

2004

Modeling incompressible solid materials using finite volume methods

Bryce L. Fowler
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Fowler, Bryce L., "Modeling incompressible solid materials using finite volume methods" (2004). *Master's Theses*. 2606.
DOI: <https://doi.org/10.31979/etd.maj4-auup>
https://scholarworks.sjsu.edu/etd_theses/2606

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

MODELING INCOMPRESSIBLE SOLID MATERIALS
USING FINITE VOLUME METHODS

A Thesis

Presented to

The Faculty of the College of Engineering

Department of Mechanical and Aerospace Engineering

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Bryce L. Fowler

August 2004

UMI Number: 1424477

INFORMATION TO USERS

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

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

UMI[®]

UMI Microform 1424477

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© 2004

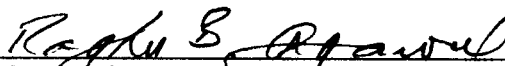
Bryce L. Fowler

ALL RIGHTS RESERVED

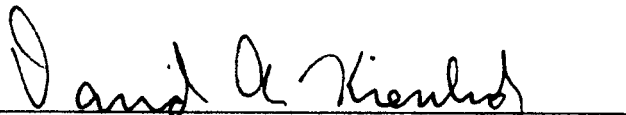
APPROVED FOR THE DEPARTMENT OF
MECHANICAL AND AEROSPACE ENGINEERING



Dr. Raymond K. Yee, Committee Chair, SJSU

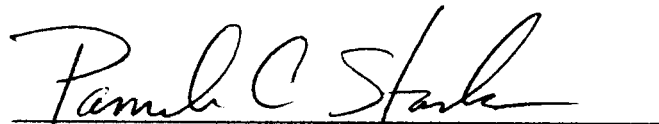


Dr. Raghu B. Agarwal, SJSU



Dr. David A. Kienholz, CSA Engineering, Inc.

APPROVED FOR THE UNIVERSITY



ABSTRACT

MODELING INCOMPRESSIBLE SOLID MATERIALS USING FINITE VOLUME METHODS

by Bryce L. Fowler

Modeling solid structures with computational fluid dynamics (CFD) methods is still in the research stage. This thesis provides an alternative to the specialized finite element formulations currently used to model incompressible materials. It also adds to the understanding of using CFD methods to model solid structures. The product of this thesis, a 2-D finite-volume code written in Matlab, may be used as a basis for a full 3-D analysis code for modeling incompressible materials using CFD methods. Two simple test cases (plane strain and axisymmetric) are presented and favorably compared to finite-element results.

ACKNOWLEDGEMENTS

The author would like to thank Prof. Raymond K. Yee, Prof. Raghu B. Agarwal, and Dr. David A. Kienholz for providing advice and insight for this work.

The author would like to acknowledge Dr. Pieter Wesseling for providing his finite volume code under the GNU General Public License (GPL). Since the work of this thesis uses much of this GPL code, the code created for this thesis is also available under the GPL.

TABLE OF CONTENTS

INTRODUCTION	1
BACKGROUND	3
COMPUTATIONAL FLUID DYNAMICS	6
DISCRETIZATION.....	8
DIFFERENCING SCHEMES	10
PRESSURE-MOMENTUM COUPLING	14
SIMPLE ALGORITHM	17
MODIFICATION FOR SOLID MECHANICS.....	19
TEST CASES	23
DISCUSSION.....	35
CONCLUSION.....	36
FUTURE WORK.....	37
BIBLIOGRAPHY	38
APPENDIX	40

LIST OF FIGURES

Figure 1: Sample finite element mesh.....	5
Figure 2: Staggered grid.....	15
Figure 3: Staggered grid indexing.....	16
Figure 4: 2-D test case	23
Figure 5: FV horizontal displacements	24
Figure 6: FV vertical displacements	25
Figure 7: Vertical displacements from FE	26
Figure 8: Finite volume stress distribution.....	28
Figure 9: Finite element stress distribution	29
Figure 10: Axisymmetric model	30
Figure 11: FV radial displacements	31
Figure 12: FV Von Mises stress.....	32
Figure 13: FE radial displacements.....	33
Figure 14: FE Von Mises stress	33

LIST OF TABLES

Table 1: First order differencing schemes	12
Table 2: Comparison of FV and FE results	27
Table 3 : Comparison of axisymmetric FV and FE results	34

INTRODUCTION

Polymers constitute a large class of nearly incompressible solid materials (i.e., Poisson's Ratio ≈ 0.5). These materials are often used as passive vibration isolators. Accurately modeling vibration isolators made of nearly incompressible materials has been extremely difficult with standard finite element analysis.

Finite element (FE) codes typically cannot model incompressible materials correctly because the fully integrated models experience node locking.¹ Modifications have been made to the FE method to accommodate incompressible materials. One is to include displacement and hydrostatic pressure as unknowns. This mixed method, however, may still experience locking. Another method is to alter the numerical integration procedure used to evaluate the element stiffness matrix and internal force vector. This is the method used in Pronto3D,² a U. S. Government-owned code developed at Sandia Labs with strong export restrictions. Commercial codes, such as MSC/Marc³ also use this method.

Another problem for modeling software is that of calculating the deformations and loads at a liquid/solid interfacial boundary. Polymer flow as found in injection, extrusion, and other polymer processing machinery is an example of this. Finite element codes are used to model solids, computational fluid dynamics codes are used to model fluids. Computational fluid dynamics (CFD) methods are classed as finite difference (FD) or finite volume (FV). Normally, a CFD code is used to model fluid flow and calculate the pressure at the liquid/solid boundary. FE code is then used to calculate the deformations and stresses in the solid. Using two different analysis techniques makes the process of design very difficult. Changes in the design mean changes must be made to

the meshes for both the CFD and FE codes. Iterative designs are typically not done due to the cumbersome nature of making design changes.

Finite volume codes have recently been modified by Fallah, Bailey, and Cross,⁴ Wheel,⁵ and others⁶ to model deformations in loaded structures. To model incompressible materials, a mixed FV procedure that includes displacement and pressure variables is used. This is similar to the “mixed” FE method.

Finite volume involves integrating the terms of the Navier-Stokes equations over each control volume. This distinguishes FV from other CFD techniques. The resulting integrands express the exact conservation of the relevant properties (e.g., velocity) for each finite size cell. The conservation of the general flow variable within a finite control volume can be expressed as a balance between the processes increasing or decreasing it. The fact that the FV method guarantees local conservation of the fluid property for each control volume makes it desirable for performing the analysis of incompressible materials.

This thesis is investigating the finite volume method for modeling incompressible materials because FV will not experience the locking that FE methods do.

BACKGROUND

The Lamé parameters for an isotropic elastic material are given as

$$L = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

$$G = \frac{E}{2(1+\nu)}$$

where

E is Young's modulus

G is the shear modulus

ν is Poisson's ratio

The bulk modulus, K , which is the ratio of volume stress to volume strain, and also relates hydrostatic pressure to volume change, is given by

$$K = 2G + 3L$$

Substitution of the Lamé parameters gives

$$\frac{K}{G} = \frac{2(1+\nu)}{3(1-2\nu)}$$

As $\nu \rightarrow 0.5$, the bulk modulus becomes infinitely large. In this case, the volume change is constrained to be zero and the material is considered incompressible.

Consider the finite elements in Figure 1. These elements are linear triangles. The displacement field is linearly interpolated throughout the element. Therefore, the strain is constant throughout the element. This means that pointwise,

$$\nabla \cdot \mathbf{u}_v^N = 0$$

where

$\nabla \cdot$ is the “div” operator, example: $\nabla \cdot \mathbf{u} = \frac{\partial u_{(x)}}{\partial x} + \frac{\partial u_{(y)}}{\partial y} + \frac{\partial u_{(z)}}{\partial z}$

\mathbf{u}_v^N is the displacement solution in the finite element space, \mathbf{V}^N

This means that the total volume of the elements cannot change. It also means that the area of each triangular element may not change, as given by

$$\frac{dA}{A} = \int_A \nabla \cdot \mathbf{u}_v^N dA = 0$$

One can see in Figure 1 that the constant area constraint means that Node A can only move in the horizontal direction, since the two lower nodes of Element 1 are fixed. Similarly, Element 2 places the restriction that Node A can only move vertically. This means that Node A is restricted to zero displacement. This argument may be repeated throughout the mesh, restricting all nodes to zero displacement, in essence locking all nodes, and giving $\mathbf{u}_v^N = 0$ as the solution to the discrete problem. Of course, incompressibility does not preclude deformation, so the solution is not physical.

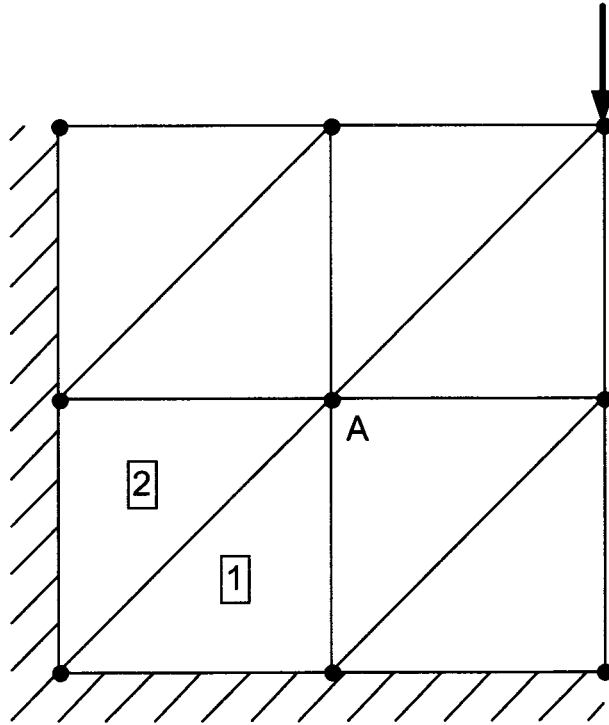


Figure 1: Sample finite element mesh

One method to overcome element locking is to relate displacement and hydrostatic pressure as unknowns. Another is to deliberately under-integrate the internal force vector $\mathbf{f}^{\text{int}^e}$ and the element stiffness matrix \mathbf{k}^e in the finite-element formulation. Another is to use high-order h, p and mixed hp elements.⁷ Another is to use the computational fluid dynamics approach of finite volumes.

COMPUTATIONAL FLUID DYNAMICS

In fluid mechanics, the equations of conservation are given by

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho\mathbf{v}\phi) = \nabla \cdot (\Gamma_\phi \nabla \phi) + S_\phi$$

where

$\phi = 1$ is the continuity equation

$\phi = h$ is the Enthalpy equation

$\phi = v_{(i)}$ is the momentum equation in the i direction

ρ is the density

\mathbf{v} is the velocity vector

Γ_ϕ is the fluid diffusion coefficient

S_ϕ is the source term

∇ is the “grad” operator, example: $\nabla \phi = \frac{\partial \phi}{\partial x} i + \frac{\partial \phi}{\partial y} j + \frac{\partial \phi}{\partial z} k$

The techniques used to discretize the various conservation equations are based on the cell centered finite volume formulation. In this method the domain over which the equation is to be solved is divided into a set of non-overlapping polyhedral control volumes, or elements. At the center of each of these control volumes a single node is positioned. This set of nodes form the selected points at which the value of ϕ is sought. The FV method involves integrating the conservation equation, above, over each control volume, as well as over time, and a solution is sought which makes each of these integrals equal to zero. Assumptions are made as to the variation between nodes of the

quantities involved in the equation. For each control volume this leads to a linear equation involving the unknown values of the scalar quantity at the node in the control volume and the nodes in the neighboring control volumes. Considering all such linear equations together produces a matrix equation of the form $\mathbf{A}\boldsymbol{\phi} = \mathbf{b}$.

DISCRETIZATION

Initially it will be assumed that the mesh over which the conservation equations are being discretized is fully non-skew. A non-skew mesh is one in which for every pair of adjacent elements

- The line connecting the centroids of the adjacent elements is parallel to the normal of the face between the elements.
- The line connecting the centroids of the adjacent elements intersects the face between the elements at the face centroid.

The transient, convection, diffusion, and source, terms are each discretized to approximate their integrals. The discretized form of the full equation is then obtained by adding together all these contributions. Assuming values at the centers of control adjacent control volumes A and B :

The transient term is

$$\frac{\partial(\rho\phi)}{\partial t}$$

The discretized transient term is

$$\frac{V_A \rho_A \phi_A - V_A^0 \rho_A^0 \phi_A^0}{\Delta t}$$

where V_A is the volume of element A , ρ_A is the density, and ϕ_A is the discrete variable at the center of element A .

The convection term is

$$\nabla \cdot (\rho \mathbf{v} \phi)$$

The discretized convection term is

$$\sum_f \rho_f (\mathbf{v} \cdot \mathbf{n})_f A_f [\alpha_f \phi_A + (1 - \alpha_f) \phi_B]$$

where ρ_f is the density of the upwind element, A_f is the area of face f , and α_f is an arithmetic averaging factor for ϕ at the face f .

The diffusion term is

$$\nabla \cdot (\Gamma_\phi \nabla \phi)$$

The discretized diffusion term is

$$\sum_f (\Gamma_\phi)_f A_f \frac{\phi_A - \phi_B}{d_{AB}}$$

where d_{AB} is the distance between the centroids of elements A and B .

The discretized source term is

$$V_A (S_C - S_A \phi_A)$$

where S_C and S_A can be functions of any stored value including ϕ .

DIFFERENCING SCHEMES

If only the convection and diffusion terms are considered, and using arithmetic averaging in the evaluation of the face value of ϕ in the convection term, the discretized equation becomes

$$\sum_f A_f \left\{ \rho_f (\mathbf{v} \cdot \mathbf{n})_f [\alpha_f \phi_A + (1 - \alpha_f) \phi_B] + (\Gamma_\phi)_f \left(\frac{\phi_A - \phi_B}{d_{AB}} \right) \right\} = 0$$

This can be rewritten as

$$a_A \phi_A = \sum_{nb} a_{nb} \phi_{nb}$$

with

$$F_f = a_f \rho_f (\mathbf{v} \cdot \mathbf{n})_f$$

$$D_f = \frac{A_f (\Gamma_\phi)_f}{d_{AB}}$$

where F_f is the strength of the convection of ϕ , and D_f is the diffusion conductance we can write

$$a_{nb} = D_f - (1 - \alpha_f) F_f$$

$$a_A = \sum_{nb} a_{nb} + \sum_f F_f$$

In the upwind scheme, by Courant, et.al.,⁸ there is no alteration to the handling of the diffusion term but the convection term uses the upwind nodal value of ϕ for the estimated interface value. Thus

$$\phi_f = \phi_A \text{ if } F_f > 0$$

and

$$\phi_f = \phi_B \text{ if } F_f < 0$$

This gives the coefficients, above, as

$$a_{nb} = D_f + \max(-F_f, 0)$$

$$a_A = \sum_{nb} a_{nb} + \sum_f F_f$$

The upwind scheme guarantees that the solution is bounded, but it often over predicts the diffusive effects.

There are many differencing schemes that have been developed to improve on the accuracy of the solution. All schemes can be written in terms of the Peclet number, which is the ratio of strength of the convection to the strength of the diffusion. In one dimension it is given by

$$P = \frac{\rho v L}{\Gamma_\phi}$$

Integrated over an entire control volume, it is given as

$$P_f = \frac{F_f}{D_f} = \frac{\rho_f (\mathbf{v} \cdot \mathbf{n})_f}{(\Gamma_\phi)_f}$$

The discretized convection-diffusion equation can now be written

$$\sum_f [D_f A(|P_f|) + \max(-F_f, 0)] (\phi_A - \phi_B) + F_f \phi_A = 0$$

First order differencing schemes in terms of the Peclet number are given in terms of $A(|P|)$ in Table 1.

Table 1: First order differencing schemes

Scheme	$A(P)$
Central differencing	$1- P /2$
Upwind	1
Hybrid ⁹	$\text{Max}(0,1- P /2)$
Power law ¹⁰	$\text{Max}(0, (1- P /10)^5)$
Exponential	$ P / \exp(P) - 1$

The full discretized conservation equation is given by

$$\frac{V_A \rho_A \phi_A - V_A^0 \rho_A^0 \phi_A^0}{\Delta t} + \sum_f \left\{ [D_f A(|P_f|) + \max(-F_f, 0)] (\phi_A - \phi_B) + F_f \phi_A \right\} = V_A (S_C - S_A \phi_A)$$

The continuity equation is given by

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$

which can also be discretized to

$$\frac{V_A \rho_A - V_A^0 \rho_A^0}{\Delta t} + \sum_f F_f = 0$$

This can be substituted into the full conservation equation to yield

$$a_A \phi_A = \sum_{nb} a_{nb} \phi_{nb} + b_A$$

where the summation is over all elements that share a face with element A , and

$$a_{nb} = D_f A(|P_f|) + \max(-F_f, 0)$$

$$b_A = S_C V_A + \frac{V_A^0 \rho_A^0 \phi_A^0}{\Delta t}$$

$$a_A = \sum_{nb} a_{nb} + \frac{V_A^0 \rho_A^0}{\Delta t} + S_A V_A$$

With adjustments at the domain boundaries, these equations may be used to calculate ϕ at all the element centroids.

PRESSURE-MOMENTUM COUPLING

Pressure source terms in the momentum equations may produce problems. If the velocities and the scalar variable of pressure are both defined at the center of a control volume, a highly non-uniform pressure field can act like a uniform field in the discretized momentum equations. This is obviously nonphysical. The staggered grid¹¹ and the Rhie-Chow¹² interpolation methods are often used to overcome this problem. The staggered grid is used in this thesis.

The idea of the staggered grid is to evaluate scalar variables, such as pressure, density, or temperature at ordinary nodal points in the center of the volume. Velocity components, however, are calculated on staggered grids centered on the cell faces. This is shown in Figure 2 for the two-dimensions.

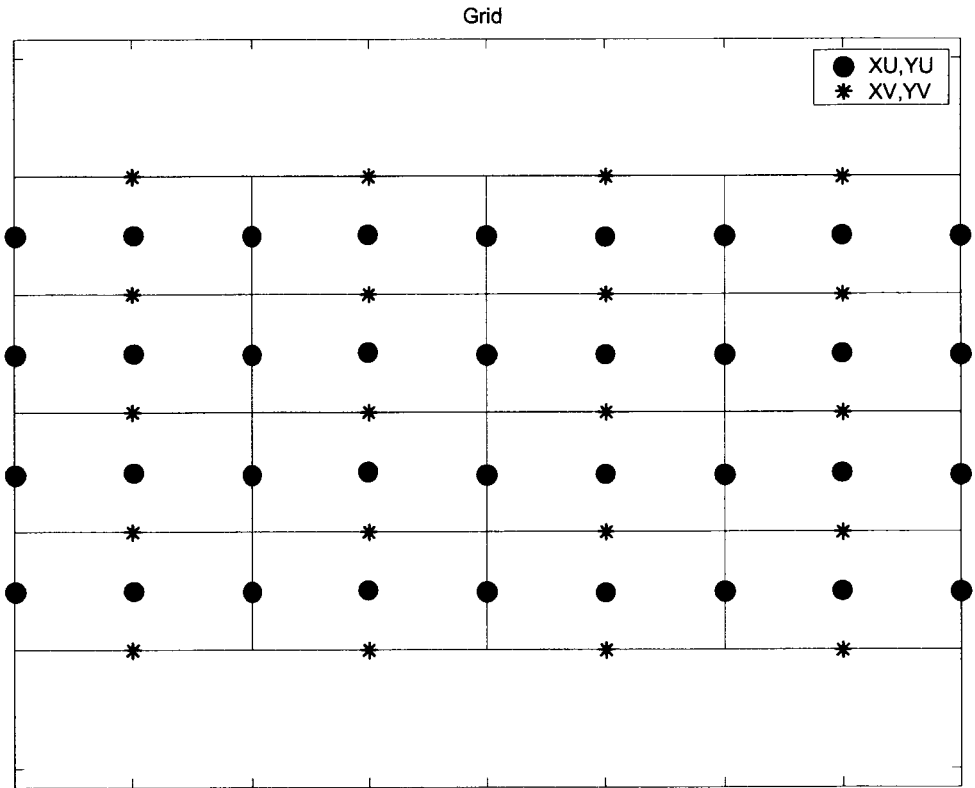


Figure 2: Staggered grid

The scalar variables are stored at the nodes marked by a bullet. The velocities are defined at the cell faces in between the nodes. The circles on the vertical grid lines indicate the locations for the horizontal u -velocities. The control volumes for u are centered about these circles. The asterisks on the horizontal grid lines indicate the locations for the vertical v -velocities. The control volumes for v centered about these asterisks. Note that the u and v control volumes are different from the scalar control volumes and different from each other.

With this layout, the cell faces are designated by lower-case letters. The cell centers are designated by upper-case letters. This is shown for a single cell in Figure 3.

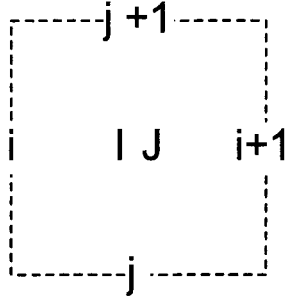


Figure 3: Staggered grid indexing

The scalar node at the cell center is designated by (I, J) . The u -velocities are stored at the east and west faces. The u -velocity on the west face is designated by (i, J) . Similarly, the v -velocity on the south face is designated by (I, j) .

Expressed in this coordinate system, the discretized u -momentum equation for the velocity at location (i, J) is given by

$$a_{i,J}u_{i,J} = \sum a_{nb}u_{nb} - \frac{p_{I,J} - p_{I-1,J}}{\delta x_u} \Delta V_u + \bar{S} \Delta V_u$$

where ΔV_u is the volume of the u -cell. The v -momentum equation is similarly

$$a_{I,j}v_{I,j} = \sum a_{nb}v_{nb} - \frac{p_{I,J-1} - p_{I,J}}{\delta y_v} \Delta V_v + \bar{S} \Delta V_v$$

SIMPLE ALGORITHM

Given a pressure field p , discretized momentum equations can be solved to obtain the velocity fields. If the pressure field is correct, the resulting velocity field will satisfy continuity. The problem is that there is no conservation equation in which the pressure replaces ϕ in the general equation. The SIMPLE algorithm,¹³ which is a predictor-corrector method, is used to provide momentum-pressure coupling.

Given estimated element center values of pressure, p^* , the velocity components u^* and v^* can be improved so that:

$$p = p^* + p'$$

$$u = u^* + u'$$

$$v = v^* + v'$$

The pressure corrector value p' can be calculated by solving the discretized continuity equation

$$a_A p'_A + \sum_{nb} a_{nb} p'_{nb} = b_A$$

For the (u, v) velocity field, this is given by

$$a_{i,j} (u_{i,j} - u_{i,j}^*) = \sum a_{nb} (u_{nb} - u_{nb}^*) + [(p_{I-1,j} - p_{I-1,j}^*) - (p_{I,j} - p_{I,j}^*)] A_{i,j}$$

$$a_{I,j} (v_{I,j} - v_{I,j}^*) = \sum a_{nb} (v_{nb} - v_{nb}^*) + [(p_{I,j-1} - p_{I,j-1}^*) - (p_{I,j} - p_{I,j}^*)] A_{I,j}$$

where $A_{i,j}$ is the area of the east or west control volume face and $A_{I,j}$ is the area of the north or south face. Using the correction formulae, these may be rewritten as

$$a_{i,j} u'_{i,j} = \sum a_{nb} u'_{nb} + (p'_{I-1,j} - p'_{I,j}) A_{i,j}$$

$$a_{I,j}v'_{I,j} = \sum a_{nb}v'_{nb} + (p'_{I,J-1} - p'_{I,J})A_{I,j}$$

The main approximation of the SIMPLE algorithm is to drop the summation terms for the velocity corrections. This gives

$$u'_{i,J} = d_{i,J}(p'_{I-1,J} - p'_{I,J})$$

$$v'_{I,j} = d_{I,j}(p'_{I,J-1} - p'_{I,J})$$

where $d_{i,J} = \frac{A_{i,J}}{a_{i,J}}$ and $d_{I,j} = \frac{A_{I,j}}{a_{I,j}}$.

The velocity corrections are then obtained as

$$u_{i,J} = u^*_{i,J} + d_{i,J}(p'_{I-1,J} - p'_{I,J})$$

$$v_{I,j} = v^*_{I,j} + d_{I,j}(p'_{I,J-1} - p'_{I,J})$$

MODIFICATION FOR SOLID MECHANICS

Fallah, et.al.,⁴ have modified the solution procedure, above, for solid mechanics problems. The momentum equations for solids in steady state are

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} + S_{Mx} = 0$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} + S_{My} = 0$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + S_{Mz} = 0$$

Stress can be related to the deformation derivatives with Hooke's Law.

$$\sigma_{xx} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \left\{ \frac{\partial u_x}{\partial x} - \alpha(T - T_r) + \frac{\nu}{1-\nu} \left[\frac{\partial u_y}{\partial y} - \alpha(T - T_r) \right] + \frac{\nu}{1-\nu} \left[\frac{\partial u_z}{\partial z} - \alpha(T - T_r) \right] \right\}$$

$$\sigma_{yy} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \left\{ \frac{\nu}{1-\nu} \left[\frac{\partial u_x}{\partial x} - \alpha(T - T_r) \right] + \frac{\partial u_y}{\partial y} - \alpha(T - T_r) + \frac{\nu}{1-\nu} \left[\frac{\partial u_z}{\partial z} - \alpha(T - T_r) \right] \right\}$$

$$\sigma_{zz} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \left\{ \frac{\nu}{1-\nu} \left[\frac{\partial u_x}{\partial x} - \alpha(T - T_r) \right] + \frac{\nu}{1-\nu} \left[\frac{\partial u_y}{\partial y} - \alpha(T - T_r) \right] + \frac{\partial u_z}{\partial z} - \alpha(T - T_r) \right\}$$

$$\sigma_{xy} = \sigma_{yx} = G \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right)$$

$$\sigma_{xz} = \sigma_{zx} = G \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \right)$$

$$\sigma_{yz} = \sigma_{zy} = G \left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \right)$$

The normal stresses can be rearranged

$$\sigma_{xx} = \sigma_{ave} + 2G \frac{\partial u_x}{\partial x} - \frac{2G}{3} \nabla \cdot \mathbf{u}$$

$$\sigma_{yy} = \sigma_{ave} + 2G \frac{\partial u_y}{\partial y} - \frac{2G}{3} \nabla \cdot \mathbf{u}$$

$$\sigma_{zz} = \sigma_{ave} + 2G \frac{\partial u_z}{\partial z} - \frac{2G}{3} \nabla \cdot \mathbf{u}$$

where \mathbf{u} is the displacement vector and σ_{ave} is the mean of the normal stresses

$$\sigma_{ave} = \frac{\sigma_{xx} + \sigma_{yy} + \sigma_{zz}}{3} = \frac{E}{3(1-2\nu)} [\nabla \cdot \mathbf{u} - 3\alpha(T - T_r)]$$

σ_{ave} can be interpreted as a pressure quantity

$$p_s = -\sigma_{ave} = -\frac{E}{3(1-2\nu)} [\nabla \cdot \mathbf{u} - 3\alpha(T - T_r)]$$

Substitution gives

$$\sigma_{xx} = -p_s + 2G \frac{\partial u_x}{\partial x} - \frac{2G}{3} \nabla \cdot \mathbf{u}$$

$$\sigma_{yy} = -p_s + 2G \frac{\partial u_y}{\partial y} - \frac{2G}{3} \nabla \cdot \mathbf{u}$$

$$\sigma_{zz} = -p_s + 2G \frac{\partial u_z}{\partial z} - \frac{2G}{3} \nabla \cdot \mathbf{u}$$

$$\sigma_{xy} = \sigma_{yx} = G \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right)$$

$$\sigma_{xz} = \sigma_{zx} = G \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \right)$$

$$\sigma_{yz} = \sigma_{zy} = G \left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \right)$$

Substituting these equations back into the momentum equations gives

$$-\frac{\partial}{\partial x} \left(p_s - \frac{G}{3} \nabla \cdot \mathbf{u} \right) + \nabla \cdot (G \nabla u_x) + S_{Mx} = 0$$

$$-\frac{\partial}{\partial y} \left(p_s - \frac{G}{3} \nabla \cdot \mathbf{u} \right) + \nabla \cdot (G \nabla u_y) + S_{My} = 0$$

$$-\frac{\partial}{\partial z} \left(p_s - \frac{G}{3} \nabla \cdot \mathbf{u} \right) + \nabla \cdot (G \nabla u_z) + S_{Mz} = 0$$

The Navier-Stokes equations without convection terms (i.e., low Reynolds number)

are

$$-\frac{\partial}{\partial x} [p - (\mu + \lambda) \nabla \cdot \mathbf{v}] + \nabla \cdot (\mu \nabla v_x) + S_{Mx} = 0$$

$$-\frac{\partial}{\partial y} [p - (\mu + \lambda) \nabla \cdot \mathbf{v}] + \nabla \cdot (\mu \nabla v_y) + S_{My} = 0$$

$$-\frac{\partial}{\partial z} [p - (\mu + \lambda) \nabla \cdot \mathbf{v}] + \nabla \cdot (\mu \nabla v_z) + S_{Mz} = 0$$

where

μ is the dynamic viscosity

λ is the bulk or second viscosity

A comparison shows that $\mu + \lambda$ corresponds to $G/3$, μ corresponds to G , and the velocity vector, \mathbf{v} , corresponds to the displacement vector, \mathbf{u} . Since the density variation in solids is negligible, the continuity equation may be neglected. These comparisons give

$$p = p_s - \frac{G}{3} \nabla \cdot \mathbf{u}$$

Substitution gives

$$p = -G \left[\frac{1}{1-2\nu} \nabla \cdot \mathbf{u} - \frac{2(1+\nu)}{1-2\nu} \alpha (T - T_r) \right]$$

$$\nabla \cdot \mathbf{u} + \frac{1-2\nu}{G} p - 2\alpha(1+\nu)(T - T_r) = 0$$

For an incompressible fluid, the continuity equation is

$$\nabla \cdot \mathbf{v} = 0$$

The discretization is

$$a_A p'_A + \sum_{nb} a_{nb} p'_{nb} = b_A$$

where the transient term vanishes in b_A and ρ is set to 1. $\nabla \cdot \mathbf{u}$ in the equation above is substituted by the discretized equation to give

$$a_A p'_A + \sum_{nb} a_{nb} p'_{nb} - b_A + \frac{1-2\nu}{G} (p'_A + p_A^*) - 2\alpha(1+\nu)(T_A - T_r) = 0$$

Finally, we get

$$a_A^{\text{mod}} p'_A + \sum_{nb} a_{nb} p'_{nb} = b_A^{\text{mod}}$$

where

$$a_{nb} = -\rho_f A_f^2 \sum_{i=1}^3 \frac{n_i^2}{a_i}$$

$$a_A^{\text{mod}} = -\sum_{nb} a_{nb} + \frac{1-2\nu}{G}$$

$$b_A^{\text{mod}} = -\sum_f A_f \rho_f (\mathbf{v}^* \cdot \mathbf{n})_f - \frac{1-2\nu}{G} p_A^* + 2\alpha(1+\nu)(T_A - T_r)$$

TEST CASES

CASE 1

A 2-dimensional test case was run with the geometry shown in Figure 4. The right edge was clamped. An enforced displacement of 0.01-inch was applied to the left edge. G was given as $10e6$ psi, and ν as 0.5. A grid of 40×40 rectangular elements was constructed. The solution for horizontal displacements is shown in Figure 5. The solution for vertical displacements is shown in Figure 6.

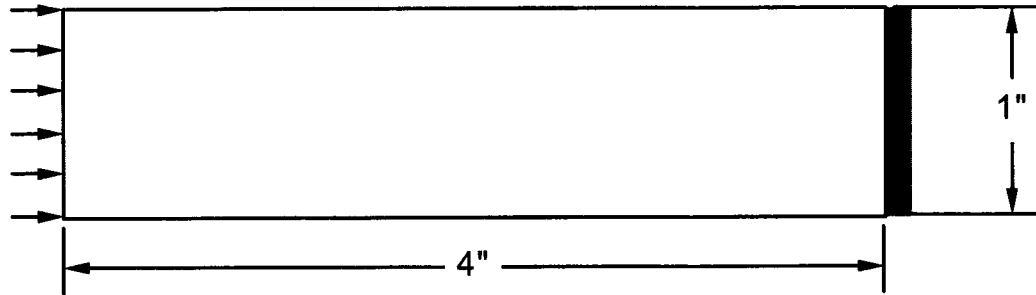


Figure 4: 2-D test case

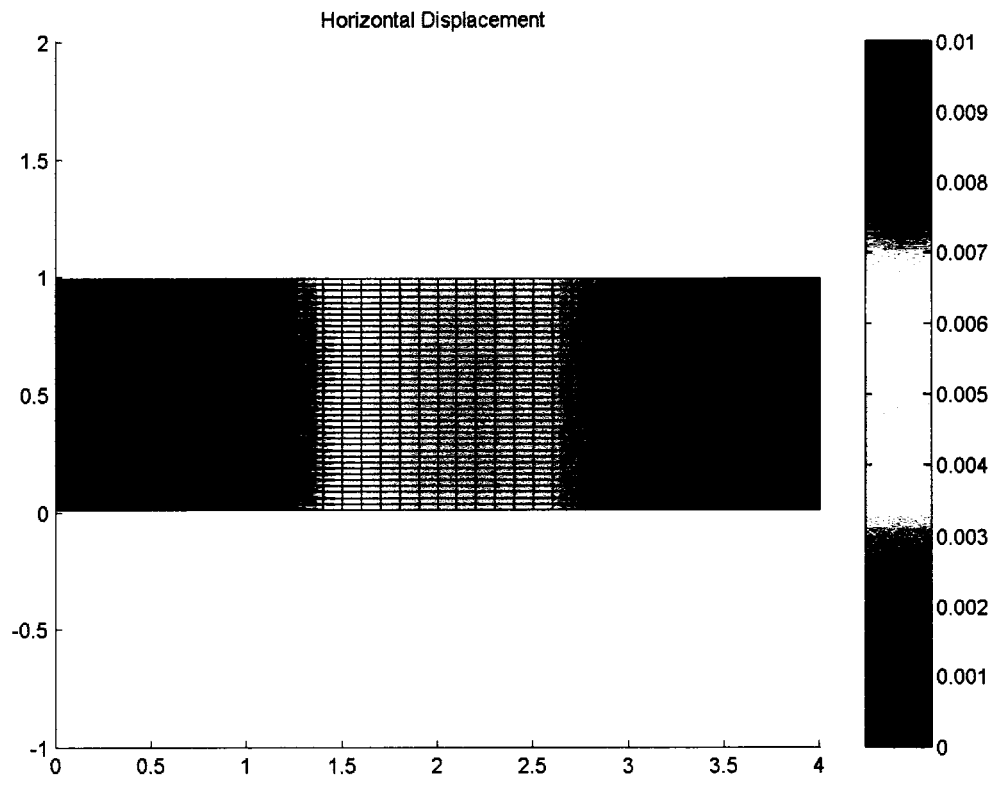


Figure 5: FV horizontal displacements

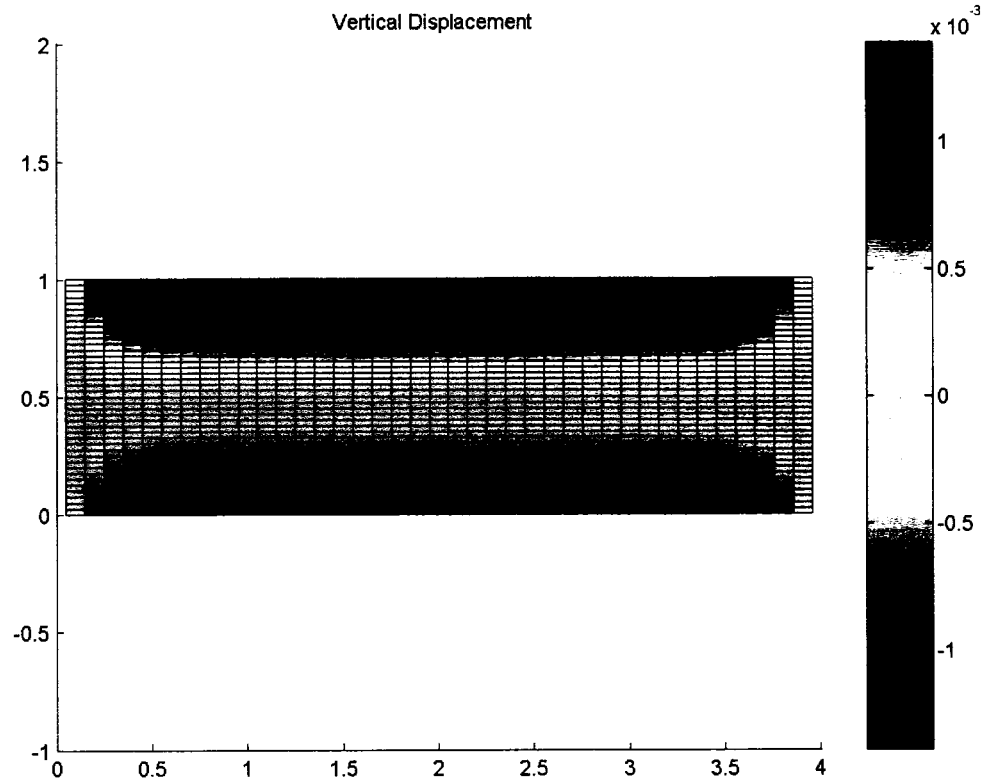


Figure 6: FV vertical displacements

The displacements compare favorably with FE results. A plane strain FE analysis was performed for the test case with $\nu = 0.499$. MSC/Nastran³ was used to perform the solution. The FE model also consisted of 40 x 40 elements. The vertical displacements from FE are shown in Figure 7.

C:\Bryce\thesis*_model.mf1
 RESULTS: 1- B.C. 1, DISPLACEMENT_1, RESTRAINT SET 1
 DISPLACEMENT - Y M N: -1.34E-03 MAX: 1.34E-03
 DEFORMATION: 1- B.C. 1, DISPLACEMENT_1, RESTRAINT SET 1
 DISPLACEMENT - Y M N: -1.34E-03 MAX: 1.34E-03
 FRAME OF REF: PART

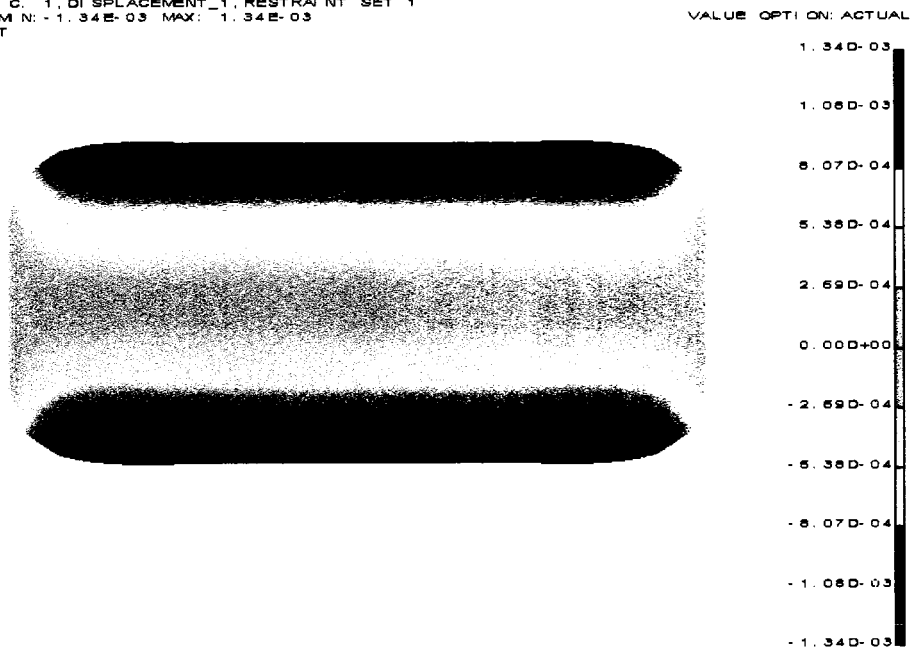


Figure 7: Vertical displacements from FE

Substituting

$$p_s = p + \frac{G}{3} \nabla \cdot \mathbf{u}$$

into the normal stress equations gives

$$\sigma_{xx} = -p + 2G \frac{\partial u_x}{\partial x} - G \nabla \cdot \mathbf{u}$$

$$\sigma_{yy} = -p + 2G \frac{\partial u_y}{\partial y} - G \nabla \cdot \mathbf{u}$$

$$\sigma_{zz} = -p + 2G \frac{\partial u_z}{\partial z} - G \nabla \cdot \mathbf{u}$$

The Von Mises stress for plane stress is

$$\sqrt{(\sigma_1^2 + \sigma_2^2) - \sigma_1 \sigma_2}$$

Substitution gives

$$\sqrt{p^2 + 3G^2 \left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \right)^2}$$

The Von Mises stress for plane strain is

$$\sqrt{(\sigma_1^2 + \sigma_2^2)(\nu^2 - \nu + 1) - \sigma_1 \sigma_2 (2\nu^2 - 2\nu - 1)}$$

Substitution gives

$$\sqrt{3p^2 + G^2 \left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \right)^2 (4\nu^2 - 4\nu + 1)}$$

For an incompressible material, the Von Mises stress for plane strain becomes $p\sqrt{3}$.

Table 2 gives a comparison of the maximum vertical displacements and the maximum Von Mises stresses for the two analyses. The correlation of the stress distributions is reasonable. Plots of the stress distribution are shown in Figure 8 and Figure 9.

Table 2: Comparison of FV and FE results

	Max vertical displacement	Max Von Mises stress
FV	0.00138	1.35e5
FE	0.00134	1.66e5

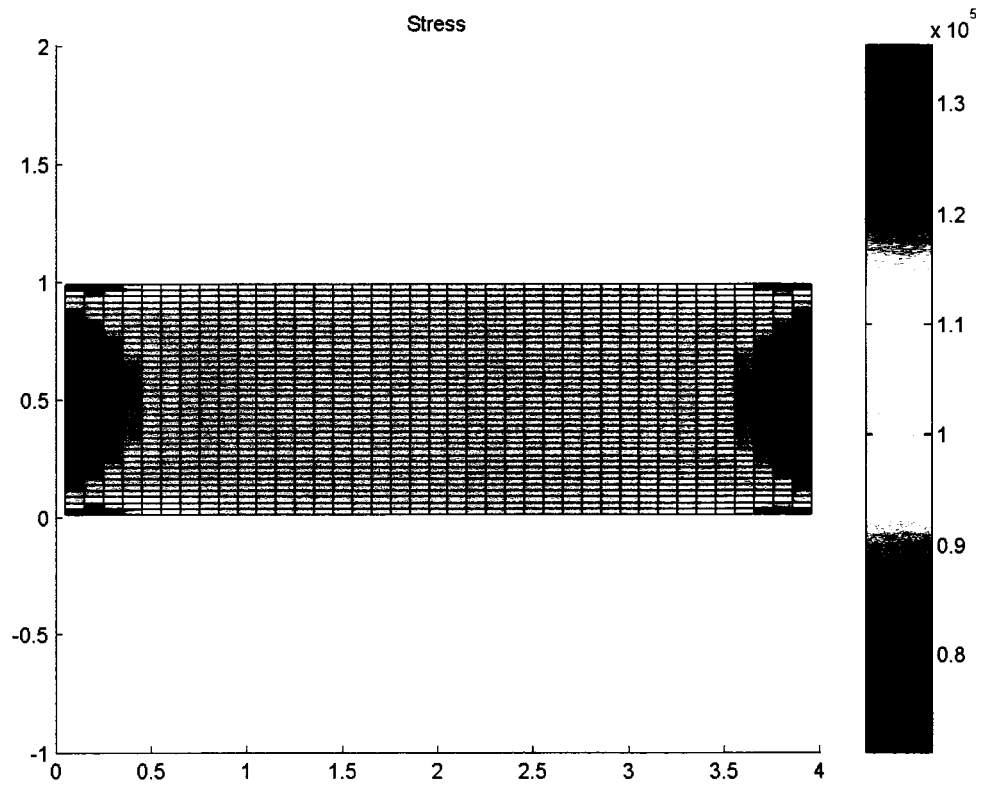


Figure 8: Finite volume stress distribution

C:\Bryce\thesis_model.mf1
RESULTS: 3- B.C. 1, STRESS_3, RESTRAINT SET 1
STRESS - VON MISES MIN: 4.00E+02 MAX: 1.00E+05
DEFORMATION: 1- B.C. 1, DISPLACEMENT_1, RESTRAINT SET 1
DISPLACEMENT - MAG MIN: 0.00E+00 MAX: 1.00E-02
FRAME OF REF: PART

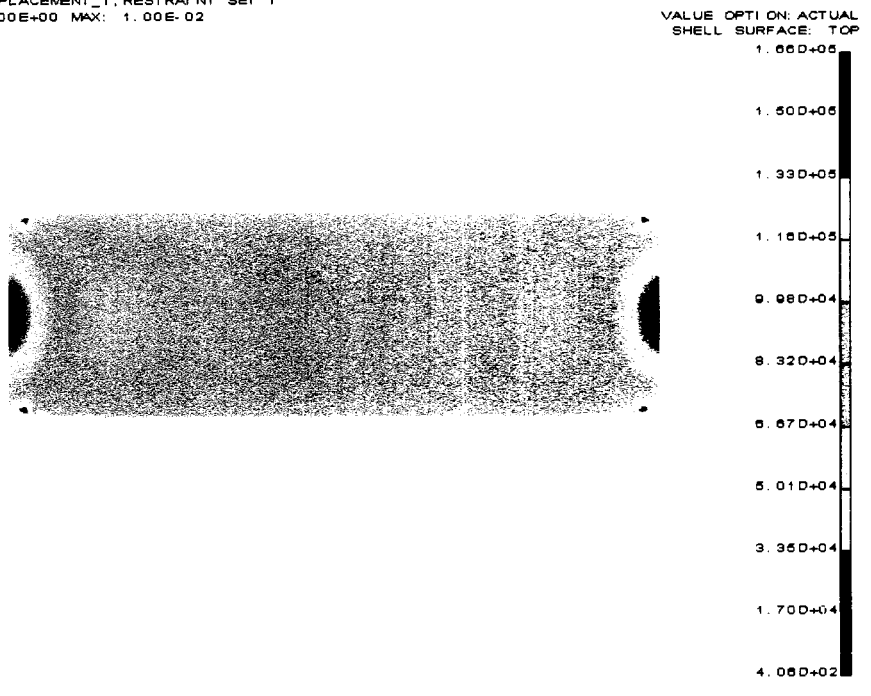


Figure 9: Finite element stress distribution

CASE 2

An axisymmetric test case using 40×40 elements with the geometry shown in Figure 10 was run. The axis of rotation for the FV solution was the bottom axis. The right boundary was fixed. An enforced displacement of 0.01 was applied to the left boundary. Figure 11 shows the radial displacements. Figure 12 shows the Von Mises stress.

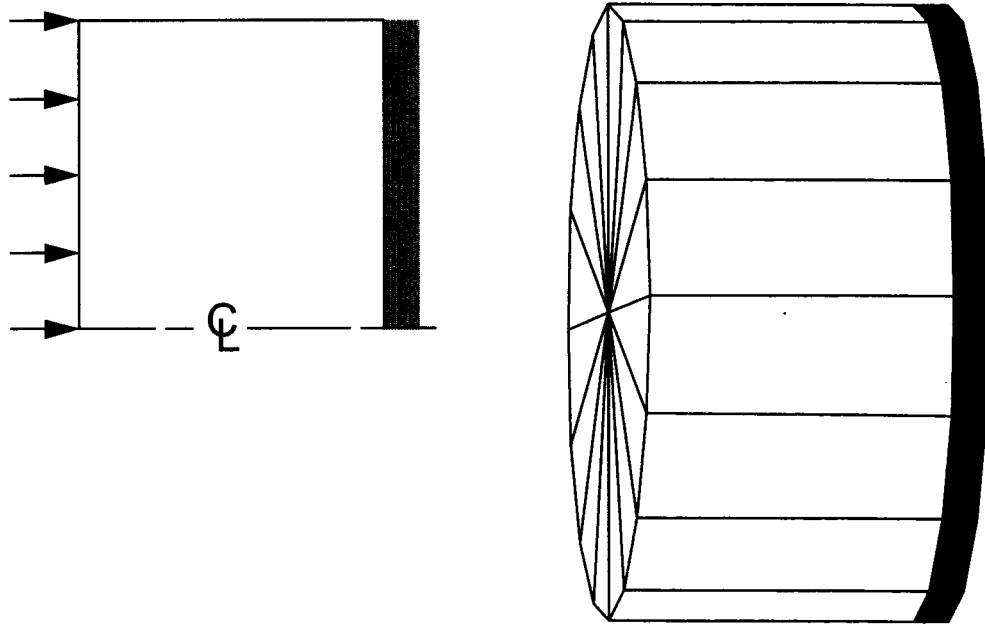


Figure 10: Axisymmetric model

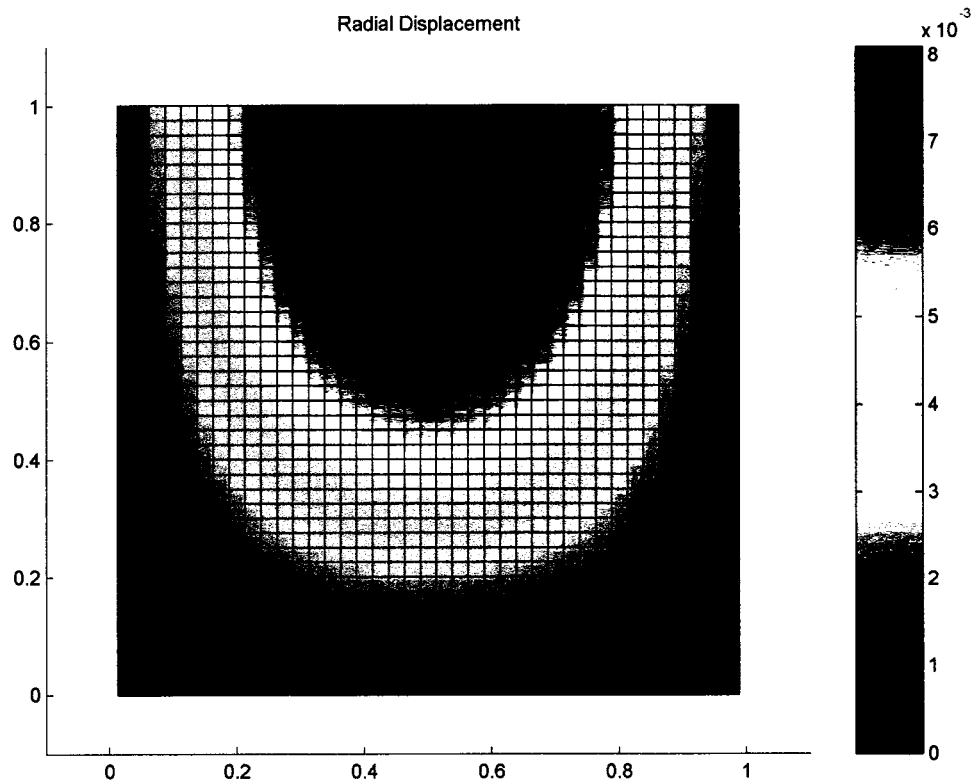


Figure 11: FV radial displacements

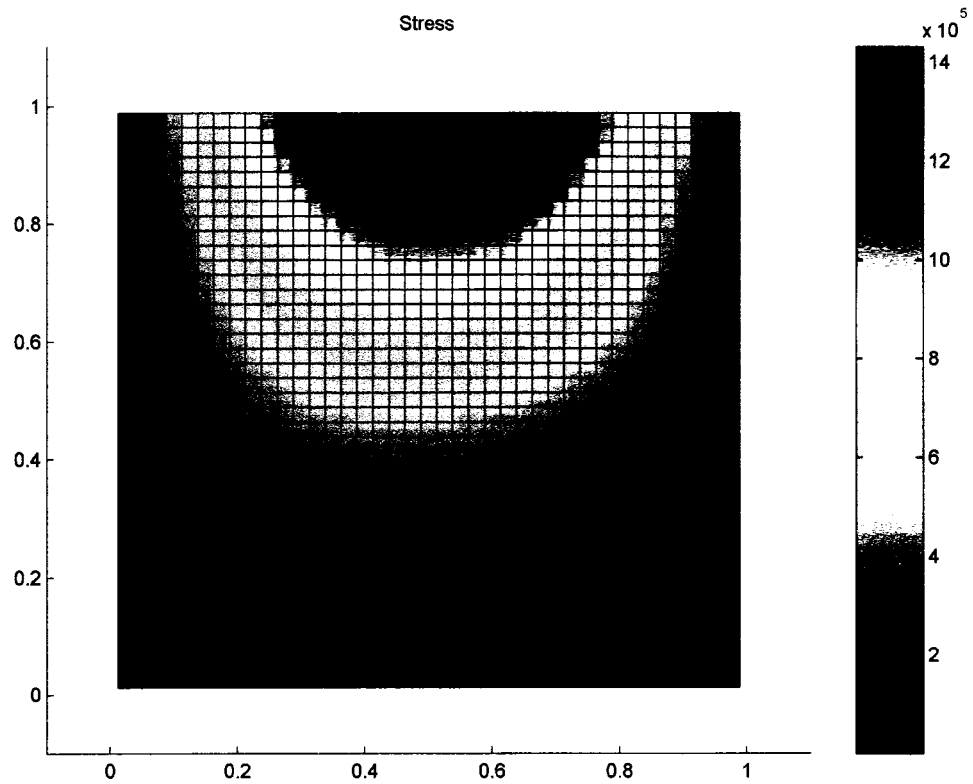


Figure 12: FV Von Mises stress

The comparative FE model had the axis of rotation on the left edge, the bottom edge clamped, and the enforced displacement of 0.01 applied to the top edge. The radial displacements are shown in Figure 13. The Von Mises stress is shown in Figure 14.

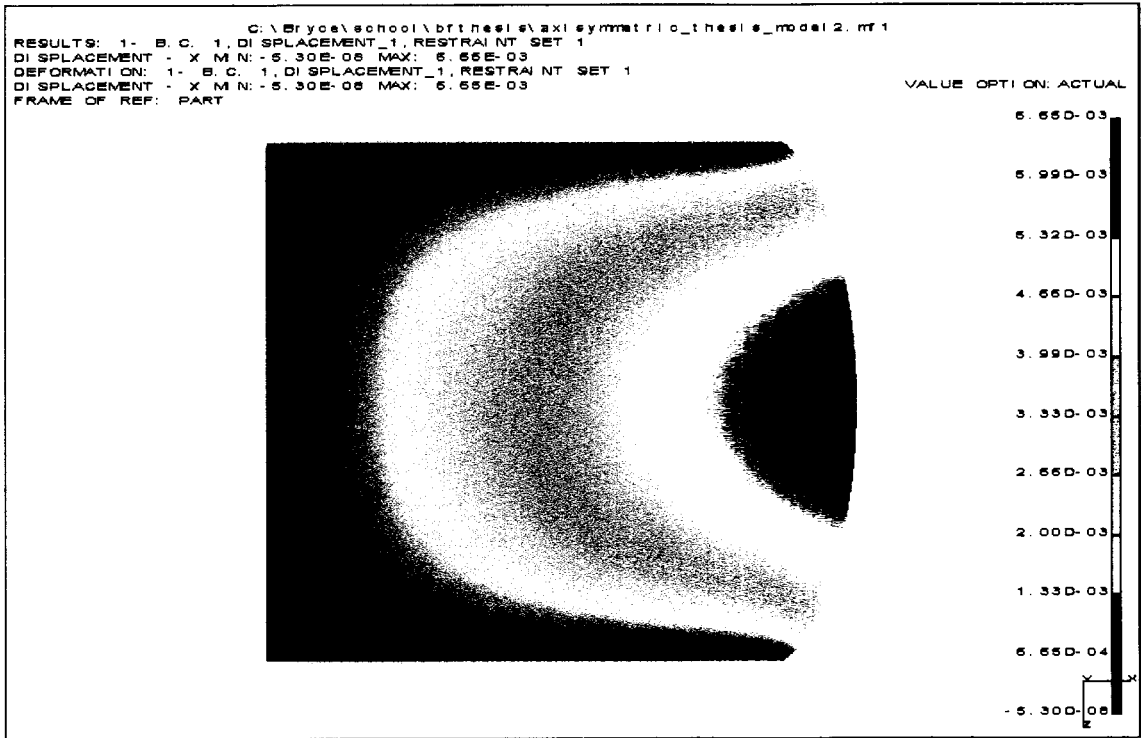


Figure 13: FE radial displacements

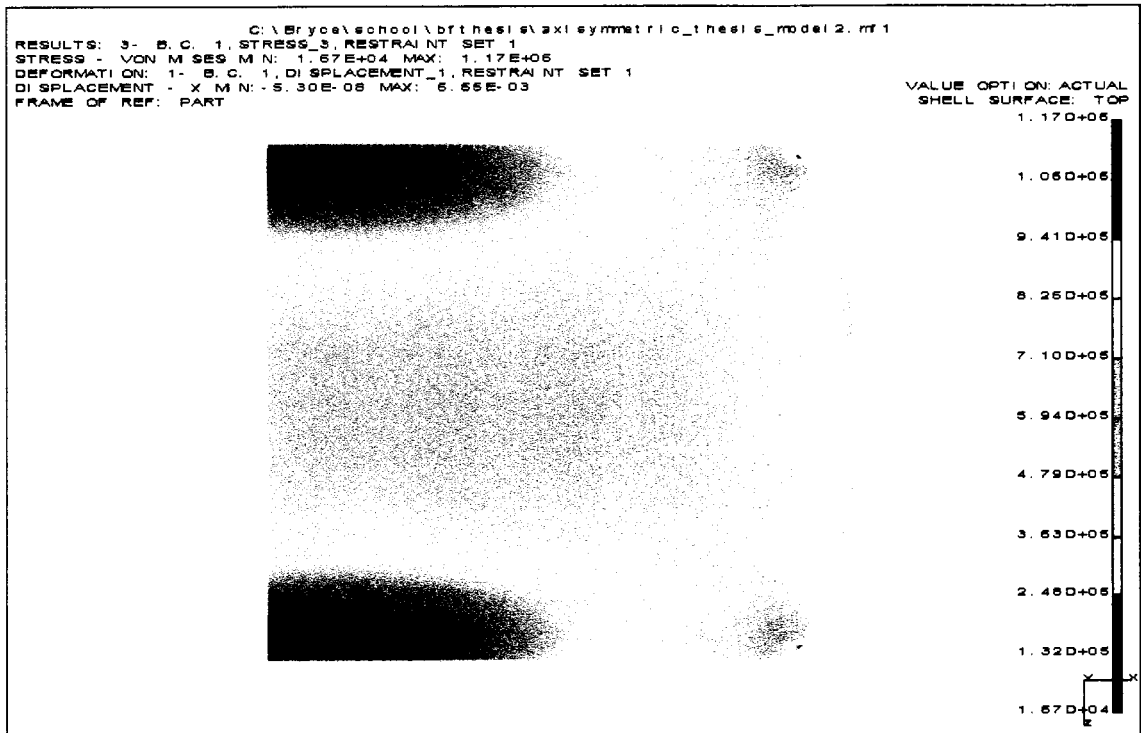


Figure 14: FE Von Mises stress

As with the plane strain model, there is good agreement between the FV and FE axisymmetric solutions. Table 3 shows the comparisons of displacement and stress for the two solution methods.

Table 3 : Comparison of axisymmetric FV and FE results

	Max radial displacement	Max Von Mises stress
FV	0.00676	1.39e6
FE	0.00665	1.17e6

DISCUSSION

The results for the two simple cases presented above show promise for using FV to model incompressible materials. Since the formulation uses flow continuity through the elements, it is not subject to the nodal lock-up that FE-based methods encounter.

Finite volume requires that the initial conditions and boundary conditions from CFD are used to describe the problem.

Initial conditions must be specified throughout the solution region. That is, \mathbf{u} must be specified for all elements at time $t = 0$. The initial conditions should satisfy continuity throughout the region.

Boundary conditions must be correctly chosen for the problem. There are three main types of boundary conditions in CFD. They are no slip, free surface, and outflow. In the context of solid mechanics, no slip means the displacement at the boundary equals the local wall displacement. A free surface means that the tangential stress components are zero at the boundary. This is the inflow condition when \mathbf{u} is given as a function of position. It is a symmetrical or slip boundary condition when $\partial\phi/\partial n = 0$. For the outflow boundary condition, pressure is typically specified along with $\nabla \cdot \mathbf{u} = 0$. Even when the pressure is not necessarily known, however, the solution will typically converge.

CONCLUSION

This thesis is based on the theory of Fallah, et. al.,⁴ for the modification of CFD for solid mechanics. The theory shows promise for accurate modeling of incompressible materials. Work was done to implement the theory in a 2-D finite volume solver written in Matlab. This Matlab code is a modification of finite volume code¹⁴ that was previously released under the GNU General Public License (GPL). Thus, the code in this thesis is also released under the GPL. Comparison of solutions from the code with finite element solutions for two simple structures shows good correlation for nearly incompressible materials (Poisson's Ratio = 0.499).

Practitioners of solid mechanics may use the code from this thesis to model the behavior of incompressible solid materials. Material stiffness may easily be calculated from the pressure and displacement. Until now, modeling incompressible solids has been very difficult to accomplish with the tools available in the public domain.

FUTURE WORK

The problem solution and boundary conditions in the current work have been implemented only for the simple 2-dimensional 4-sided orthogonal case. Multiple boundary conditions for domains containing steps and non-orthogonal shapes should be implemented. Irregular grids may also be implemented.

Eventually, this work may be used as the basis for a 3-dimensional modeler.

BIBLIOGRAPHY

-
- ¹ Laursen, T.A., S.W. Attaway, and R.I. Zadoks. "SEACAS Theory Manuals: Part III. Finite Element Analysis in Nonlinear Solid Mechanics." SAND98-1760/3. Unlimited Release. August 1998.
 - ² Attaway, S.W., et.al., "PRONTO3D Users' Instructions: A Transient Dynamic Code for Nonlinear Structural Analysis." SAND98-1361. Unlimited Release. August 1998.
 - ³ MSC/Marc. Vers. 2001. Computer Software. MSC.Software Corporation, 2001. Windows or Linux.
 - ⁴ Fallah, N.A., C. Bailey, and M. Cross. "CFD APPROACH FOR SOLID MECHANICS ANALYSIS." *European Congress on Computational Methods in Applied Sciences and Engineering*. ECCOMAS 2000. Barcelona. 11-14 Sep. 2000.
 - ⁵ Wheel, M.A. "A Finite Volume Approach to Modeling Incompressible Materials." *Finite Element Analysis of Elastomers* 1 (1999): 105-116.
 - ⁶ Demirdzic, I., S. Muzferija, "Finite Volume methods for stress analysis in complex domains," *International Journal of Numerical Methods in Engineering* 37 (1994): 3751-3766.
 - ⁷ Suri, M. "Analytic and Computational Assessment of Locking in the hp Finite Element Method." Department of Mathematics and Statistics. University of Maryland. Baltimore, MD 21228. USA. Sep. 1994.
 - ⁸ Courant, R., E. Isaacson, and M. Rees. "On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Difference." *Communications in Pure and Applied Mathematics* 5 (1952): 243-255.
 - ⁹ Spalding, D.B. "A Novel Finite Difference Formulation for Differential Expressions Involving both First and Second Derivatives." *International Journal for Numerical Methods in Engineering* 4 (1972): 551-559.
 - ¹⁰ Patankar, S.V. "A Calculation Procedure for Two Dimensional Elliptic Situations." *Numerical Heat Transfer* 2 (1979): n.p.
 - ¹¹ Versteeg, H.K., and W. Malalasekera, "An Introduction to Computational Fluid Dynamics, The Finite Volume Method." n.p.: Prentice Hall, 1995.
 - ¹² Rhie, C.M., and W. L. Chow, "Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation." *AIAA Journal* 21(11) (1983): 1525-1532.

-
- ¹³ Murthy, J.Y. "COMPUTATIONAL FLUID MECHANICS, Draft Notes."
Department of Mechanical Engineering. Carnegie Mellon University. Spring
2001.
- ¹⁴ Wesseling, P. "Principles of Computational Fluid Dynamics." Springer-Verlag.
Berlin: 2001. ISBN 3-540-67853-0.

APPENDIX

The code that follows calculates and creates the plots in Figure 5, 6 and 8.

```
*****
% cfd.m: two dimensional finite volume code modified for solid mechanics
% solution, Bryce Fowler 2003
%
% Numerical solution of 2-D instationary Navier-Stokes equations.
% Rectangular domain; nonuniform Cartesian staggered grid.

% Original code Copyright 2001 P. Wesseling
% This program and its subprograms may be freely used, modified and distributed
% under the GNU General Public License: http://www.gnu.org/copyleft/gpl.html

% Theory is given in Chapter 6 of
% Principles of Computational Fluid Dynamics, by P. Wesseling
% Springer-Verlag, Berlin etc., 2001. ISBN 3-540-67853-0
% See http://dutita0.twi.tudelft.nl/nw/users/wesseling/
%
clear all;
global adj Re geval n u0 v0 J K yu yv alpha
Re = 1; % Reynolds number is:  $Re = D \cdot V \cdot \rho / \mu$ , D=length, V=velocity,
rho=density, mu=viscosity
geval = 1;
omega = 1/2; % Omega-scheme for time-stepping
E = 30e6;
nu = 0.5;
G = E/(2*(1+nu))
adj = (1-2*nu)/G;

dt = 0.01; % Time step
tend = 40; % End time
converged = 1e-6; % Convergence criterion
umax = 0.01; % Estimates of max. velocity components required for stability
vmax = 0.01;
makemovie = 0;

xbound = [0,4]; % horizontal boundary in order of increasing x.
ybound = [0,1]; % Similar to xbound in y-direction.
nx = [10]; % Number of cells along x-segments.
ny = [10]; % Number of cells along y-segments.

% Type of boundary condition:
% 0: free
% 1: no-slip
% 2: inflow
% 3: outflow
% Give type of boundary condition at lower and upper boundaries
xbc = [0; 0];

% Give type of boundary condition along left and right boundaries
ybc = [2; 1];

create_grid % Generate the FV grid

h = waitbar(0, 'Computing...');
```

```

viscous_matrix % Generate viscous matrices Bu, Bv and Masku, Maskv
grad_and_div % Generate pressure gradient matrices Pu, Pv and
                % divergence matrices Du, Dv
                % Generate initial u0, v0, p0
%u0 = ones(size(YU))*0.01; % Initial velocity left to right of 0.01
u0 = zeros(size(YU)); % Initial velocity left to right of 0.01
u0(:, :) = 0.01;
v0 = zeros(size(XV));
p0 = (xu(end) - XP)*12/(Re*(yv(end)-yv(1))^2);
u0 = u0'; u0 = u0(:); v0 = v0'; v0 = v0(:); p0 = p0'; p0 = p0(:);

[Lp,Up] = lu([Du Dv]*[Pu;Pv]+adj); % LU decomposition of pressure
correction matrix

nstep = floor(tend/dt); % Number of time steps
relchange = ones(nstep,1); % For plotting relative change per time step
t = 0; n = 0;
right_hand_side % Generate right-hand sides ru, rv; has to be moved
% inside time loop if boundary conditions time-dependent
u1 = u0;
v1 = v0;
n = 1;
indx = 1;

if(makemovie)
    figure(1), clf
    draw_grid;
    hold on;
    title('Magnitude Displacement')
    uq = reshape(u1,size(XU'));
    uqc = filter2([0.5; 0.5],uq);
    uqc = uqc(1:end-1,:);
    vq = reshape(v1,size(XV'));
    vqc = filter2([0.5 0.5],vq);
    vqc = vqc(:,1:end-1);
    quiver(xv,yu,uqc,vqc)
    axis([0 4 -1 2])
    hold off
    m(indx) = getframe;
    indx = indx + 1;
end

for n = 1:nstep % Time-stepping with omega scheme
    t = n*dt;
    u0 = u1;
    v0 = v1;
    %right_hand_side % Generate right-hand sides ru, rv; can be moved
    outside
    % time loop if boundary conditions not time-dependent

    % Navier-Stokes, predictor:
    inertia_matrix % Generate inertia matrices, Cu, Cv, and adapt right-hand
    sides
    rum = (ru + rum)'; rvm = (rv + rvm)'; rum = rum(:); rvm = rvm(:);
    Cu = dt*(Bu + Cu)+adj; Cv = dt*(Bv + Cv)+adj;
    pstar = [p0; zeros(ny,1)];
    u1 = (speye(length(u0)) + omega*Cu)\...
        (Masku*u0 + dt*(rum - Pu*p0) - (1-omega)*Cu*u0 - adj*pstar);
    v1 = (speye(length(v0)) + omega*Cv)\...
        (Maskv*v0 + dt*(rvm - Pv*p0) - (1-omega)*Cv*v0 - adj*pstar);

```

```

% End of Navier-Stokes predictor

dp = Up\ (Lp\ (Du*u1 + Dv*v1 - adj*p0));
p0 = p0 + dp/dt;
u1 = u1 - Pu*dp;
v1 = v1 - Pv*dp;

mm = max([norm(u1,inf),norm(v1,inf)]);
relchange(n) = max([norm(u1-u0,inf)/mm,...
    norm(v1-v0,inf)/mm, norm(dp,inf)/(norm(p0,inf) + 0.5*mm^2)]);
if (relchange(n) < converged)
    break;
end

if(makemovie)
    if (relchange(n) > 1e-4)
        figure(1), clf
        draw_grid;
        hold on;
        title('Magnitude Displacement')
        uq = reshape(u1,size(XU'));
        uqc = filter2([0.5; 0.5],uq);
        uqc = uqc(1:end-1,:);
        vq = reshape(v1,size(XV'));
        vqc = filter2([0.5 0.5],vq);
        vqc = vqc(:,1:end-1);
        quiver(xv,yu,uqc,vqc)
        axis([0 4 -1 2])
        hold off
    end

    % figure(1), clf, hold on
    % title('Horizontal Displacement')
    % uq = reshape(u1,size(XU'));
    % uq = uq'; vq = zeros(size(uq));
    % surf(xu,yu,uq,'Facecolor','Texturemap')
    % colorbar('vert')
    % axis([0 4 -1 2])
    % hold off

    %figure(1), clf, hold on
    %title('Vertical Displacement')
    %vq = reshape(v1,size(XV'));
    %vq = vq'; uq = zeros(size(vq));
    %surf(xv,yv,vq,'Facecolor','Texturemap')
    %colorbar('vert')
    %axis([0 4 -1 2])
    %hold off

    m(indx) = getframe;
    indx = indx + 1;
end

waitbar(n/nstep,h)
end
close(h);

if(makemovie)
    movie(m);
end

```

```

        movie2avi(m, 'displ_magnitude.avi');
end

disp(['Relative change = ', num2str(relchange(end))])
divergence = norm(Du*u1 + Dv*v1, inf);
disp(['max u1 = ', num2str(max(u1))]);
disp(['max v1 = ', num2str(max(v1))]);
disp(['max p0 = ', num2str(max(p0))]);

% Plot of isobars
figure, clf, hold on
title('Pressure')
pp = reshape(p0, size(XP'));
pp = pp';
surf(xv, yu, pp, 'Facecolor', 'Texturemap')
colorbar('vert')
axis([0 4 -1 2])
hold off
print -dpng pressure_surf

figure, clf, hold on
title('Stress')
pp = reshape(p0, size(XP'));
pp = pp';
%cvals = linspace(min(min(pp)), max(max(pp)), 10);
%contour(xv, yu, pp, cvals, 'k')
uq = reshape(u1, size(XU'));
uq = uq';
for k=1:size(uq, 2)-1
    uqc(:, k) = abs(uq(:, k+1) - uq(:, k));
end
vq = reshape(v1, size(XV'));
vq = vq';
for k=1:size(vq, 1)-1
    vqc(k, :) = abs(vq(k+1, :) + vq(k, :));
end
vmstress = sqrt(pp.^2 + 3*G^2*(uqc-vqc).^2);
surf(xv, yu, vmstress, 'Facecolor', 'Texturemap')
colorbar('vert')
axis([0 4 -1 2])
hold off
print -dpng stress_surf
disp(['max vm = ', num2str(max(max(vmstress)))]);

% Velocity profiles
figure, clf, hold on
draw_grid;
hold on;
title('Horizontal Displacement')
uq = reshape(u1, size(XU'));
uq = uq'; vq = zeros(size(uq));
quiver(xu, yu, uq, vq)
axis([0 4 -1 2])
hold off
print -dpng horiz_displ_quiver

figure, clf, hold on
title('Horizontal Displacement')
uq = reshape(u1, size(XU'));
uq = uq'; vq = zeros(size(uq));

```

```

surf(xu,yu,uq,'Facecolor','Texturemap')
colorbar('vert')
axis([0 4 -1 2])
hold off
print -dpng horiz_displ_surf

figure, clf, hold on
draw_grid;
hold on;
title('Vertical Displacement')
vq = reshape(v1,size(XV'));
vq = vq'; uq = zeros(size(vq));
quiver(xv,yv,uq,vq)
axis([0 4 -1 2])
hold off
print -dpng vert_displ_quiver

figure, clf, hold on
title('Vertical Displacement')
vq = reshape(v1,size(XV'));
vq = vq'; uq = zeros(size(vq));
surf(xv,yv,vq,'Facecolor','Texturemap')
colorbar('vert')
axis([0 4 -1 2])
hold off
print -dpng vert_displ_surf

figure, clf, hold on
draw_grid;
hold on;
title('Magnitude Displacement')
uq = reshape(u1,size(XU'));
uqc = filter2([0.5; 0.5],uq);
uqc = uqc(1:end-1,:);
vq = reshape(v1,size(XV'));
vqc = filter2([0.5 0.5],vq);
vqc = vqc(:,1:end-1);
quiver(xv,yv,uqc,vqc)
axis([0 4 -1 2])
hold off
print -dpng mag_displ_quiver

%dispmag = sqrt(uqc.^2 + vqc.^2);

% Plot of relative change per time step
figure, clf
yy = linspace(yv(1),yv(end),30);
semilogy(1:n,relchange(1:n),'k-')
title('Relative change per time step')
print -dpng convergence

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CREATE_GRID
%
% Indexing convention in staggered grid:
% +--k+1--+
% |         |
% j   jk   j+1
% |         |
% +---k---+

```



```

%
dx = ones(1,nx) * (xbound(2)-xbound(1))/nx; % horizontal size of primary (i.e.
pressure) cell
dy = ones(1,ny) * (ybound(2)-ybound(1))/ny; % vertical size of primary (i.e.
pressure) cell
[DX,DY] = meshgrid(dx,dy);          % Sizes of primary cells
J = length(dx);
K = length(dy);

% Create staggered grid
xu = [xbound(1),xbound(1)+cumsum(dx)];          % x-coordinates of u-nodes
xv = 0.5*(xu(1:end-1)+xu(2:end)); % x-coordinates of v-nodes
yv = [ybound(1),ybound(1)+cumsum(dy)];          % y-coordinates of v-nodes
yu = 0.5*(yv(1:end-1)+yv(2:end)); % y-coordinates of u-nodes
[XU,YU] = meshgrid(xu,yu); % mesh of u-node coordinates
[XV,YV] = meshgrid(xv,yv); % mesh of v-node coordinates
[XP,YP] = meshgrid(xv,yu); % Coordinates of pressure nodes at centers of
finite volumes

DXU = XU;          % Size of finite volumes for u; preallocation.
DXU(:,1) = DX(:,1)/2;
DXU(:,2:J) = (DX(:,1:J-1) + DX(:,2:J))/2;
DXU(:,J+1) = DX(:,J)/2;
DYU = [DY,dy'];
DYV = XV;          % Size of finite volumes for v; preallocation.
DYV(1,:) = DY(1,+)/2;
DYV(2:K,:) = (DY(1:K-1,+) + DY(2:K,+))/2;
DYV(K+1,:) = DY(K,+)/2;
DXV = [DX;dx];
volu = DXU.*DYU;
vol = 1./volu';
n = (J+1)*K;
invvolu = spdiags(vol(:), 0, n, n);          % (u-volume)^(-1)
volv = DXV.*DYV;
vol = 1./volv';
n = J*(K+1);
invvolv = spdiags(vol(:), 0, n, n);          % (v-volume)^(-1)

hx = min(dx);
hy = min(dy);

clear vol

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%VISCOUS_MATRIX
% Vectorized matrix generation by evaluation of stencil coefficients
% Output: Bu, Bv
%
%      |   b5   |
% [B] = |b2 b3 b4|
%      |   b1   |
%
% Indexing convention in staggered grid:
% +--k+1--+
% |         |
% j   jk   j+1
% |         |
% +---k----+

global Re

```

```

r = 1/Re;

% Mask for old u, v in Dirichlet boundary points in time-stepping:
masku = ones(size(XU)); maskv = ones(size(XV));

%.....Diagonals of viscous matrix Bu for u.....
au1 = zeros(K,J+1); au2=au1; au3=au1; au4=au1; au5=au1;          % Diagonals of
Bu
au1(2:K,:) = -2*r.*DXU(2:K,:)./(DYU(1:K-1,:) + DYU(2:K,:));
au2(:,2:J+1) = -r*DY./DX;
au4(:,1:J) = au2(:,2:J+1);          au5(1:K-1,:) = au1(2:K,:);
au3          = - au1 - au2 - au4 - au5;

%.....Diagonals of viscous matrix Bv for v.....
av1 = zeros(K+1,J); av2=av1; av3=av1; av4=av1; av5=av1;          % Diagonals of
Bv
av1(2:K+1,:) = -r*DX./DY;
av2(:,2:J) = -2*r*DYV(:,2:J)./(DXV(:,1:J-1)+DXV(:,2:J));
av4(:,1:J-1) = av2(:,2:J); av5(1:K,:) = av1(2:K+1,:);
av3          = - av1 - av2 - av4 - av5;

% .....Boundary corrections.....
jseg = [0,cumsum(nx)];          % j-indices of horizontal segment boundaries
for seg = 1:length(xbc(1,:))
    if (xbc(1,seg) == 1)|(xbc(1,seg) == 2) % No-slip or inflow at lower boundary
        j = 1+jseg(seg);          au3(1,j) = au3(1,j) + r*DX(1,j)/DY(1,j);
        j = 2+jseg(seg):jseg(seg+1); au3(1,j) = au3(1,j) + 2*r*DXU(1,j)./DYU(1,j);
        j = 1+jseg(seg+1);          au3(1,j) = au3(1,j) + r*DX(1,j-1)/DY(1,j-1);

    elseif (xbc(1,seg) == 0)|(xbc(1,seg) == 3)          % Outflow at lower
boundary
        % Do nothing
    else
        error('Wrong xbc')
    end
    if (xbc(2,seg) == 1)|(xbc(2,seg) == 2) % No-slip or inflow at upper boundary
        j = 1+jseg(seg);          au3(K,j) = au3(K,j) + r*DX(K,j)/DY(K,j);
        j = 2+jseg(seg):jseg(seg+1); au3(K,j) = au3(K,j) + 2*r*DXU(K,j)./DYU(K,j);
        j = 1+jseg(seg+1);          au3(K,j) = au3(K,j) + r*DX(K,j-1)/DY(K,j-1);

    elseif (xbc(2,seg) == 0)|(xbc(2,seg) == 3)          % Outflow at upper
boundary
        % Do nothing
    else
        error('Wrong xbc')
    end
end
kseg = [0,cumsum(ny)];          % k-indices of vertical segment boundaries
for seg = 1:length(ybc(1,:))
    if (ybc(1,seg) == 1)|(ybc(1,seg) == 2) % No-slip or inflow at left boundary
        k = 1+kseg(seg):kseg(seg+1);
        au3(k,1) = 0; masku(k,1) = 0;
        au1(k,1) = 0; au2(k,1) = 0; au4(k,1) = 0; au5(k,1) = 0;
        k = 1+kseg(seg);          av3(k,1) = av3(k,1) + r*DY(k,1)/DX(k,1);
        k = 2+kseg(seg):kseg(seg+1); av3(k,1) = av3(k,1) + 2*r*DYV(k,1)./DXV(k,1);
        k = 1+kseg(seg+1);          av3(k,1) = av3(k,1) + r*DY(k-1,1)/DX(k-1,1);

    elseif ybc(1,seg) == 3          % Outflow at left boundary
        % Do nothing
    else

```

```

    error('Wrong ybc')
end

if (ybc(2,seg) == 1)|(ybc(2,seg) == 2) % No-slip or inflow at right boundary
    k = 1+kseg(seg):kseg(seg+1);
    au3(k,J+1) = 0; masku(k,J+1) = 0;
    au1(k,J+1) = 0; au2(k,J+1) = 0; au4(k,J+1) = 0; au5(k,J+1) = 0;
    k = 1+kseg(seg); av3(k,J) = av3(k,J) + r*DY(k,J)/DX(k,J);
    k = 2+kseg(seg):kseg(seg+1); av3(k,J) = av3(k,J) + 2*r*DYV(k,J)./DXV(k,J);
    k = 1+kseg(seg+1); av3(k,J) = av3(k,J) + r*DY(k-1,J)/DX(k-1,J);

elseif ybc(2,seg) == 3 % Outflow at right boundary
    % Do nothing
else
    error('Wrong ybc')
end
end
for seg = 1:length(xbc(1,:))
    if (xbc(1,seg) == 1)|(xbc(1,seg) == 2) % No-slip or inflow at lower boundary
        j = 1+jseg(seg):jseg(seg+1);
        av3(1,j) = 0; maskv(1,j) = 0;
        av1(1,j) = 0; av2(1,j) = 0; av4(1,j) = 0; av5(1,j) = 0;

        elseif (xbc(1,seg) == 0)|(xbc(1,seg) == 3) % Outflow at lower
boundary
            % Do nothing
        else
            error('Wrong xbc')
        end
        if (xbc(2,seg) == 1)|(xbc(2,seg) == 2) % No-slip or inflow at upper boundary
            j = 1+jseg(seg):jseg(seg+1);
            av3(K+1,j) = 0; maskv(K+1,j) = 0;
            av1(K+1,j) = 0; av2(K+1,j) = 0; av4(K+1,j) = 0; av5(K+1,j) = 0;

            elseif (xbc(2,seg) == 0)|(xbc(2,seg) == 3) % Outflow at upper
boundary
                % Do nothing
            else
                error('Wrong xbc')
            end
        end
    end

    au1 = au1'; au1 = au1(:); au2 = au2'; au2 = au2(:);
    au3 = au3'; au3 = au3(:); au4 = au4'; au4 = au4(:);
    au5 = au5'; au5 = au5(:);
    %[au1 au2 au3 au4 au5] % Display stencil on screen
    n = (J+1)*K;
    au1 = [au1(J+2:n); zeros(J+1,1)]; % Shifts to accomodate spdiags
    au2 = [au2(2:n); 0]; au4 = [0; au4(1:n-1)]; au5 = [zeros(J+1,1); au5(1:n-J-
1)];
    d = [-J-1; -1; 0; 1; J+1];
    Bu = invvolu*spdiags([au1 au2 au3 au4 au5], d, n, n);

    av1 = av1'; av1 = av1(:); av2 = av2'; av2 = av2(:);
    av3 = av3'; av3 = av3(:); av4 = av4'; av4 = av4(:);
    av5 = av5'; av5 = av5(:);
    %[av1 av2 av3 av4 av5] % Display stencil on screen
    n = J*(K+1);
    av1 = [av1(J+1:n); zeros(J,1)]; % Shifts to accomodate spdiags

```

```

av2 = [av2(2:n); 0];          av4 = [0; av4(1:n-1)];      av5 = [zeros(J,1)];
av5(1:n-J)];
d = [-J; -1; 0; 1; J];
Bv = invvoly*spdiags([av1 av2 av3 av4 av5], d, n, n);
masku = masku'; masku = masku(:);
Masku = spdiags(masku,0,length(masku),length(masku));
maskv = maskv'; maskv = maskv(:);
Maskv = spdiags(maskv,0,length(maskv),length(maskv));

clear au1 au2 au3 au4 au5 av1 av2 av3 av4 av5 d masku maskv

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%GRAD_AND_DIV
% Generates pressure gradient matrices Pu, Pv and divergence matrix D
% Vectorized matrix generation by evaluation of stencil coefficients
% Output: Pu, Pv, Du, Dv

% Indexing convention in staggered grid:
% +--k+1--+
% |         |
% j   jk   j+1
% |         |
% +---k---+

J= length(dx); K = length(dy);

%.....Pressure gradient matrix Pu .....

%
% [Pu] = | 0 |
%         | p1 p2 0 |
%         | 0 |

p2 = 1./DXU(1,:); p1 = zeros(size(p2)); p1(1:J) = -p2(2:J+1);
Puu = spdiags([p1 p2],[-1;0], J+1, J);
Pu = kron(speye(K),Puu);

%.....Boundary corrections.....
for seg = 1:length(ybc(1,:))
    if (ybc(1,seg) == 1)|(ybc(1,seg) == 2) % No-slip or inflow at left boundary
        k = 1+kseg(seg):kseg(seg+1); Pu(1+(k-1)*(J+1),:) = 0;
    end
    if (ybc(2,seg) == 1)|(ybc(2,seg) == 2) % No-slip or inflow at right boundary
        k = 1+kseg(seg):kseg(seg+1); Pu(k*(J+1),:) = 0;
    end
end

%.....Pressure gradient matrix Pv .....

%
% [Pv] = | 0 |
%         | 0 p2 0 |
%         | p1 |

pp1 = zeros(size(XV)); pp2 = pp1; % Diagonals of Pv
pp1(2:K+1,:) = -1./DYV(2:K+1,:); pp2 = -pp1;
pp2(1,:) = 1./DYV(1,:); pp2(K+1,:) = 0;

%.....Boundary corrections.....
for seg = 1:length(xbc(1,:))
    if (xbc(1,seg) == 1)|(xbc(1,seg) == 2) % No-slip or inflow at lower boundary
        j = 1+jseg(seg):jseg(seg+1); pp1(1,j) = 0; pp2(1,j) = 0;
    end
end

```

```

end
if (xbc(2,seg) == 1)|(xbc(2,seg) == 2) % No-slip or inflow at upper boundary
    j = 1+jseg(seg):jseg(seg+1); pp1(K+1,j) = 0; pp2(K+1,j) = 0;
end
end
n = J*(K+1); p1 = reshape(pp1',n,1); p2 = reshape(pp2',n,1);
p1 = [p1(J+1:n); zeros(J,1)]; % Shift to accomodate spdiags
Pv = spdiags([p1 p2],[-J;0], n,n-J);

%.....u divergence matrix Du .....

%
% [Du] =  $\begin{vmatrix} & 0 & \\ 0 & p1 & p2 \\ & 0 & \end{vmatrix}$ 
%

Duu = spdiags([-ones(J,1) ones(J,1)], [0;1], J,J+1);
Du = kron(spdiags(dy',0,K,K),Duu);

%.....v divergence matrix Dv .....

%
% [Dv] =  $\begin{vmatrix} & p2 & \\ 0 & p1 & 0 \\ & 0 & \end{vmatrix}$ 
%

Dvv = spdiags([-ones(K,1) ones(K,1)], [0;1], K,K+1);
Dv = kron(Dvv,spdiags(dx',0,J,J));

clear pp1 pp2 p1 p2 Puu Duu Dv % Save storage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%RIGHT_HAND_SIDE
% Generates contribution of boundary conditions to right-hand side ru, rv for
% momentum equations

% Functions called: ubd, vbd, pbd

% Indexing convention in staggered grid:
% +--k+1--+
% | |
% j jk j+1
% | |
% +---k---+

tm = t + (omega-1)*dt;
ru = zeros(size(XU)); rv = zeros(size(XV));
for seg = 1:length(xbc(1,:))
    if xbc(1,seg) == 1 % No-slip at lower boundary
        % Do nothing
    elseif xbc(1,seg) == 2 % Inflow at lower boundary
        j = 1+jseg(seg);
        ru(1,j) = ru(1,j) +...
            ubd(tm,j,1,'lower')*r*DX(1,j)/(DY(1,j)*volu(1,j));
        for j = 2+jseg(seg):jseg(seg+1)
            ru(1,j) = ru(1,j) +...
                ubd(tm,j,1,'lower')*2*r*DXU(1,j)/(DYU(1,j).*volu(1,j));
        end
        j = 1+jseg(seg+1);
        ru(1,j) = ru(1,j) +...
            ubd(tm,j,1,'lower')*r*DX(1,j-1)/(DY(1,j-1)*volu(1,j));
    end
end

```

```

elseif (xbc(1,seg) == 0)|(xbc(1,seg) == 3)           % Outflow at lower
boundary
    % Do nothing
else
    error('Wrong value for xbc in right_hand_side.m, line a')
end
if xbc(2,seg) == 1                                 % No-slip at upper boundary
    % Do nothing
elseif xbc(2,seg) == 2                            % Inflow at upper boundary
    j = 1+jseg(seg);
    ru(K,j) = ru(K,j) +...
        ubd(tm,j,K+1,'upper')*r*DX(K,j)/(DY(K,j)*volu(K,j));
    for j = 2+jseg(seg):jseg(seg+1);
        ru(K,j) = ru(K,j) +...
            ubd(tm,j,K+1,'upper')*2*r*DXU(K,j)/(DYU(K,j).*volu(K,j));
    end
    j = 1+jseg(seg+1);
    ru(K,j) = ru(K,j) +...
        ubd(tm,j,K+1,'upper')*r*DX(K,j-1)/(DY(K,j-1)*volu(K,j));
elseif (xbc(2,seg) == 0)|(xbc(2,seg) == 3)       % Outflow at upper
boundary
    % Do nothing
else
    error('Wrong value for xbc in right_hand_side.m, line b')
end
end

for seg = 1:length(ybc(1,:))
if ybc(1,seg) == 1                                 % No-slip at left boundary
    k = 1+kseg(seg):kseg(seg+1);
    ru(k,1) = 0;
elseif ybc(1,seg) == 2                            % Inflow at left boundary
    for k = 1+kseg(seg):kseg(seg+1)
        ru(k,1) = ubd(tm,1,k,'left')/dt;
    end
    k = 1+kseg(seg);
    rv(k,1) = rv(k,1) +...
        vbd(tm,1,k,'left')*r*DY(k,1)/(DX(k,1)*volv(k,1));
    for k = 2+kseg(seg):kseg(seg+1)
        rv(k,1) = rv(k,1) +...
            vbd(tm,1,k,'left')*2*r*DYV(k,1)/(DXV(k,1).*volv(k,1));
    end
    k = 1+kseg(seg+1);
    rv(k,1) = rv(k,1) +...
        vbd(tm,1,k,'left')*r*DY(k-1,1)/(DX(k-1,1)*volv(k,1));
elseif ybc(1,seg) == 3                            % Outflow at left boundary
    for k = 1+kseg(seg):kseg(seg+1)
        ru(k,1) = ru(k,1) + pbd(tm,xu(1),yu(k),'left')*2./DX(k,1);
    end
else
    error('Wrong value for ybc in right_hand_side.m, line c')
end

if ybc(2,seg) == 1                                 % No-slip at right boundary
    for k = 1+kseg(seg):kseg(seg+1)
        ru(k,J+1) = 0;
    end
elseif ybc(2,seg) == 2                            % Inflow at right boundary
    for k = 1+kseg(seg):kseg(seg+1)
        ru(k,J+1) = ubd(tm,J+1,k,'right')/dt;

```

```

end
k = 1+kseg(seg);
rv(k,J) = rv(k,J) +...
    vbd(tm,J+1,k,'right')*r*DY(k,J)/(DX(k,J)*volv(k,J));
for k = 2+kseg(seg):kseg(seg+1)
    rv(k,J) = rv(k,J) +...
        vbd(tm,J+1,k,'right')*2*r*DYV(k,J)/(DXV(k,J).*volv(k,J));
end
k = 1+kseg(seg+1);
rv(k,J) = rv(k,J) +...
    vbd(tm,J+1,k,'right')*r*DY(k-1,J)/(DX(k-1,J)*volv(k,J));
elseif ybc(2,seg) == 3 % Outflow at right boundary
    for k = 1+kseg(seg):kseg(seg+1)
        ru(k,J+1) = ru(k,J+1) - pbd(tm,xu(J+1),yu(k),'right')*2./DX(k,J);
    end
else
    error('Wrong value for ybc in right_hand_side.m, line d')
end
end

for seg = 1:length(xbc(1,:))
    if xbc(1,seg) == 1 % No-slip at lower boundary
        j = 1+jseg(seg):jseg(seg+1);
        rv(1,j) = 0;
    elseif xbc(1,seg) == 2 % Inflow at lower boundary
        for j = 1+jseg(seg):jseg(seg+1)
            rv(1,j) = vbd(tm,j,1,'lower')'/dt;
        end
    elseif (xbc(1,seg) == 0)|(xbc(1,seg) == 3) % Outflow at lower
boundary
        for j = 1+jseg(seg):jseg(seg+1)
            rv(1,j) = rv(1,j) + pbd(tm,xv(j),yv(1),'lower')*2./DY(1,j);
        end
    else
        error('Wrong xbc')
    end
    if xbc(2,seg) == 1 % No-slip at upper boundary
        for j = 1+jseg(seg):jseg(seg+1)
            rv(K+1,j) = 0;
        end
    elseif xbc(2,seg) == 2 % Inflow at upper boundary
        for j = 1+jseg(seg):jseg(seg+1)
            rv(K+1,j) = vbd(tm,j,K+1,'upper')/dt;
        end
    elseif (xbc(2,seg) == 0)|(xbc(2,seg) == 3) % Outflow at upper
boundary
        for j = 1+jseg(seg):jseg(seg+1)
            rv(K+1,j) = rv(K+1,j) - pbd(tm,xv(j),yv(K+1),'upper')*2./DY(K,j);
        end
    else
        error('Wrong xbc')
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%INERTIA_MATRIX
% Discretization matrices for inertia term and modification of right-hand sides
% Output: Cu, Cv
%
% | c5 |
% [C] = |c2 c3 c4|

```

```

%          |   c1   |

% Indexing convention in staggered grid:
%  +--k+1--+
%  |         |
%  j   jk   j+1
%  |         |
%  +---k---+

global adj;

au1 = zeros(K,J+1); au2=au1; au3=au1; au4=au1; au5=au1;    % Diagonals of Cu
av1 = zeros(K+1,J); av2=av1; av3=av1; av4=av1; av5=av1;    % Diagonals of Cv

% Central scheme for inertia term
up = reshape(u0,size(XU')); up = up';          % Two-index ordering of u0
vp = reshape(v0,size(XV')); vp = vp';          % and v0
upv = [up;up(K,:)];          % upv: old u in cell vertices
upv(2:K,:) = (upv(1:K-1,:).*DYU(1:K-1,:) + upv(2:K,:).*DYU(2:K,...
    ./ (DYU(1:K-1,:) + DYU(2:K,:)));
vpv = [vp;vp(:,J)];          % vpv: old v in cell vertices
vpv(:,2:J) = (vpv(:,1:J-1).*DXV(:,1:J-1) + vpv(:,2:J).*DXV(:,2:J)...
    ./ (DXV(:,1:J-1) + DXV(:,2:J)));

au1(2:K,:) = - vpv(2:K,:)./(2*DYU(2:K,:));
au2(:,2:J+1) = - up(:,1:J)./(2*DXU(:,2:J+1));
au4(:,1:J) = up(:,2:J+1)./(2*DXU(:,1:J));
au5(1:K-1,:) = vpv(2:K,:)./(2*DYU(1:K-1,:));
au3(:,1) = - up(:,1)./(2*DXU(:,1));
au3(:,J+1) = up(:,J+1)./(2*DXU(:,J+1));
au3(1:K-1,:) = au3(1:K-1,:) + au5(1:K-1,:);
au3(2:K,:) = au3(2:K,:) + au1(2:K,:);
% Further contributions to au3(1,:) and au3(K,:) to be added below depending on
% boundary conditions

av1(2:K+1,:) = - vp(1:K,:)./(2*DYV(2:K+1,:));
av2(:,2:J) = - upv(:,2:J)./(2*DXV(:,2:J));
av4(:,1:J-1) = upv(:,2:J)./(2*DXV(:,1:J-1));
av5(1:K,:) = vp(2:K+1,:)./(2*DYV(1:K,:));
av3(1,:) = - vp(1,:)./(2*DYV(1,:));
av3(K+1,:) = vp(K+1,:)./(2*DYV(K+1,:));
av3(:,2:J) = av3(:,2:J) + av2(:,2:J);
av3(:,1:J-1) = av3(:,1:J-1) + av4(:,1:J-1);
% Further contributions to av3(:,1) and av3(:,J) to be added below depending on
% boundary conditions.

% .....Boundary corrections.....

rvm = zeros(size(XV));          % Contribution to right-hand side
tm = t + (omega-1)*dt;
for seg = 1:length(ybc(1,:))
    if (ybc(1,seg) == 1) | (ybc(1,seg) == 2) % No-slip or inflow at left
        boundary
            k = 1+kseg(seg);
            rvm(k,1) = rvm(k,1) + 0.5*up(k,1).*vbd(tm,1,k,'left')*DY(k,1);
            k = 2+kseg(seg):kseg(seg+1);
            rvm(k,1) = rvm(k,1) + upv(k,1).*vbd(tm,1,k,'left')*.DYV(k,1);
            k = kseg(seg+1)+1;
            rvm(k,1) = rvm(k,1) + 0.5*up(k-1,1).*vbd(tm,1,k,'left')*DY(k-1,1);
    end
end

```



```

elseif ybc(1,seg) == 3          % Outflow at left boundary
    k = 1+kseg(seg);
    av3(k,1) = av3(k,1) - 0.5*up(k,1)*DY(k,1)/(DYV(k,1)*DXV(k,1));
    k = 2+kseg(seg):kseg(seg+1);
    av3(k,1) = av3(k,1) - upv(k,1)./DXV(k,1);
    k = kseg(seg+1)+1;
    av3(k,1) = av3(k,1) - 0.5*up(k-1,1)*DY(k-1,1)/(DYV(k,1)*DXV(k,1));
else
    error('Wrong ybc')
end

if (ybc(2,seg) == 1)|(ybc(2,seg) == 2) % No-slip or inflow at right
boundary
    k = 1+kseg(seg);
    rvm(k,J) = rvm(k,J) - 0.5*up(k,J+1).*vbd(tm,J,k,'right')*DY(k,J);
    k = 2+kseg(seg):kseg(seg+1);
    rvm(k,J) = rvm(k,J) - upv(k,J+1).*vbd(tm,J,k,'right').*DYV(k,J);
    k = kseg(seg+1)+1;
    rvm(k,J) = rvm(k,J) - 0.5*up(k-1,J+1).*vbd(tm,J,k,'right')*DY(k-1,J);

elseif ybc(2,seg) == 3          % Outflow at right boundary
    k = 1+kseg(seg);
    av3(k,J) = av3(k,J) + 0.5*up(k,J+1)*DY(k,J)/(DYV(k,J)*DXV(k,J));
    k = 2+kseg(seg):kseg(seg+1);
    av3(k,J) = av3(k,J) + upv(k,J+1)./DXV(k,J);
    k = kseg(seg+1)+1;
    av3(k,J) = av3(k,J) + 0.5*up(k-1,J+1)*DY(k-1,J)/(DYV(k,J)*DXV(k,J));
else
    error('Wrong ybc')
end
end

rum = zeros(size(XU));          % Contribution to right-hand side
for seg = 1:length(xbc(1,:))
    if (xbc(1,seg) == 1)|(xbc(1,seg) == 2) % No-slip or inflow at lower
boundary
        j = 1+jseg(seg):jseg(seg+1); av3(1,j) = 0;
        av1(1,j) = 0; av2(1,j) = 0; av4(1,j) = 0; av5(1,j) = 0;
        j = 1+jseg(seg);
        rum(1,j) = rum(1,j) + 0.5*vp(1,j).*ubd(tm,j,1,'lower')*DX(1,j);
        j = 2+jseg(seg):jseg(seg+1);
        rum(1,j) = rum(1,j) + vpv(1,j).*ubd(tm,j,1,'lower').*DXU(1,j);
        j = jseg(seg+1)+1;
        rum(1,j) = rum(1,j) + 0.5*vp(1,j-1).*ubd(tm,j,1,'lower')*DX(1,j-1);
    elseif (xbc(1,seg) == 0)|(xbc(1,seg) == 3) % Outflow at lower
boundary
        j = 1+jseg(seg);
        au3(1,j) = au3(1,j) - 0.5*vp(1,j)*DX(1,j)/(DXU(1,j)*DYU(1,j));
        j = 2+jseg(seg):jseg(seg+1);
        au3(1,j) = au3(1,j) - vpv(1,j)./DYU(1,j);
        j = jseg(seg+1)+1;
        au3(1,j) = au3(1,j) - 0.5*vp(1,j-1)*DX(1,j-1)/(DXU(1,j)*DYU(1,j));
    else
        error('Wrong xbc')
    end
end
if (xbc(2,seg) == 1)|(xbc(2,seg) == 2) %No-slip or inflow at upper
boundary
    j = 1+jseg(seg):jseg(seg+1);
    av3(K+1,j) = 0;
    av1(K+1,j) = 0; av2(K+1,j) = 0; av4(K+1,j) = 0; av5(K+1,j) = 0;

```

```

        j = 1+jseg(seg);
        rum(K,j) = rum(K,j) - 0.5*vp(K+1,j).*ubd(tm,j,K,'upper')*DX(K,j);
        j = 2+jseg(seg):jseg(seg+1);
        rum(K,j) = rum(K,j) - vpv(K+1,j).*ubd(tm,j,K,'upper')*.DXU(K,j);
        j = jseg(seg+1)+1;
        rum(K,j) = rum(K,j) - 0.5*vp(K+1,j-1).*ubd(tm,j,K,'upper')*DX(K,j-1);
    elseif (xbc(2,seg) == 0) | (xbc(2,seg) == 3)           % Outflow at upper
boundary
        j = 1+jseg(seg);
        au3(K,j) = au3(K,j) + 0.5*vp(K+1,j)*DX(K,j)/(DXU(K,j)*DYU(K,j));
        j = 2+jseg(seg):jseg(seg+1);
        au3(K,j) = au3(K,j) + vpv(K+1,j)./DYU(K,j);
        j = jseg(seg+1)+1;
        au3(K,j) = au3(K,j) + 0.5*vp(K+1,j-1)*DX(K,j-1)/(DXU(K,j)*DYU(K,j));
    else
        error('Wrong xbc')
    end
end
end

for seg = 1:length(ybc(1,:))
    if (ybc(1,seg) == 1) | (ybc(1,seg) == 2) %No-slip or inflow at left boundary
        k = 1+kseg(seg):kseg(seg+1); rum(k,1) = 0; au3(k,1) = 0;
        au1(k,1) = 0; au2(k,1) = 0; au4(k,1) = 0; au5(k,1) = 0;
    end
    if (ybc(2,seg) == 1) | (ybc(2,seg) == 2) %No-slip or inflow at right
boundary
        k = 1+kseg(seg):kseg(seg+1); rum(k,J+1) = 0; au3(k,J+1) = 0;
        au1(k,J+1) = 0; au2(k,J+1) = 0; au4(k,J+1) = 0; au5(k,J+1) = 0;
    end
end
for seg = 1:length(xbc(1,:))
    if (xbc(1,seg) == 1) | (xbc(1,seg) == 2) %No-slip or inflow at lower
boundary
        j = 1+jseg(seg):jseg(seg+1); rvm(1,j) = 0;
    end
    if (xbc(2,seg) == 1) | (xbc(2,seg) == 2) %No-slip or inflow at upper
boundary
        j = 1+jseg(seg):jseg(seg+1); rvm(K,j) = 0;
    end
end
end

rum = rum./volu; rvm = rvm./volv; % Contributions to right-hand side

au1 = au1'; au1 = au1(:);
au2 = au2'; au2 = au2(:);
au3 = au3'; au3 = au3(:);
au4 = au4'; au4 = au4(:);
au5 = au5'; au5 = au5(:);
%[au1 au2 au3 au4 au5]           % Display stencil on screen
nn = (J+1)*K;
au1 = [au1(J+2:nn); zeros(J+1,1)]; % Shifts to accomodate spdiags
au2 = [au2(2:nn); 0]; au4 = [0; au4(1:nn-1)]; au5 = [zeros(J+1,1); au5(1:nn-J-
1)];
d = [-J-1; -1; 0; 1; J+1];
Cu = spdiags([au1 au2 au3 au4 au5], d, nn, nn);

av1 = av1'; av1 = av1(:);
av2 = av2'; av2 = av2(:);
av3 = av3'; av3 = av3(:);

```

```

av4 = av4'; av4 = av4(:);
av5 = av5'; av5 = av5(:);
%[av1 av2 av3 av4 av5] % Display stencil on screen
nn = J*(K+1);
av1 = [av1(J+1:nn); zeros(J,1)]; % Shifts to accomodate spdiags
av2 = [av2(2:nn); 0]; av4 = [0; av4(1:nn-1)]; av5 = [zeros(J,1);
av5(1:nn-J)];
d = [-J; -1; 0; 1; J];
Cv = spdiags([av1 av2 av3 av4 av5], d, nn, nn);

%clear au1 au2 au3 au4 au5 av1 av2 av3 av4 av5 d nn

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ye = pbd(t, x, y, side)
%PBD Prescribes pressure p at outflow boundaries
% Possible values for side: 'lower', 'upper', 'left', 'right'

global geval

ye = 0;
if (geval == 1|geval == 7) % Horizontal Poiseuille flow
    if strcmp(side, 'right') == 1 % Outflow
        ye = 0;
    else
        ye = 0;
    end
elseif (geval == 2) % Vertical Poiseuille flow
    if strcmp(side, 'upper') == 1 % Outflow
        ye = 0;
    else
        ye = 0;
    end
elseif (geval == 3) % Backward facing step
    ye = 0;
elseif (geval == 4) % Driven cavity
    ye = 0;
elseif (geval == 5|geval == 6) % Uniform flow under angle alpha
    ye = 1;
else
    error('Wrong value in input for parameter geval')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ye = ubd(t, j, k, side)
%UBD Prescribes u at inflow boundaries
% Possible values for side: 'lower', 'upper', 'left', 'right'

global geval u0 J K yu yv yseglen alpha

ye = 0;
if (geval == 1) % Horizontal Poiseuille flow to the right
    if strcmp(side, 'left') == 1
        ye = u0(k*(J+1)); % Inflow profile = outflow profile
    else
        ye = 0;
    end
elseif (geval == 2) % Vertical Poiseuille flow
    ye = 0;
elseif (geval == 3) % Backward facing step

```

```

    if strcmp(side, 'left') == 1
        ye = 6*(yu(k) - yseglen(end)).*(yv(end) - yu(k))/(yv(end) -
yseglen(end))^3;
    elseif strcmp(side, 'right') == 1
        ye = 6*(yu(k) - yv(1)).*(yv(end) - yu(k))/(yv(end) - yv(1))^3;
    else
        ye = 0;
    end
elseif (geval == 4) % Driven cavity
    if strcmp(side, 'upper') == 1
        ye = 1;
    else
        ye = 0;
    end
elseif (geval == 5|geval == 6) % Uniform flow under angle alpha
    ye = cos(alpha);
elseif (geval == 7) % Horizontal Poiseuille flow to the left
    if strcmp(side, 'right') == 1
        ye = u0(k*(J+1)); % Inflow profile = outflow profile
    else
        ye = 0;
    end
else
    error('Wrong value in input for parameter geval')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ye = vbd(t, j, k, side)
%VBD Prescribes v at inflow boundaries
% Possible values for side: 'lower', 'upper', 'left', 'right'

global geval v0 J K alpha

ye = 0;
if (geval == 1|geval == 7) % Horizontal Poiseuille flow
    ye = 0;
elseif (geval == 2) % Vertical Poiseuille flow
    if strcmp(side, 'lower') == 1
        ye = v0(j+K*J); % Inflow profile = outflow profile
    else
        ye = 0;
    end
elseif (geval == 3) % Backward facing step
    ye = 0;
elseif (geval == 4) % Driven cavity
    ye = 0;
elseif (geval == 5|geval == 6) % Uniform flow under angle alpha
    ye = sin(alpha);
else
    error('Wrong value in input for parameter geval')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% draw_grid
[X,Y] = meshgrid(xu,yv); % Volume boundaries
figure, clf, hold on
title('Grid')
plot(X,Y,':k') % Vertical grid lines
plot(X',Y',':k') % Horizontal grid lines
%[X,Y] = meshgrid(xv,yu); % Nodes

```

```
%plot(X,Y,'k.','MarkerSize',3) % Vertical points
%plot(X',Y','k.','MarkerSize',3) % Horizontal points
axis([0 4 -1 2])
hold off
```