3*S1*T1

SHAPE(3)=R3*S3*T1

SHAPE(4)=R1*S3*T1

SHAPE(5)=R1*S1*T3

SHAPE(6)=R3*S1*T3

SHAPE(7)=R3*S3*T3

SHAPE(8)=R1*S3*T3

SHAPE(9)=R2*S1*T1

SHAPE(10)=R3*S2*T1

SHAPE(11)=R2*S3*T1

SHAPE(12)=R1*S2*T1

SHAPE(13)=R1*S1*T2

SHAPE(14)=R3*S1*T2

SHAPE(15)=R3*S3*T2

SHAPE(16)=R1*S3*T2

SHAPE(17)=R2*S1*T3

SHAPE(18)=R3*S2*T3

SHAPE(19)=R2*S3*T3

SHAPE(20)=R1*S2*T3

*

*****

* Calculate the derivatives of the shape functions w.r.t. R

*****

*

DERIV(1, 1)=-S1*T1/2.

DERIV(1, 2) = S1*T1/2.

DERIV(1, 3)= S3*T1/2.

DERIV(1, 4)=-S3*T1/2.

DERIV(1, 5)=-S1*T3/2.

DERIV(1, 6)= S1*T3/2.

DERIV(1, 7)= S3*T3/2.

DERIV(1, 8)=-S3*T3/2.

DERIV(1, 9)=-2.*R*S1*T1

DERIV(1,10)= S2*T1/2.

DERIV(1,11)=-2.*R*S3*T1

DERIV(1,12)=-S2*T1/2.

DERIV(1,13)=-S1*T2/2.

DERIV(1,14)= S1*T2/2.

DERIV(1,15)= S3*T2/2.

DERIV(1,16)=-S3*T2/2.

DERIV(1,17)=-2.*R*S1*T3

DERIV(1,18)= S2*T3/2.

DERIV(1,19)=-2.*R*S3*T3

DERIV(1,20)=-S2*T3/2.

*

*****

* Calculate the derivatives of the shape functions w.r.t. S

*****

*

DERIV(2, 1)=-R1*T1/2.

DERIV(2, 2)=-R3*T1/2.

DERIV(2, 3)=R3*T1/2.

DERIV(2, 4)= R1*T1/2.

DERIV(2, 5)=-R1*T3/2.

DERIV(2, 6)=-R3*T3/2.

DERIV(2, 7)= R3*T3/2.

DERIV(2, 8)= R1*T3/2.

DERIV(2, 9)=-R2*T1/2.

DERIV(2,10)=-2.*S*R3*T1

DERIV(2,11)= R2*T1/2.

DERIV(2,12)=-2.*S*R1*T1

DERIV(2,13)=-R1*T2/2.

DERIV(2,14)=-R3*T2/2.

DERIV(2,15)= R3*T2/2.

```

DERIV(2,16)= R1*T2/2.
DERIV(2,17)=-R2*T3/2.
DERIV(2,18)=-2.*S*R3*T3
DERIV(2,19)= R2*T3/2.
DERIV(2,20)=-2.*S*R1*T3

```

*

*****

* Calculate the derivatives of the shape functions w.r.t. T

*****

*

```

DERIV(3, 1)=-R1*S1/2.
DERIV(3, 2)=-R3*S1/2.
DERIV(3, 3)=-R3*S3/2.
DERIV(3, 4)=-R1*S3/2.
DERIV(3, 5)= R1*S1/2.
DERIV(3, 6)= R3*S1/2.
DERIV(3, 7)= R3*S3/2.
DERIV(3, 8)= R1*S3/2.
DERIV(3, 9)=-R2*S1/2.
DERIV(3,10)=-R3*S2/2.
DERIV(3,11)=-R2*S3/2.
DERIV(3,12)=-R1*S2/2.
DERIV(3,13)=-2.*T*R1*S1
DERIV(3,14)=-2.0*R3*S1*T
DERIV(3,15)=-2.0*T*R3*S3
DERIV(3,16)=-2.*T*R1*S3
DERIV(3,17)= R2*S1/2.
DERIV(3,18)= R3*S2/2.
DERIV(3,19)= R2*S3/2.
DERIV(3,20)= R1*S2/2.

```

```

WRITE(6,*) (SHAPE(I),I=1,NN)

```

```

WRITE(6,*) ((DERIV(I,J),J=1,NN),I=1,3)

```

```

RETURN

```

```

END

```

*****

*

```

SUBROUTINE MIDSIDH(COORD, LNO MAX, LNODS, LNODZ, LTYPE, NEL, NELZ,
1 ELXYZ,NDIM,NODMAX,MIDNOD)

```

* DATE: 23RD OCTOBER 1983

* PURPOSE : TO MODIFY THE COORDINATES OF THE NODES OF A BRICK
 * ELEMENT FOR THE HIERARCHICAL FORMULATION. IT MODI-
 * FIES THE COORDINATES OF THE MIDSIDE NODES IF GIVEN
 * AND KEEPS THEM TO BE ZERO IF THEY ARE NOT GIVEN.

*

```

IMPLICIT LOGICAL (A-Z)

```

```

INTEGER

```

```

1  I, LNODZ, LNOMAX, LTYPE.
2  MIDNOD, MIDS20(3,12), ND, NDIM, NEL, NELZ, NOD.
3  NODE, NODMAX, LNODS(LNOMAX, NELZ)
  REAL*8
1  COORD(NODMAX, 7), ELXYZ(NDIM, LNODZ)
*
**** LIST OF CONNECTIVITY DATA FOR MIDSIDE NODES IN THE 20 NODE
* BRICK. 8 NODE QUAD AND BEAM ARE A SUBSET.
*
  DATA MIDS20/ 9,1,2, 10,2,3, 11,3,4, 12,4,1,
1      13,1,5, 14,2,6, 15,3,7, 16,4,8,
2      17,5,6, 18,6,7, 19,7,8, 20,8,5/
*
**** WE COLLECT THE NODAL COORDINATES FOR THE ELEMENT IN ELXYZ(LNODZ, NDI
*
  DO 20 NOD=1, LNODZ
    NODE = IABS(LNODS(NOD, NEL))
    DO 10 ND = 1, NDIM
      ELXYZ(ND, NOD) = COORD(NODE, ND)
10    CONTINUE
20    CONTINUE
*
* CALCULATE THE CO-ORDINATE VALUES FOR THE EDGES OF THE
* 8 NODE QUAD AND THE 20 NODE BRICK FOR HIERARCHICAL FORMULATION
* IT MODIFIES THE COORDINATES FOR THE EDGES IF THEY ARE SPECIFIED
* AND MAKES THEM ZERO IF THE COORDINATES ARE NOT SPECIFIED.
*
  DO 40 I = 1, MIDNOD
    NODE = IABS(LNODS(MIDS20(1, I), NEL))
    DO 30 ND = 1, NDIM
      IF (ELXYZ(ND, MIDS20(1, I)) .EQ. 5.5E+55) THEN
        ELXYZ(ND, MIDS20(1, I)) = 0.0
        COORD(NODE, ND) = (ELXYZ(ND, MIDS20(2, I)) +
1          ELXYZ(ND, MIDS20(3, I))) * 0.5
      ELSE
        ELXYZ(ND, MIDS20(1, I)) = ELXYZ(ND, MIDS20(1, I)) -
1          ((ELXYZ(ND, MIDS20(2, I)) + ELXYZ(ND, MIDS20(3, I))) / 2.)
      ENDIF
    COORD(NODE, ND) = ELXYZ(ND, MIDS20(1, I))
30    CONTINUE
40    CONTINUE
*
**** ARE ALL NODE COORD. NOW ASSIGNED?
*
  DO 140 NOD=1, LNODZ
    NODE = IABS(LNODS(NOD, NEL))
    DO 130 ND = 1, NDIM

```

```

      IF (ELXYZ(ND,NOD).EQ.5.5E+55) THEN
        PRINT*,'UNASSIGNED COORDINATE IN MIDSID'
        STOP
      ENDIF
130    CONTINUE
140    CONTINUE
*9000  FORMAT(' NODE',6X,'X',12X,'Y',12X,'Z',/)
*9010  FORMAT(1X,I6,3F13.7)
      RETURN
      END
*
*****
*
      SUBROUTINE LAUFTURB(COORD,LNODS,LNODZ,NEL,TVISC,NELZ,NODMAX
1      ,RADIUS,NITERA)
C
      INTEGER I,LNODZ,NELZ,NODMAX,NEL,KPOIN,LNODS(LNODZ,NELZ)
1      NITERA
C
      REAL*8
1      COORD(NODMAX,7),TVISC(LNODZ),RADIUS(20),DIAM,XITA2,RAD
C
      SAVE DIAM
*
*****
*   Find the diameter of the pipe
*****
*
      IF (NITERA.EQ.1) THEN
        DIAM = 0.0
        DO 80 I=1,NODMAX
          RAD=SQRT((COORD(I,2)**2)+(COORD(I,3)**2))
          IF(DIAM.LT.RAD*2.) THEN
            DIAM = RAD*2.
          ENDIF
80      CONTINUE
      ENDIF
*
*****
*   Initialize the turbulent viscosities to an arbitrary figure
*   of 2.22E+22 in double precision
*****
*
      DO 90 I=1,LNODZ
        TVISC(I) = 2.22D+22
90      CONTINUE
*
*****

```

* Assign a node a turbulent viscosity based on Laufer's pipe
 * data. The viscosity is associated with the radial distance
 * of a point.

*****

*

DO 100 I= 1, LNODZ

KPOIN=IABS(LNODS(I,NEL))

RADIUS(I)=SQRT((COORD(KPOIN,2)**2)+(COORD(KPOIN,3)**2))

XITA2=1.-(2.*RADIUS(I)/DIAM)

IF(XITA2.LE.0.1.AND.XITA2.GE.-0.1) THEN

TVISC(I)=0.0

ELSEIF(XITA2.LE.0.22.AND.XITA2.GT.0.10) THEN

TVISC(I)=0.00806

ELSEIF(XITA2.LE.0.275.AND.XITA2.GT.0.22) THEN

TVISC(I)=0.01263

ELSEIF(XITA2.LE.0.35.AND.XITA2.GT.0.275) THEN

TVISC(I)=0.01343

ELSEIF(XITA2.LE.0.475.AND.XITA2.GT.0.35) THEN

TVISC(I)=0.01307

ELSEIF(XITA2.LE.0.60.AND.XITA2.GT.0.475) THEN

TVISC(I)=0.01271

ELSEIF(XITA2.LE.1.1.AND.XITA2.GT.0.60) THEN

TVISC(I)=0.01236

ENDIF

IF(TVISC(I).EQ.2.22E+22) THEN

PRINT*, 'NO VALUE FOR NODE ', KPOIN, ' WITH RADIUS ', RADIUS(I)

1, ' AND XITA2 ', XITA2

ENDIF

100 CONTINUE

*

*****

* Change the absolute eddy viscosity values at the midside nodes
 * to differences away from a linear interpolation between the
 * two bounding corner nodes to remain commensurate with the
 * hierarchical formulation

*****

*

TVISC(9) =TVISC(9) -((TVISC(1)+TVISC(2))/2.)

TVISC(10)=TVISC(10)-((TVISC(2)+TVISC(3))/2.)

TVISC(11)=TVISC(11)-((TVISC(3)+TVISC(4))/2.)

TVISC(12)=TVISC(12)-((TVISC(4)+TVISC(1))/2.)

TVISC(13)=TVISC(13)-((TVISC(1)+TVISC(5))/2.)

TVISC(14)=TVISC(14)-((TVISC(2)+TVISC(6))/2.)

TVISC(15)=TVISC(15)-((TVISC(3)+TVISC(7))/2.)

TVISC(16)=TVISC(16)-((TVISC(4)+TVISC(8))/2.)

TVISC(17)=TVISC(17)-((TVISC(5)+TVISC(6))/2.)

TVISC(18)=TVISC(18)-((TVISC(6)+TVISC(7))/2.)

TVISC(19)=TVISC(19)-((TVISC(7)+TVISC(8))/2.)


```

TVISC(20)=TVISC(20)-((TVISC(8)+TVISC(5))/2.)
RETURN
END

*
*****
*
SUBROUTINE NOXAXL(ELSTIF,LVABZ,NSPL,NEL)
C
  INTEGER
  1 I,LVABZ,IROWU,ICOLU,END,START,PTR(8),NSPL,NEL
C
  REAL*8
  1 ELSTIF(68,68)
C
  DATA PTR /5,9,21,25,36,48,51,60/
*
*****
*   This subroutine modifies the element stiffness matrix to slave
*   downstream nodes to upstream nodes (for u velocities only)
*   to impose the condition of no x-wise acceleration for the element.
*   To do this, the element K matrix is condensed from its original
*   68x68 size to 60x60 by removing the dof corresponding to the
*   u velocity of the slave node.
*****
*
*
*****
*   First, add the columns of the slave dof to the master dof.
*****
*
  DO 100 IROWU = 1,LVABZ
    ELSTIF(IROWU,1) = ELSTIF(IROWU,1)+ELSTIF(IROWU,5)
    ELSTIF(IROWU,13) = ELSTIF(IROWU,13)+ELSTIF(IROWU,9)
    ELSTIF(IROWU,17) = ELSTIF(IROWU,17)+ELSTIF(IROWU,21)
    ELSTIF(IROWU,29) = ELSTIF(IROWU,29)+ELSTIF(IROWU,25)
    ELSTIF(IROWU,42) = ELSTIF(IROWU,42)+ELSTIF(IROWU,36)
    ELSTIF(IROWU,45) = ELSTIF(IROWU,45)+ELSTIF(IROWU,48)
    ELSTIF(IROWU,54) = ELSTIF(IROWU,54)+ELSTIF(IROWU,51)
    ELSTIF(IROWU,66) = ELSTIF(IROWU,66)+ELSTIF(IROWU,60)
100  CONTINUE
*
*****
*   Now, add the rows of the slave dof to the master.
*****
*
  DO 110 ICOLU = 1,LVABZ
    ELSTIF(1,ICOLU) = ELSTIF(1,ICOLU)+ELSTIF(5,ICOLU)
    ELSTIF(13,ICOLU) = ELSTIF(13,ICOLU)+ELSTIF(9,ICOLU)

```

```

        ELSTIF(17,ICOLU) = ELSTIF(17,ICOLU)+ELSTIF(21,ICOLU)
        ELSTIF(29,ICOLU) = ELSTIF(29,ICOLU)+ELSTIF(25,ICOLU)
        ELSTIF(42,ICOLU) = ELSTIF(42,ICOLU)+ELSTIF(36,ICOLU)
        ELSTIF(45,ICOLU) = ELSTIF(45,ICOLU)+ELSTIF(48,ICOLU)
        ELSTIF(54,ICOLU) = ELSTIF(54,ICOLU)+ELSTIF(51,ICOLU)
        ELSTIF(66,ICOLU) = ELSTIF(66,ICOLU)+ELSTIF(60,ICOLU)
110  CONTINUE
*
*****
*   Now, compact the matrix by removing the rows and columns of the
*   slave. Here, END will correspond to the number of columns left
*   in the matrix, LVABZ will be the number of rows left.
*****
*
      END = LVABZ
*
*****
*   There are 8 dofs to remove. START is the pointer to the
*   first position in the matrix where to begin the compaction.
*   PTR is the array of pointers to the dof in the original K
*   matrix that must be removed.
*****
*
*
*****
*   First, remove the slave columns
*****
*
      DO 120 I=1,8
        START = PTR(I) +2 -I
        DO 130 IROWU = 1,LVABZ
          DO 140 ICOLU = START, END
            ELSTIF(IROWU,ICOLU-1) = ELSTIF(IROWU,ICOLU)
140          CONTINUE
130          CONTINUE
            END = END -1
120        CONTINUE
*
*****
*   Now, remove the rows of the slave by repeating the above
*   procedure interchanging IROWU & ICOLU, END & LVABZ.
*****
*
      DO 150 I=1,8
        START = PTR(I) +2 -I
        DO 160 IROWU = START,LVABZ
          DO 170 ICOLU = 1,END
            ELSTIF(IROWU-1,ICOLU) = ELSTIF(IROWU,ICOLU)

```

```

170     CONTINUE
180     CONTINUE
      LVABZ = LVABZ - 1
150     CONTINUE
      WRITE(UNIT=1,REC=NEL) NSPL,LVABZ.
1      ((ELSTIF(IROWU,ICOLU),IROWU=1,LVABZ),ICOLU=1,LVABZ)
      RETURN
      END

```

*****

```

      SUBROUTINE RAGHAVENDER(ELSTIF,NEL,LVABZ)

```

```

****    DATE   : 21 JUNE, 1984

```

```

      REAL*8

```

```

1      ELSTIF(68,68),THETA1(3),THETA2(3),THETA3(3),C,A1,A2
2      C1,C2,C3,C4,C5

```

```

      INTEGER

```

```

1      LVABZ,NEL,I

```

```

      DATA THETA1/30, 20, 10/
      DATA THETA2/20, 10, 0/
      DATA THETA3/25, 15, 5/

```

```

      C = 0.0174532

```

```

      C1 = TAN(C*THETA3(NEL))/2.

```

```

      C2 = TAN(C*THETA3(NEL))/2.

```

```

      C3 = -0.5

```

```

      C4 = -0.5

```

```

      C5 = TAN(C*THETA3(NEL))

```

```

      A1 = TAN(C*THETA1(NEL))

```

```

      A2 = TAN(C*THETA2(NEL))

```

```

      DO 300 I=1,LVABZ

```

```

      IF (NEL.EQ.1) THEN

```

```

      ELSTIF(I,5)=ELSTIF(I,5)+ELSTIF(I,7)*A2+ELSTIF(I,35)*((A2*C4)+C2)

```

```

      ELSTIF(I,9)=ELSTIF(I,9)+ELSTIF(I,11)*A2+ELSTIF(I,41)*(A2*C4+C2)

```

```

      ELSTIF(I,21)=ELSTIF(I,21)+ELSTIF(I,23)*A2+ELSTIF(I,59)*(A2*C4+C2)

```

```

      ELSTIF(I,25)=ELSTIF(I,25)+ELSTIF(I,27)*A2+ELSTIF(I,65)*(A2*C4+C2)

```

```

      ELSTIF(I,2) = ELSTIF(I,2)+ELSTIF(I,35)*C3

```

```

      ELSTIF(I,1)=ELSTIF(I,1)+ELSTIF(I,35)*(C1)

```

```

      ELSTIF(I,14) = ELSTIF(I,14)+ELSTIF(I,41)*C3

```

```

      ELSTIF(I,13)=ELSTIF(I,13)+ELSTIF(I,41)*(C1)

```

```

      ELSTIF(I,18) = ELSTIF(I,18)+ELSTIF(I,59)*C3

```

```

      ELSTIF(I,17)=ELSTIF(I,17)+ELSTIF(I,59)*(C1)

```

```

      ELSTIF(I,30) = ELSTIF(I,30)+ELSTIF(I,65)*C3

```

```

      ELSTIF(I,29)=ELSTIF(I,29)+ELSTIF(I,65)*(C1)

```

```

ELSTIF(I,33) = ELSTIF(I,33)+ELSTIF(I,35)*C5
ELSTIF(I,39) = ELSTIF(I,39)+ELSTIF(I,41)*C5
ELSTIF(I,57) = ELSTIF(I,57)+ELSTIF(I,59)*C5
ELSTIF(I,63) = ELSTIF(I,63)+ELSTIF(I,65)*C5
ELSE
ELSTIF(I,5)=ELSTIF(I,5)+ELSTIF(I,7)*A2+ELSTIF(I,35)*((A2+C4)+C2)
ELSTIF(I,9)=ELSTIF(I,9)+ELSTIF(I,11)*A2+ELSTIF(I,41)*((A2+C4)+C2)
ELSTIF(I,21)=ELSTIF(I,21)+ELSTIF(I,23)*A2+ELSTIF(I,59)*((A2+C4)+C2)
ELSTIF(I,25)=ELSTIF(I,25)+ELSTIF(I,27)*A2+ELSTIF(I,65)*((A2+C4)+C2)
ELSTIF(I,1)=ELSTIF(I,1)+ELSTIF(I,3)*A1+ELSTIF(I,35)*((A1+C3)+C1)
ELSTIF(I,13)=ELSTIF(I,13)+ELSTIF(I,15)*A1+ELSTIF(I,41)*((A1+C3)+C1)
ELSTIF(I,17)=ELSTIF(I,17)+ELSTIF(I,19)*A1+ELSTIF(I,59)*((A1+C3)+C1)
ELSTIF(I,29)=ELSTIF(I,29)+ELSTIF(I,31)*A1+ELSTIF(I,65)*((A1+C3)+C1)
ELSTIF(I,33) = ELSTIF(I,33)+ELSTIF(I,35)*C5
ELSTIF(I,39) = ELSTIF(I,39)+ELSTIF(I,41)*C5
ELSTIF(I,57) = ELSTIF(I,57)+ELSTIF(I,59)*C5
ELSTIF(I,63) = ELSTIF(I,63)+ELSTIF(I,65)*C5
ENDIF
300  CONTINUE
      RETURN
      END

```

*****

```

SUBROUTINE INIT(COORD,NODMAX,NEWRHS,NDFMAX,NFIX,VFIX,NODFIX,LNODS,
1  LNODZ,NELZ)

```

*****

* Subroutine to initialize the degrees of freedom of each node to the
 * value prescribed in the input file fixity list. Default is 0 0
 *****

```

INTEGER NODMAX,I,J,K,NEWRHS,NFIX,NODFIX(NFIX),NODE,LNODZ,NELZ,
1  NDFMAX,LNODS(LNODZ,NELZ),MIDS20(3,12),CORNER1,CORNER2

```

C

```

REAL*8 COORD(NODMAX,7),VFIX(NEWRHS,NDFMAX,NFIX)

```

C

```

DATA MIDS20/ 9,1,2, 10,2,3, 11,3,4, 12,4,1,
1 13,1,5, 14,2,6, 15,3,7, 16,4,8,
2 17,5,6, 18,6,7, 19,7,8, 20,8,5/

```

C

```

DO 10 I=1,NODMAX
  COORD(I,4)=0.0
  COORD(I,5)=0.0
  COORD(I,6)=0.0
  COORD(I,7)=0.0

```

10

```

CONTINUE
DO 20 I=1,NFIX

```

3 of/de 3

MICRO

```

      NODE = NODFIX(I)
      COORD(NODE,4) = VFIX(1,4,I)
      COORD(NODE,5) = VFIX(1,1,I)
      COORD(NODE,6) = VFIX(1,2,I)
      COORD(NODE,7) = VFIX(1,3,I)
20    CONTINUE
*
*****
*   Change absolute velocities at midside nodes to deviations from
*   linear interpolation between corner nodes to keep hierarchical
*   dof values
*****
*
C      DO 30 I=1,NELZ
C          DO 40 J=1,12
C              CORNER1 = LNODS(MIDS20(2,J),I)
C              CORNER2 = LNODS(MIDS20(3,J),I)
C              NODE = LNODS(MIDS20(1,J),I)
C              DO 50 K=5,7
C                  COORD(NODE,K)=COORD(NODE,K)-((COORD(CORNER1,K) +
C 1              COORD(CORNER2,K))/2.)
C50    CONTINUE
C40    CONTINUE
C30    CONTINUE
      RETURN
      END
*
*****
*
      SUBROUTINE STIFCHECK(ELSTIF,LVABZ,ERHSU,ERHSV,ERHSW)
C
C      INTEGER LVABZ,IROW,JCOL,PTR
C
C      REAL*8 ELSTIF(LVABZ,LVABZ),SUM(32),SUMU(32),SUMV(32),SUMW(32)
1      SUMP(32),ERHSU(68),ERHSV(68),ERHSW(68)
*
*****
*   Subroutine to sum the individual rows of the element stiffness
*   matrix. The sum should be zero for all rows just as it is
*   for the rigid body displacement for the stress case.
*
*   Only the first 32 rows and columns are considered since they
*   are the only ones which actually provide for velocity and
*   pressure gradients of zero (remember hierarchical formulation)
*
*   Normally this subroutine is used only in the debugging phase
*   and the call to it in VFL3D20 is commented out since the information
*   output is not required.

```

*****

*

```

DO 1 IROW=1,LVABZ
  WRITE(6,FMT='(I2,8G10.2)') IROW,(ELSTIF(IROW,JCOL),JCOL=1,8)
  WRITE(6,FMT='(3X,12G10.2)') (ELSTIF(IROW,JCOL),JCOL=9,20)
  WRITE(6,FMT='(3X,12G10.2)') (ELSTIF(IROW,JCOL),JCOL=21,32)
  WRITE(6,FMT='(/)')
  WRITE(6,FMT='(3X,12G10.2)') (ELSTIF(IROW,JCOL),JCOL=33,44)
  WRITE(6,FMT='(3X,12G10.2)') (ELSTIF(IROW,JCOL),JCOL=45,56)
  WRITE(6,FMT='(3X,12G10.2)') (ELSTIF(IROW,JCOL),JCOL=57,LVABZ)
  WRITE(6,FMT='(/)')
1  CONTINUE
DO 3 IROW = 1,68
  WRITE(6,FMT='(3G11.3)') ERHSU(IROW),ERHSV(IROW),ERHSW(IROW)
3  CONTINUE
C   DO 5 IROW = 1,32
C     SUM(IROW) = 0.0
C     SUMU(IROW) = 0.0
C     SUMV(IROW) = 0.0
C     SUMW(IROW) = 0.0
C     SUMP(IROW) = 0.0
C5  CONTINUE
C   DO 10 IROW=1,32
C     DO 20 JCOL=1,32
C       SUM(IROW) = SUM(IROW) + ELSTIF(IROW,JCOL)
C20  CONTINUE
C     IF(ABS(SUM(IROW)).LT.1E-06) SUM(IROW)=0.0
C10  CONTINUE
C     DO 30 IROW=1,32
C       DO 40 JCOL=1,8
C         PTR=4+JCOL-3
C         SUMU(IROW) = SUMU(IROW) + ELSTIF(IROW,PTR)
C         SUMV(IROW) = SUMV(IROW) + ELSTIF(IROW,PTR+1)
C         SUMW(IROW) = SUMW(IROW) + ELSTIF(IROW,PTR+2)
C         SUMP(IROW) = SUMP(IROW) + ELSTIF(IROW,PTR+3)
C40  CONTINUE
C       IF(ABS(SUMU(IROW)).LT.1E-06) SUMU(IROW)=0.0
C       IF(ABS(SUMV(IROW)).LT.1E-06) SUMV(IROW)=0.0
C       IF(ABS(SUMW(IROW)).LT.1E-06) SUMW(IROW)=0.0
C       IF(ABS(SUMP(IROW)).LT.1E-06) SUMP(IROW)=0.0
C30  CONTINUE
C     WRITE(6,500)
C     DO 100 IROW=1,32
C       WRITE(6,501) IROW,SUM(IROW),SUMU(IROW),SUMV(IROW)
C       1      ,SUMW(IROW),SUMP(IROW)
C100 CONTINUE
C500 FORMAT(X,'THE SUM OF THE ROWS OF THE ELEMENT STIFFNESS MATRIX
C       1      ARE GIVEN BELOW',/)

```

C501   FORMAT(X, 'THE SUMS (TOTAL,U,V,W,P) FOR ROW ',I2, ' IS ',5G15.5)
      RETURN
      END

Appendix E

SEPERATE

```

*****
*
* Name - SEPERATE.FTN
*
* Program to sepearate the columns of u, v,w, and p in order
* to evaluate the iterative data by use of a plotting routine.
* The output is directly to the pad so that it may cut and pasted
* as desired.
*
* Input - ASGARD output file trimmed of all entries except
*         the iterated primitive values
*
* Output- Several files containing either all nodes with
*         iterated data for up to five iterations or u,v,w,
*         and p for one node for up to five iterations
*
* Subroutines - READER,REMOVE, SORT,INIT,MODIFY
* Functions   - FIND
*
* Author - Marc Dulude
* Date   - 20 Feb 1985
*
*****

```

```

*
PROGRAM SEPARATE
C
IMPLICIT LOGICAL (A-Z)
C
INTEGER ITERMAX,NMAX,ELMAX
PARAMETER (ITERMAX=20,NMAX=141,ELMAX=20)
C
INTEGER
1 I,J,K,L,M,NFIX,NODMAX,NELZ,TYPE,FLAG,NUMBER,NODLIST(NMAX)
2, NITER,TRY,CORNER(8,ELMAX),CORNOD(8*ELMAX),MODFLAG
3,LNODS(20,ELMAX)
C
REAL
1 VALUES(5,NMAX,ITERMAX),FIX(6,NMAX),TEMP(5,NMAX,ITERMAX)
C
CHARACTER*1 YUPPER,YLOWER,REPLY,AGAIN
DATA YUPPER,YLOWER /'Y','y'/

```

```

*****
*
*   DICTIONARY OF VARIABLES -
*
*   AGAIN   - character string used to read response to query of
*             whether or not the user has finished with the data
*   CORNER   - 2 dimensional array of node numbers corresponding to
*             the corner node numbers
*   CORNOD   - 1 dimensional sorted array corresponding to the array
*             CORNER
*   CREATE   - character string used to find specific location in the
*             input file
*   ELMAX    - maximum number of elements the program is now capable
*             of handling. Required for array dimensioning
*   FID      - character string used to store the name of the input
*             file
*   FIX      - real array used to store the node number, fixity code,
*             u,v,w, and p values of the nodes with prescribed
*             values
*   FIXNO    - the sequence number in the list of NFIX variables
*   FLAG     - flag used to identify various cases in the program
*   I,J,K,L,M - integer indices used for DO loop counters and as
*             limits to nested DO's
*   IO       - IOSTAT number returned from the file OPEN in case of
*             error. Translated to give error message
*   ITERMAX  - maximum number of iterations the program is now capable
*             of interpreting. Required for array dimensioning
*   MARK     - position of a node in the reduced VALUES array as a
*             result of removing nodes through REMOVE
*   MEMBER   - logical variable returned to READER from SORT to
*             whether the node given is part of the set of corner
*             nodes or not
*   NELZ     - number of elements in the problem
*   NFIX     - number of nodes with prescribed values
*   NITER    - maximum number of iterations in the problem
*   NMAX     - maximum number of nodes the program is now capable of
*             interpreting. Required for array dimensioning
*   NODE     - the node currently being investigated in the subroutine
*             SORT
*   NODMAX   - maximum number of iterations in this problem
*   NUM      - number of corner nodes resulting from NELZ elements
*   NUMBER   - integer value calculated to govern the number of times
*             to loop in particular WRITE loops. Governed by the
*             fact that the plotting routine currently used can
*             handle 5 separate x and y variables in its input file
*   REPLY    - character string used to read response to query of
*             whether or not to remove prescribed value nodes from

```

```

*           the output
*  RHSNO    - integer right hand side number corresponding to nodes
*             with prescribed values
*  SORTFLAG- flag to whether a sort is required of the subroutine
*             SORT or whether it should return a TRUE logical
*             variable, MEMBER, indicating the node is a corner node
*  STRING    - dummy character string used to read through useless
*             data in the input file
*  TEMP      - temporary value used in sorting the values of the array
*             CORNER to the array CORNOD
*  TIME      - character string used to find specific location in the
*             input file
*  TRY       - number of cycles the loop must go through in order to
*             output all iterations for a node
*  TYPE      - character string used to read response to identify what
*             kind of output is desired
*  VALUES   - 3 dimensional array of the iterated values read from
*             the output file. The first column gives the node
*             number, the second to fifth give the pressure and
*             u,v,w values respectively.
*  YUPPER    - upper case Y character string
*  YLOWER    - lower case y character string

```

```

*****

```

```

*
*  FLAG=0
*  MODFLAG=1
*  CALL READER(VALUES,NFIX,NODMAX,NITER,NELZ,FLAG,FIX,ITERMAX,NMAX
1,          ELMAX,CORNER,CORNOD,LNODS)
C
5  CONTINUE
   PRINT*,'What kind of output do you want?'
   PRINT*,'Enter - (1) for u,v,w,p charts'
   PRINT*,'Enter - (2) for node charts.'
   READ (5,*) TYPE
C
   PRINT*,'Do you want the values output as read or do you want',
1     ' the midside nodes'
   PRINT*,'changed from hierarchical deviations to absolute',
1     ' quantities?'
   PRINT*,'Enter - (1) for unchanged values'
   PRINT*,'          - (2) for absolute midside values.'
   PRINT*,'Default is no modification unless you have already',
1     ' modified'
   PRINT*,'it from where there is no turning back.'
   READ(5,*) MODFLAG
   IF(MODFLAG.EQ.2) CALL MODIFY(ELMAX,NELZ,LNODS,NMAX,ITERMAX
1,          VALUES,NITER,NODMAX)

```

C

```

PRINT*, 'Do you wish to remove some nodes from the output?'
PRINT*, 'Enter Y or y for YES. Default is NO.'
READ (5, FMT='(A1)') REPLY
IF (REPLY.EQ.YUPPER.OR.REPLY.EQ.YLOWER) THEN
  CALL REMOVE(FIX, NFIX, VALUES, NODMAX, NITER, CORNER, ELMAX, CORNOD
1,    NELZ, NMAX, ITERMAX, NODLIST, TEMP)
ENDIF

```

C

```

IF (TYPE.EQ.2) GO TO 200

```

*

*****

```

* Case of one file for each of u,v,w, and p with all nodes
* in each file for up to 5 separate iterations.
* Iteration number 1 is not displayed due to the highly
* transient nature of the test.

```

*****

*

```

NUMBER = (NITER-1)/5 + 1
DO 10 I=1, NUMBER
  J= (I-1)*5 + 1
  K= J + 4
  IF (K.GT.NITER) K=NITER
  WRITE(6,*) 'U FOR ITERATION ', J, ' TO ITERATION ', K
  DO 20 L=1, NODMAX
    WRITE(6,*) VALUES(1,L,J), (VALUES(2,L,M), M=J,K)
20  CONTINUE

```

C

```

  WRITE(6,*) 'V FOR ITERATION ', J, ' TO ITERATION ', K
  DO 30 L=1, NODMAX
    WRITE(6,*) VALUES(1,L,J), (VALUES(3,L,M), M=J,K),
30  CONTINUE

```

C

```

  WRITE(6,*) 'W FOR ITERATION ', J, ' TO ITERATION ', K
  DO 40 L=1, NODMAX
    WRITE(6,*) VALUES(1,L,J), (VALUES(4,L,M), M=J,K)
40  CONTINUE

```

C

```

  WRITE(6,*) 'P FOR ITERATION ', J, ' TO ITERATION ', K
  DO 50 L=1, NODMAX
    WRITE(6,*) VALUES(1,L,J), (VALUES(5,L,M), M=J,K)
50  CONTINUE
10  CONTINUE
GO TO 500

```

*

*****

```

* Case of one node with u,v,w,p for each
*****

```

```

*
200  CONTINUE
    NUMBER = ((NITER-1)/5 + 1)
    DO 100 I=1,NODMAX
        DO 110 TRY=1,NUMBER
            J= (TRY-1)*5 + 1
            K= J + 4
            IF (K.GT.NITER) K=NITER
            WRITE(6,FMT=' (/)')
            WRITE(6,*) 'U,V,W, AND P FOR NODE ',VALUES(1,I,2),
1              ' FOR ITERATIONS ',J,' TO ',K
            DO 120 L=J,K
                WRITE (6,*) L,(VALUES(M,I,L),M=2,5)
120      CONTINUE
110      CONTINUE
100      CONTINUE
C
500  CONTINUE
*
*****
*   Allow the user to use the same read values to look at various
*   cases by asking him if he wants to quit or keep going.
*****
*
PRINT*,'Do you wish to stop or use this same data for'
PRINT*,'further analysis? Answer y or n.'
PRINT*,'Default is NO.'
READ(5,FMT='(A1)') AGAIN
IF (AGAIN.EQ.YUPPER.OR.AGAIN.EQ.YLOWER) GO TO 5
STOP
END
*
*****
*
*   Subroutine to read the necessary data from the input file
*   (ie. the ASGARD output file).
*
*   Called from SEPARATE
*
*   Calling to INIT and SORT
*
*****
*
SUBROUTINE READER (VALUES,NFIX,NODMAX,NITER,NELZ,FLAG,FIX
1, ITERMAY,NMAX,ELMAX,CORNER,CORNOD,LNODS)
IMPLICIT LOGICAL (A-Z)

```

C
C

```

      INTEGER
1     I,J,K,NODMAX,NITER,NFIX,NELZ,FLAG,RHSNO,FINO,ITERMAX
2     ,NMAX,IO, SORTFLAG,ELMAX,CORNER(8,ELMAX),CORNOD(8*ELMAX)
3     ,LNODS(20,ELMAX)

C
      REAL
1     VALUES(5,NMAX,ITERMAX),FIX(6,NMAX)

C
      LOGICAL MEMBER

C
      CHARACTER*80 STRING
      CHARACTER*6 TIME
      CHARACTER*20 CREATE
      CHARACTER*20 FID
      DATA TIME /' Time: '/
      DATA CREATE /' CREATE ELEMENT FILE'/
      DATA FID /' '/

C
      PRINT*, 'What is the name of the input file?'
      READ (5,FMT='(A20)') FID
      OPEN (UNIT=10,FILE=FID,STATUS='OLD',ERR=1000,IOSTAT=IO)

*
*****
*   Find convenient datum from which to read the rest of the data
*   from the file (ie. the line "Time:")
*****
*
      CALL INIT (VALUES,ITERMAX,NMAX,CORNER,ELMAX,CORNOD)
      DO 10 I=1,20
          READ (10,FMT='(A6)') STRING
          IF (STRING.EQ.TIME) GO TO 15
10      CONTINUE
*
*****
*   Something wrong. TITLEN may give longer intro than 20 lines.
*   If so, change the length of this DO loop accordingly.
*****
*
      PRINT*, 'Cannot find the "Time:" line'
      PRINT*, 'Stopping in READER'
      STOP

C
15      CONTINUE
*
*****
*   Read NELZ,NFIX,NITER, & NODMAX from required lines of the file
*****
*

```

```

DO 20 I=1,17
  READ (10,FMT='(A10)') STRING
20  CONTINUE
  READ (10,500) STRING,NELZ
  READ (10,510) STRING,NFIX
  PRINT*, 'NELZ = ',NELZ
  PRINT*, 'NFIK = ',NFIK
  DO 30 I=1,10
    READ (10,FMT='(A1)') STRING
30  CONTINUE
  READ (10,520) STRING,NITER
  DO 40 I=1,9+NELZ
    READ (10,FMT='(A1)') STRING
40  CONTINUE
  READ (10,530) STRING,NODMAX
  PRINT*, 'Maximum number of iterations = ',NITER
  PRINT*, 'NODMAX = ',NODMAX
*
*****
*   Read the nodes with prescribed values
*****
*
  DO 50 I=1,14+NODMAX
    READ (10,FMT='(A1)') STRING
50  CONTINUE
    DO 60 I=1,NFIK
      READ (10,*) RHSNO,FIKNO,(FIK(J,I),J=1,6)
60  CONTINUE
  DO 70 I=1,20
    READ (10,FMT='(A20)') STRING
    IF (STRING.EQ.CREATE) GO TO 75
70  CONTINUE
    PRINT*, 'Cannot find CREATE ELEMENT FILE line.'
    PRINT*, 'Stopping in READER.'
    STOP
75  CONTINUE
*
*****
*   Read in the element topology so that the values associated with
*   corner nodes can be made to read 4 numbers yet only 3
*   numbers for the midside nodes. The topology will also be used
*   to modify the hierarchical midside values to absolute quantities
*   if the user wishes.
*****
*
  READ (10,FMT='(A1)') STRING
  DO 80 I=1,NELZ-1
    READ (10,540) STRING,(LNODS(J,I),J=1,8)

```

```

      READ (10,FMT='(12I7)') (LNODS(J,I),J=9,20)
      DO 85 K=1,2
        READ (10,FMT='(A1)') STRING
85      CONTINUE
80      CONTINUE
*
*****
*   Read in the topology of the last element
*****
*
      READ (10,540) STRING,(LNODS(J,I),J=1,8)
      READ (10,FMT='(12I7)') (LNODS(J,I),J=9,20)
      READ (10,FMT='(A1)') STRING
*
*****
*   Gleen the corner node numbers of the elements from the
*   topology list, LNODS, before going to SORT.
*****
*
      DO 88 I=1,NELZ
        DO 90 J=1,8
          CORNER(J,I) = LNODS(J,I)
90      CONTINUE
88      CONTINUE
      SORTFLAG=1
      CALL SORT (SORTFLAG,CORNER,I,MEMBER,ELMAX,CORNQD,NELZ)
*
*****
*   Read the values of u,v,w,and p for all iterations from the file
*****
C
*****
*   Read the first iteration values
*****
*
      J=1
      DO 95 I=1,NODMAX
        SORTFLAG = 2
        MEMBER = .FALSE.
        CALL SORT (SORTFLAG,CORNER,I,MEMBER,ELMAX,CORNQD,NELZ)
        IF (MEMBER) THEN
          READ (10,FMT=*) (VALUES (K,I,J),K=1,5)
        ELSE
          READ (10,FMT=*) (VALUES (K,I,J),K=1,4)
          VALUES(5,I,J)=2.22E+22
        ENDIF
95      CONTINUE
*

```



```

*****
*   Now, for the second through to the last iteration
*****
*
      DO 100 J=2,NITER
C          DO 995 I=1,NODMAX
C              WRITE (6,*) (VALUES(K,I,J-1),K=1,5)
C995          CONTINUE
      DO 110 I=1,NODMAX
          SORTFLAG = 2
          MEMBER = .FALSE.
          CALL SORT (SORTFLAG,CORNER,I, MEMBER,ELMAX,CORNOD,NELZ)
*
*****
*   If a reading error occurs, it's because we've tried to read the
*   "CONVERGED SOLUTION" line of the file so go & read the
*   converged solution that follows.
*****
*
          IF (MEMBER) THEN
              READ (10,*,ERR=200) (VALUES (K,I,J),K=1,5)
          ELSE
              READ (10,*,ERR=200) (VALUES (K,I,J),K=1,4)
              VALUES(5,I,J)=2.22E+22
          ENDIF
110      CONTINUE
          DO 120 I=1,2
              READ (10,FMT='(A1)') STRING
120      CONTINUE
100      CONTINUE
          GO TO 300
*
*****
*   Case of the converged solution
*****
*
200      CONTINUE
          NITER=J
          DO 210 I=1,3
              READ (10,FMT='(A1)') STRING
210      CONTINUE
          DO 220 I=1,NODMAX
              SORTFLAG=2
              MEMBER = .FALSE.
              CALL SORT (SORTFLAG,CORNER,I, MEMBER,ELMAX,CORNOD,NELZ)
              IF (MEMBER) THEN
                  READ (10,*) (VALUES (K,I,J),K=1,5)
              ELSE

```

```

      READ (10,*) (VALUES (K,I,J),K=1,4)
      VALUES(5,I,J)=2.22E+22
      ENDIF
220  CONTINUE
300  CONTINUE
*
*****
*   FORMAT statements
*****
*
500  FORMAT (A20,I4)
510  FORMAT (A48,I4)
520  FORMAT (A53,I4)
530  FORMAT (A31,I4)
540  FORMAT (A68,8I7)
      RETURN
1000 PRINT*, 'Error in opening input file.'
      PRINT*, 'IOSTAT is ',IO
      STOP
      END
*
*****
*
*   Subroutine serving a dual function. Originally, with a SORTFLAG
*   of 1, it changes the 2 dimensional array, CORNER, into a 1 D
*   array, CORNOD, that will be used to establish whether or not
*   specified nodes are corner nodes or not. This is the second
*   function. With a SORTFLAG of 2, the routine returns a logical
*   variable, MEMBER, which is true if NODE was a corner node and
*   remains false if it is not.
*
*   Called from READER and REMOVE
*
*****
*
      SUBROUTINE SORT (SORTFLAG,CORNER,NODE,MEMBER,ELMAX,CORNOD
1, NELZ)
C
      IMPLICIT LOGICAL (A-Z)
C
      INTEGER
1  SORTFLAG,ELMAX,I,J,K,CORNER(8,ELMAX),CORNOD(8*ELMAX)
2, TEMP,NUM,NODE,NELZ
C
      LOGICAL MEMBER
C
      NUM = 8*NELZ
      IF (SORTFLAG.EQ.2) GO TO 100

```

```

*
*****
*   Go through the array CORNER to convert it to the one dimensional
*   array, CORNOD.
*****

```

```

*
      I=0
      DO 10 J=1,NELZ
        DO 20 K=1,8
          I=I+1
          CORNOD(I)=IABS(CORNER(K,J))
20      CONTINUE
10      CONTINUE
      I=1
30      CONTINUE
      IF (CORNOD(I+1).LT.CORNOD(I)) THEN
        TEMP = CORNOD(I+1)
        CORNOD(I+1) = CORNOD (I)
        CORNOD(I) = TEMP
      ENDIF
      I=I+1
      IF (I.LT.NUM) GO TO 30
      GO TO 200

```

```

*
*****
*   This portion of the subroutine steps through the entries in the
*   one dimensional array, CORNOD, comparing it to the given node
*   number, NODE, finally jumping out of the loop if one of the
*   entries does correspond. If so, MEMBER is set to TRUE.
*****

```

```

*
100     CONTINUE
      DO 40 I=1,NUM
        IF (NODE.EQ.CORNOD(I)) GO TO 45
40      CONTINUE
        MEMBER = .FALSE.
        GO TO 200
45      CONTINUE
        MEMBER = .TRUE.
200     CONTINUE
      RETURN
      END

```

```

*
*****
*   Subroutine to remove the fixed value nodes from the data to be
*   output
*

```

```

*   Called from SEPARATE
*
*   Calling to SORT and FIND
*
*****
*
      SUBROUTINE REMOVE(FIX,NFIX,VALUES,NODMAX,NITER,CORNER,ELMAX
1,      CORNOD,NELZ,NMAX,ITERMAX,NODLIST,TEMP)
C
      IMPLICIT LOGICAL (A-Z)
C
      INTEGER
1  NFIX,NODMAX,NITER,ELMAX,NELZ,NODE,I,J,K,ITER, SORTFLAG, MARK
2,  NMAX,ITERMAX,FIND,CORNER(8,NELZ),CORNOD(8*NELZ),FLAG
3,  NUMBER,NODLIST(NODMAX)
C
      REAL
1  FIX(6,NFIX),VALUES(5,NMAX,ITERMAX),TEMP(5,NODMAX,NITER)
C
      LOGICAL MEMBER
C
      DO 989 J=1,NITER
C        WRITE(6,*) J
C        DO 990 I=1,NODMAX
C          WRITE (6,*) (VALUES(K,I,J),K=1,5)
C990    CONTINUE
C989    CONTINUE
      PRINT*,'Do you wish (1) prescribed value nodes removed; or'
      PRINT*,'          (2) list of user input nodes only kept?'
      PRINT*,' '
      PRINT*,'Enter 1 or 2'
      READ (6,*) FLAG
      IF (FLAG.EQ.2) GO TO 100
*
*****
*   The only nodes that are removed from the data to be output are
*   corner nodes with a fixity code of 1111 or midside nodes with
*   fixity codes of 111. The reason for this selectivity is that
*   any other nodes will allow at least one of the variables to
*   vary and should, therefore, be included.
*****
*
      SORTFLAG = 2
      DO 10 I=1,NFIX
        NODE = FIX(1,I)
        MEMBER = .FALSE.
        CALL SORT(SORTFLAG,CORNER,NODE,MEMBER,ELMAX,CORNOD,NELZ)
*

```

*****

- * If the node number to remove is the last one in the array.
- * VALUES, merely decrement the length of the array (ie. NODMAX)
- * thereby giving the illusion that the node has been removed.

*****

*

```

      IF (NODE.NE.VALUES(1,NODMAX,1)) THEN
      IF ((MEMBER.AND.FIX(2,I).EQ.1111).OR.(.NOT.MEMBER.AND.
1      FIX(2,I).EQ.111)) THEN
      NODMAX=NODMAX-1
      MARK=FIND(NODE,VALUES,NMAX,ITERMAX,NODMAX)

```

*

*****

- * Step through each of the nodes above the node to be removed for all
- * iterations and set the index corresponding to each 1 less than
- * before (also decrement the length of the array, NODMAX, accordingly).
- * Thus, the values above the one in question "squeeze" it out of the
- * array.

*****

*

```

      DO 20 ITER=1,NITER
      DO 30 J=MARK,NODMAX
      DO 40 K=1,5
      VALUES(K,J,ITER)=VALUES(K,J+1,ITER)
40      CONTINUE
30      CONTINUE
20      CONTINUE
      ENDIF
      ELSE
      NODMAX=NODMAX-1
      ENDIF
10  CONTINUE
      GO TO 200

```

*

*****

- * Case of keeping only a user input list of nodes

*****

*

```

100  CONTINUE
      PRINT*,'How many nodes do you wish to keep?'
      READ(5,*) NUMBER
      IF (NUMBER.GT.NODMAX) THEN
      PRINT*,'That is more than the number of nodes in the problem'
      PRINT*,'Try again'
      GO TO 100
      ENDIF
      PRINT*,'Enter the node numbers to keep, one per line'

```

*

```

*****
*   Read in the list of nodes the user wants to keep
*****
*
      DO 110 I=1,NUMBER
        READ(5,*) NODLIST(I)
110    CONTINUE
*
*****
*   Isolate the nodes to keep in VALUES and store them in the
*   temporary array, TEMP
*****
*
      DO 120 I=1,NUMBER
        NODE = NODLIST(I)
        DO 130 J=1,NITER
          DO 140 K=1,5
            TEMP(K,I,J) = VALUES(K,NODE,J)
140        CONTINUE
130      CONTINUE
120    CONTINUE
*
*****
*   Give the isolated nodal values to keep back to the array, VALUES
*****
*
      NODMAX = NUMBER
      DO 150 I=1,NITER
        DO 160 J=1,NODMAX
          DO 170 K=1,5
            VALUES(K,J,I) = TEMP(K,J,I)
170        CONTINUE
160      CONTINUE
150    CONTINUE
200    CONTINUE
      RETURN
      END
C
*
*****
*   Function used to find the current position of the input node,
*   NODE, in the array VALUES. Required to eliminate NODE from the
*   array since it is a prescribed value node and thus invariant.
*
*   Called from REMOVE
*
*****

```

```

*
  INTEGER FUNCTION FIND(NODE,VALUES,NMAX,ITERMAX,NODMAX)
C
  IMPLICIT LOGICAL (A-Z)
C
  INTEGER NODE,NMAX,ITERMAX,NODMAX,I,TEMP
C
  REAL VALUES(5,NMAX,ITERMAX)
C
  DO 10 I=1,NODMAX
    TEMP = VALUES(1,I,1)
    IF (NODE.EQ.TEMP) THEN
      FIND=I
      GO TO 20
    ENDIF
10  CONTINUE
    PRINT*,'Cannot find node to remove in the array VALUES'
    PRINT*,'Node unable to find is number ',NODE
    PRINT*,'Stopping in FUNCTION FIND'
    STOP
20  CONTINUE
    RETURN
  END

```

```

*
*****
*
* Subroutine to initialize all arrays with zero entries to ease
* the debugging task
*
* Called from READER
*
*****

```

```

*
  SUBROUTINE INIT (VALUES,ITERMAX,NMAX,CORNER,ELMAX,CORNOD)
  INTEGER
1  ITERMAX,NMAX,ELMAX,CORNER(8,ELMAX),CORNOD(8*ELMAX),I,J,K
C
  REAL VALUES(5,NMAX,ITERMAX)
C
  DO 10 I=1,ELMAX
    DO 20 J=1,8
      CORNER(I,J)=0.0
20  CONTINUE
10  CONTINUE
C
  DO 30 I=1,8*ELMAX
    CORNOD(I)=0.0
30  CONTINUE

```

```

C
      DO 40 I=1,ITERMAX
        DO 50 J=1,NMAX
          DO 60 K=1,5
            VALUES(K,J,I)=0.0
60      CONTINUE
50      CONTINUE
40      CONTINUE
C
      RETURN
      END

*
*****
*
*   Subroutine to change the hierarchical deviations used as the dof
*   for the midside nodes of an element to an absolute quantity.
*   This is done by bearing in mind that the dof is the deviation
*   away from the linearly interpolated values from the respective
*   corner nodes.
*
*****
*
      SUBROUTINE MODIFY(ELMAX,NELZ,LNODS,NMAX,ITERMAX,VALUES,NITER
1,      NODMAX)
C
      INTEGER
      1  I,J,K,L,ELMAX,NMAX,ITERMAX,NELZ,NODMAX,NITER,PTR1,PTR2,NODE
      2, LNODS(20,ELMAX),MIDS20(3,12)
C
      REAL *8 VALUES(6,NMAX,ITERMAX)
C
      DATA MIDS20/ 9,1,2, 10,2,3, 11,3,4, 12,4,1, 13,1,5, 14,2,6,
1      15,3,7, 16,4,8, 17,5,6, 18,6,7, 19,7,8, 20,8,5/
*
*****
*   Find the two surrounding corner nodes, PTR1 & PTR2, for the
*   midside node in question, NODE using the local numbering scheme
*   and the global topology stored in LNODS.
*****
*
      DO 10 I = 1,NELZ
        DO 20 J = 1,12
          NODE = LNODS(MIDS20(1,J),I)
*
*****
*   If the node number is positive, it is not the last occurrence
*   so no need to modify it yet.
*****

```



```

*
  IF (NODE.GT.0) GOTO 20
  NODE = IABS(NODE)
  PTR1 = IABS(LNODS(MIDS20(2,J),I))
  PTR2 = IABS(LNODS(MIDS20(3,J),I))
  DO 30 K = 1,NITER-1
    DO 40 L = 2,5

```

```

*****

```

```

*   The midside node value becomes the sum of its original value and
*   the value of the linear interpolation value found using the two
*   surrounding corner nodes.

```

```

*****

```

```

*
      VALUES(L,NODE,K)=VALUES(L,NODE,K) + ((VALUES(L,PTR1,K)
1      + VALUES(L,PTR2,K))*0.5)
40      CONTINUE
30      CONTINUE
20      CONTINUE
10      CONTINUE
      RETURN
      END

```

END

2|3·1 2·|87

FIN