1992

An implicit numerical scheme for the simulation of internal viscous flows on unstructured grids

Philip Charles Eberhardt Jorgenson
Iowa State University

An implicit numerical scheme for the simulation of internal viscous flows on unstructured grids

Jorgenson, Philip Charles Eberhardt, Ph.D.

Iowa State University, 1992

An implicit numerical scheme for the simulation of internal viscous flows on unstructured grids

by

Philip Charles Eberhardt Jorgenson

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Mechanical Engineering
Major: Mechanical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1992

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my major professor, Dr. Richard H. Pletcher, for his guidance and encouragement during my years as a graduate student.

My appreciation is extended as well to the members of my graduate committee, Dr. Bruce R. Munson, Dr. Theodore H. Okiishi, Dr. Joseph M. Prusa, and Dr. John C. Tannehill. A special thanks goes to Dr. Richard G. Hindman for acting as a substitute during my final oral examination.

My sincere gratitude goes out to the Internal Fluid Mechanics Division at NASA Lewis Research Center for providing the time and the money to attend school and complete the research for my dissertation.

My parents, Norman and Gloria, and brother, Roy, deserve a special thanks for their undivided attention and moral support.

NOMENCLATURE

Roman Symbols

$\vec{b}$	vector of residuals on right hand side of matrix equation
e	internal energy per unit mass
i	cell index
qx	heat conduction in x direction
qy	heat conduction in y direction
s	semiperimeter of a triangle
t	physical time
u	velocity component in the x direction
v	velocity component in the y direction
w	vector of primitive variables
x	Cartesian coordinate aligned with the horizontal
$\vec{x}$	vector of unknowns in matrix equation
y	Cartesian coordinate aligned with the vertical
$\mathbf{A}$	sparse matrix of coefficients
A_i	area of cell i
$\mathbf{A}_p$	preconditioning Jacobian matrix

$\mathbf{A}_t$	temporal Jacobian matrix
$\mathbf{A}_x$	spatial Jacobian matrix
C	speed of sound
C_1	Sutherland law constant
C_2	Sutherland law constant
Cp	specific heat at constant pressure
E	Cartesian flux vector in x direction
F	Cartesian flux vector in y direction
G	Cartesian flux vector written in primitive variables in x direction
H	Cartesian flux vector written in primitive variables in y direction
L	characteristic length
M	Mach number
P	pressure
Pr	Prandtl number
Q	vector of conserved quantities written in primitive variables
R	gas constant
Re	Reynolds number
T	temperature
U	vector of conserved quantities

Greek Symbols

γ	ratio of specific heats

κ	thermal conductivity coefficient
μ	dynamic viscosity
ρ	density
τ	pseudo time
τ_{xx}	shearing stress
τ_{xy}	shearing stress
τ_{yy}	shearing stress

Subscripts

d	cylinder diameter
h	inlet channel height
v	viscous flux
x	derivative or quantity in x direction
y	derivative or quantity in y direction
ref	reference quantity

Superscripts

$\hat{}$	provisional quantities
$\tilde{}$	nondimensional quantities

1. INTRODUCTION

The development of a general computer code that can predict the flow about complex geometries which include complex flow structures is desirable in computational fluid dynamics. Many numerical schemes proposed to date which use the finite difference or finite volume formulation of the flow equations were written to take advantage of some inherent grid structure which then permits flow solutions to be obtained efficiently [1] - [3]. A structured mesh can be defined as a domain that is discretized such that the neighborhood of a cell or node can be related to its own index number. This structure, which makes the solver so efficient, often makes it difficult to obtain reasonable grids about complex flow geometries. Many of these solution algorithms can be found discussed in review papers [4], [5]. Several structured grids can be used to break up a complex domain into more manageable subdomains. Here the difficulty becomes one of obtaining the necessary flow and geometric information where the grids intersect each other. Grid patching or grid overlaying techniques have been used with structured grids [6], [7]; however, the use of these techniques usually requires special coding in the flow solver to circumvent problems like metric discontinuities and function interpolations between the various grids. An unstructured grid flow solver can alleviate many of the problems associated with structured grids. However, unlike the structured grid, the cell neighborhood of an unstructured grid must be defined explicitly. This information is usually provided to the flow code in the form of a connectivity matrix. The triangle is the simplest and most convenient geometric figure that can be used to cover a two-dimensional domain. An advantage of using a simple triangular shaped cell is the ability to generate grids about arbitrary geometries. Another advantage is the ability to add cells in high gradient regions of the flow field as well as those regions of the flow that are of interest without concern for the surrounding cells. The main disadvantages of using an unstructured mesh lies in the added complexity and memory requirements of the flow solver.

The next section will review in general, research that has been reported in the area of unstructured grid flow code development. Other aspects of computational fluid dynamics will also be discussed that are precursors to the present research. The second section of this chapter will outline what was accomplished in this study and how it relates to other investigations.

1.1 Review of Related Work of Previous Investigators

In the recent past there has been a research thrust to develop flow codes that can be used on unstructured grids. Some of this development has come out of research in computer graphics [8]. The finite element methods have employed unstructured grids in structures problems and have recently been used to obtain solutions of the compressible flow equations. Peraire et al. [9] have reported solutions of the Euler equations in two dimensions using a finite element solution algorithm. They used linear triangular elements with explicit time stepping. The grid generation was done by the advancing front method. This technique will be discussed briefly in the chapter on grid generation.

Holmes and Snyder [10] demonstrated the use of a Delaunay triangulator to generate a mesh about arbitrary geometries. The grid generation used in the current work is based on this technique. Holmes and Connel [11] later reported on the use of this triangular grid generation along with a body fitted quadrilateral mesh to solve the two-dimensional Navier-Stokes equations. The quadrilateral grid was used near a solid wall boundary where one-dimensional refinement was desired. The code was written in a quasi-three-dimensional form to include the effects of radius change, stream tube variation, and rotation which occur in turbomachinery blade row flows. A central difference finite volume algorithm was used with a Runge-Kutta time integration scheme for the computation of viscous flows.

The finite volume method was used by Jameson and Mavriplis [12] to compute the solution of the two-dimensional Euler equations. Here comparisons were made between the use of a structured mesh and a structured mesh that had been triangulated. A central difference formulation was used and an explicit Runge-Kutta scheme was incorporated to advance the solution to a steady state. A residual averaging technique was added to relax the Courant-Fredrichs-Lewy limit. A multigrid scheme was also used to quickly remove the high frequency errors from the solution. Solutions were computed on a NACA 0012 airfoil and a KORN airfoil to demonstrate the capability of the unstructured computational technique.

A comparison of triangular and quadrilateral grid based flow codes was made by Lindquist and Giles [13]. It was found that the second order node-based schemes of Ni and that of Jameson work as well on triangular grids as they do on quadrilateral grids.

The Navier-Stokes equations were solved by Mavriplis et al. [14] on a regular unstructured triangular grid with a five stage Runge-Kutta type scheme and multigrid. This research showed that it was possible to use highly stretched triangles when computing high Reynolds number flows about airfoils. The scheme was found to be competitive with the structured viscous flow solvers both in accuracy as well as computational efficiency.

Several researchers have used upwind schemes on unstructured triangular grids. A Newton iteration method was used by Venkatakrishnan and Barth [15] to solve the Euler and Navier-Stokes equations implicitly. The inviscid fluxes were computed using the Roe flux difference splitting scheme. The viscous terms were formulated in two ways. The first approach made use of an approximation similar to the thin layer approximation for structured solvers. This means that only four cells were needed to compute the viscous terms. The second approach made no such approximation and required ten cells to compute the viscous terms. Computations were made for the NACA 0012 airfoil.

Another study of the application of upwind schemes to unstructured triangular mesh based solvers was carried out by Barth and Jespersen [16]. The Euler equations were solved using a finite volume discretization and an explicit time stepping scheme. Both cell-centered and cell-vertex schemes were considered. A multi-dimensional reconstruction of cell averaged values was used with a Roe flux function to compute flux quantities on the edges of the control volume. First order and higher order schemes were used to compute flows on the NACA 0012 airfoil. The higher order scheme was used for computing flow about a three element airfoil.

Whitaker and Grossman [17] used a Roe approximate Riemann solver and a four stage Runge-Kutta time integration on an unstructured triangular grid to obtain flow solutions. A non-standard weighting of the Runge-Kutta stages was used to accelerate the solutions to convergence. The scheme was demonstrated on a Mach 6.57 flow over a blunt body, a shock reflection problem, NACA 0012 airfoil, RAE 2822 airfoil, and a Karman-Trefftz airfoil with flap.

Unstructured grids have also been used by Caruso [18] to predict the performance of airfoils with leading edge ice accretions. Both Euler and Navier-Stokes solutions were computed. A central differencing scheme was used with an added isotropic dissipation term to prevent odd-even decoupling. A fourth-order Runge-Kutta scheme was used to integrate the equations in time. The study also included time dependent ice-growth problems. This showed that the use of unstructured grids allows for local grid refinement that is generally unavailable with structured grids.

Unsteady flows have also been addressed with unstructured grids. Batina [19] computed flows with both a central difference scheme and an upwind method. A Runge-Kutta integration scheme was used with a global constant time step to maintain time accuracy. Computations were made on a harmonic rolling swept flat-plate delta wing. The central difference method was also used to compute the inviscid unsteady three-dimensional flow over the Langley supersonic fighter. The upwind Euler scheme was tested on the steady three-dimensional ONERA M6 wing.

Unstructured grids have also been used to predict the inviscid flow over bodies in relative motion. Hase et al. [20] employed a cell-centered upwind Euler solver along with a point Gauss-Seidel relaxation scheme. Flow at a Mach number of 0.8 was computed for a NACA 0012 airfoil falling away from a solid wall.

Usab and Jiang [21] reported on a scheme to compute quasi-three-dimensional inviscid flow in turbomachinery cascades. The explicit Runge-Kutta finite-volume time-marching scheme was used on an adaptive unstructured grid. The solution scheme was demonstrated on a multi-element airfoil. Quasi-three-dimensional cascade flow was predicted on the NASA Rotor 67 at three spanwise locations.

High Reynolds number flows were addressed by Barth [22]. An edge based data structure was used in this formulation. Barth found it advantageous to use stretched cells in computing high Reynolds number flows and concluded that the triangulation method should minimize the maximum angle of a given stretched triangle. Turbulence modeling on unstructured grids was also addressed. A combination of an implicit sparse matrix solver and an explicit Runge-Kutta scheme was used to march the Navier-Stokes equations to a steady-state. Turbulent flow was computed over the RAE 2822 airfoil and an airfoil with a flap.

Other implicit methods have been used by Venkatakrishnan and Mavriplis [23] to solve the matrix equation resulting from the discretization of the Euler and Navier-Stokes equations. A preconditioned generalized minimum residual technique was developed to solve for the inviscid or viscous flows over various airfoil configurations. Three different preconditioners were tested with the solver (incomplete LU factorization(ILU), block diagonal factorization, and symmetric successive over-relaxation(SSOR)). The iterative schemes using ILU and SSOR as solvers were also investigated.

Several researchers have extended their methods to three dimensions [24] - [26]. Baker [24] has addressed the problem of solid surface modeling for generating tetrahedral meshes. Batina [26] has extended an implicit upwind solver to three dimensions. The flux difference splitting of Roe was used with a Gauss-Seidel relaxation procedure. The inviscid flow was computed over a Boeing 747 aircraft.

Recently an unstructured Euler solver was implemented on a massively parallel computer by Das et al. [27]. A multi-stage Runge-Kutta scheme was used to integrate the three-dimensional Euler equations. A node reordering technique was used to cluster local grid points in memory. This essentially reduces the bandwidth of the cell connectivity array. Inviscid solutions were computed on the ONERA M6 wing and an aircraft configuration.

Researchers have often found it difficult to solve the compressible Navier-Stokes equations at low Mach numbers. Several investigators have developed schemes on structured grids that solve the compressible equations at low Mach number by using a technique of preconditioning [28] - [31]. Choi and Merkel [28] used the Euler equations and computed flow over a bump at a Mach number as low as 0.05. The preconditioning in this case only affected the time derivative of the energy equation. Later Choi and Merkel [32] included time-derivative preconditioning for the Navier-Stokes equations. To date none of the methods proposed for unstructured grids have utilized Mach number preconditioning.

1.2 Scope of Current Research

The research in this study considers the use of unstructured grids in predicting low Mach number flows through internal geometries. To date. there has not been much work done toward applying unstructured grids to viscous internal flows at low Mach numbers.

The Euler or Navier-Stokes equations are discretized in finite volume form using primitive variables; however, the conservation law form is retained. The equations are solved iteratively using the implicit Gauss-Seidel method. Another implicit method of solution used in this work is a commercially available sparse matrix package. Several iterative conjugate gradient based solvers and matrix preconditioners are considered.

The grid generation is similar to that reported by Holmes and Snyder [10]. The method of Delaunay triangulation is used to discretize the computational domain. The initial points are the discretized boundaries. The interior is then resolved by adding new points to the computational domain that satisfy the Delaunay criterion.

Computer memory is minimized by storing only the non-zero block matrices of the larger sparse matrix equation. Preconditioning of the time derivative term is used to allow efficient calculations at vanishingly small Mach numbers. The equations can be marched in real or pseudo time for a steady state solution. However, both real and pseudo derivative terms are retained so that time dependent problems can be computed in later research.

A coloring scheme was implemented so that the Gauss-Seidel algorithm could take advantage of the vector processor capabilities. This was done with an algorithm based on the four color theorem which was proven in 1976 through exhaustive computation [33]. The added minimal memory requirements were offset by a significant decrease in computer time required by the colored Gauss-Seidel iterative procedure. The overall effect on the convergence rate was minimal as expected.

Results are given for some typical test cases which have been computed by other investigators on structured grids. Other test cases are used to demonstrate the capability of the unstructured grid flow code approach. Solutions are first shown for inviscid flow over a bump at subsonic and transonic Mach numbers. Viscous solutions are then presented for developing flow in a channel at various Reynolds numbers. The advantage of using temporal preconditioning was demonstrated when solving over a wide range of Mach numbers. The sparse iterative solver SITRSOL was also used for comparison with the Gauss-Seidel relaxation scheme. A symmetric sudden expansion was used to demonstrate the capability of the code to compute separated flows. The viscous flow over a cascade of tandem circular cylinders was also computed. Finally, the flow in a four-port valve at two different Reynolds numbers was computed to show the geometric capability that is available for applying boundary conditions through the use of unstructured grids.

2. GOVERNING EQUATIONS

The Navier-Stokes equations were used to model viscous fluid flow problems in this study. In conservation law form and physical coordinates these equations can be written in vector form as

$$\frac{\partial U}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} = 0 \tag{2.1}$$

where the vectors

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix}, \qquad E = \begin{bmatrix} \rho u \\ \rho u^2 + P - \tau_{xx} \\ \rho uv - \tau_{xy} \\ (P+e)u - u\tau_{xx} - v\tau_{xy} + q_x \end{bmatrix},$$

and

$$F = \begin{bmatrix} \rho v \\ \rho uv - \tau_{xy} \\ \rho v^2 + P - \tau_{yy} \\ (P+e)v - u\tau_{xy} - v\tau_{yy} + q_y \end{bmatrix}.$$

In this work only Newtonian fluids will be considered, so the shear stress tensors are defined as

$$\begin{aligned}
\tau_{xx} &= -\tfrac{2}{3}\mu(u_x + v_y) + 2\mu u_x = \tfrac{2}{3}\mu(2u_x - v_y),\\
\tau_{xy} &= \mu(v_x + u_y),\\
\tau_{yy} &= -\tfrac{2}{3}\mu(u_x + v_y) + 2\mu v_x = \tfrac{2}{3}\mu(2v_y - u_x),
\end{aligned} \tag{2.2}$$

and

$$\begin{aligned}
q_x &= -\kappa T_x,\\
q_y &= -\kappa T_y.
\end{aligned} \tag{2.3}$$

If a further assumption is made that the gas is ideal. where $\rho = P/RT$ with R being the gas constant per unit mass, the Navier-Stokes equations can be written as

$$\frac{\partial Q(w)}{\partial t} + \frac{\partial G(w)}{\partial x} + \frac{\partial H(w)}{\partial y} = 0 \tag{2.4}$$

with

$$w = \begin{bmatrix} P \\ u \\ v \\ T \end{bmatrix},$$

$$Q = \begin{bmatrix} \frac{P}{T} \\ \frac{Pu}{T} \\ \frac{Pv}{T} \\ \frac{P}{T}[(C_p - R)T + \frac{u^2}{2} + \frac{v^2}{2}] \end{bmatrix}.$$

$$G = \begin{bmatrix} \frac{Pu}{T} \\ \frac{Pu^2}{T} + RP - \tau xx \\ \frac{Puv}{T} - \tau xy \\ \frac{P}{T}(C_pT + \frac{u^2}{2} + \frac{v^2}{2})u - u\tau xx - v\tau xy + qx \end{bmatrix},$$

and

$$H = \begin{bmatrix} \frac{Pv}{T} \\ \frac{Puv}{T} - \tau xy \\ \frac{Pv^2}{T} + RP - \tau yy \\ \frac{P}{T}(C_pT + \frac{u^2}{2} + \frac{v^2}{2})v - u\tau xy - v\tau yy + qy \end{bmatrix}.$$

The test cases presented in this work involve the laminar flow of air where the viscosity is assumed to follow the Sutherland formula,

$$\mu = \frac{C_1 T^{\frac{3}{2}}}{T + C_2} \tag{2.5}$$

where C_1 and C_2 are equal $1.458 \times 10^{-6} kg/(m \cdot s \cdot \sqrt{^oK})$ and 110.4^oK respectively.

The equations are nondimensionalized by using the substitutions

$$\begin{bmatrix} \tilde{t} \\ \tilde{x} \\ \tilde{y} \\ \tilde{u} \\ \tilde{v} \\ \tilde{P} \\ \tilde{T} \\ \tilde{\mu} \\ \tilde{R} \\ \tilde{C}p \\ \tilde{C}_1 \\ \tilde{C}_2 \end{bmatrix} = \begin{bmatrix} \frac{t}{L_{ref}/u_{ref}} \\ \frac{x}{L_{ref}} \\ \frac{y}{L_{ref}} \\ \frac{u}{u_{ref}} \\ \frac{v}{u_{ref}} \\ \frac{P}{\rho_{ref}u_{ref}^2} \\ \frac{T}{T_{ref}} \\ \frac{\mu}{\mu_{ref}} \\ \frac{R}{u_{ref}^2/T_{ref}} \\ \frac{Cp}{u_{ref}^2/T_{ref}} \\ \frac{C_1}{\mu_{ref}/\sqrt{T_{ref}}} \\ \frac{C_2}{T_{ref}} \end{bmatrix} . \tag{2.6}$$

The $\tilde{}$ refers to a term that is nondimensional. The variables subscripted ref are reference quantities specific to a particular flow calculation.

The Navier-Stokes equations are now written in nondimensional form

$$\frac{\partial Q(\tilde{w})}{\partial \tilde{t}} + \frac{\partial G(\tilde{w})}{\partial \tilde{x}} + \frac{\partial H(\tilde{w})}{\partial \tilde{y}} = 0 \tag{2.7}$$

with

$$\tilde{w} = \begin{bmatrix} \tilde{P} \\ \tilde{u} \\ \tilde{v} \\ \tilde{T} \end{bmatrix},$$

$$Q = \begin{bmatrix} \frac{\tilde{P}}{\tilde{T}} \\ \frac{\tilde{P}\tilde{u}}{\tilde{T}} \\ \frac{\tilde{P}\tilde{v}}{\tilde{T}} \\ \frac{\tilde{P}}{\tilde{T}}[(\tilde{C}p - \tilde{R})\tilde{T} + \frac{\tilde{u}^2}{2} + \frac{\tilde{v}^2}{2}] \end{bmatrix}.$$

$$G = \begin{bmatrix} \frac{\tilde{P}\tilde{u}}{\tilde{T}} \\ \frac{\tilde{P}\tilde{u}^2}{\tilde{T}} + \tilde{R}\tilde{P} - \tilde{\tau}xx \\ \frac{\tilde{P}\tilde{u}\tilde{v}}{\tilde{T}} - \tilde{\tau}xy \\ \frac{\tilde{P}}{\tilde{T}}(\tilde{C}p\tilde{T} + \frac{\tilde{u}^2}{2} + \frac{\tilde{v}^2}{2})\tilde{u} - \tilde{u}\tilde{\tau}xx - \tilde{v}\tilde{\tau}xy - \frac{\tilde{C}p\tilde{\mu}\tilde{R}}{Pr\cdot Re}\tilde{T}x \end{bmatrix},$$

and

$$H = \begin{bmatrix} \frac{\tilde{P}\tilde{v}}{\tilde{T}} \\ \frac{\tilde{P}\tilde{u}\tilde{v}}{\tilde{T}} - \tilde{\tau}xy \\ \frac{\tilde{P}\tilde{v}^2}{\tilde{T}} + \tilde{R}\tilde{P} - \tilde{\tau}yy \\ \frac{\tilde{P}}{\tilde{T}}(\tilde{C}p\tilde{T} + \frac{\tilde{u}^2}{2} + \frac{\tilde{v}^2}{2})\tilde{v} - \tilde{u}\tilde{\tau}xy - \tilde{v}\tilde{\tau}yy - \frac{\tilde{C}p\tilde{\mu}\tilde{R}}{Pr\cdot Re}\tilde{T}y \end{bmatrix},$$

where

$$\begin{aligned} \tilde{\tau}xx &= \frac{2}{3}\frac{\tilde{R}\tilde{\mu}}{Re}(2\tilde{u}x - \tilde{v}y), \\ \tilde{\tau}xy &= \frac{\tilde{R}\tilde{\mu}}{Re}(\tilde{v}x + \tilde{u}y), \\ \tilde{\tau}yy &= \frac{2}{3}\frac{\tilde{R}\tilde{\mu}}{Re}(2\tilde{v}y - \tilde{u}x). \end{aligned} \qquad (2.8)$$

The Reynolds and Prandtl numbers are defined as

$$Re = \frac{\rho_{ref} u_{ref} R_{ref}}{\mu_{ref}}, \qquad Pr = \frac{Cp_{ref} \mu_{ref}}{\kappa_{ref}}$$

respectively. It is important to note that though these equations are written in terms of primitive variables(P, u, v, T) that they are still in conservation law form.

All subsequent equations are nondimensional so the $\hat{}$ is dropped for convenience.

3. GRID GENERATION

The numerical solution of the Euler or Navier-Stokes equations requires the discretization of the computational region in such a way that the geometry as well as the flow physics is predicted to a desired accuracy. Typically, computational grids have been generated with an inherent global structure in mind. This structure results in having grid lines coincident with specified boundaries of the flow domain. There are several advantages to using a structured discretization. First, grids can be easily generated about simple geometries. Second, the flow solver can exploit the grid structure for increased speed. However, it is difficult to generate a structured grid about complex geometries without using special grid patching or grid overlaying techniques. These are techniques which allow the use of more than one grid to discretize a complex computational domain. Special computer coding is required to handle the individual grids as discretized regions and the intersections of the grids. Also, grid adaptation to geometry or flow features must be done within the restrictions of the grid structure. If local refinement is desired in a structured grid, an entire row of cells must be added to the computational domain. This often results in adding cells in regions of the flow where refinement is not required resulting in a more costly computation.

Another method of discretizing a domain is to use an unstructured grid formulation. The triangular shaped cell is the simplest geometric shape that can be used to cover a two-dimensional computational domain. With an unstructured grid, individual cells can no longer refer to their neighbors simply by incrementing an index as in a structured grid. Instead, the neighborhood of a cell is determined through a connectivity matrix. This connectivity matrix usually contains cell based information as well as edge based information. Details of the connectivity matrix required by the computer flow code developed in this work will be discussed later in this chapter.

The use of a triangular unstructured grid formulation has some distinct advantages over a structured grid. One advantage to using triangular cells is that with them it is easy to generate grids about complex geometries. This reduces the amount of time required to generate a suitable grid. Also grid adaptation can be done locally without adding unnecessary cells to other regions of the domain. As a result, the calculation becomes more competitive with the structured grid formulation. One disadvantage of using an unstructured grid is the added memory necessary for the connectivity matrix. The added level of complexity in writing the flow code is another disadvantage to using unstructured grids.

Several methods can be used to generate an unstructured triangular mesh. One method is to simply place points in a domain and connect them by hand. This becomes inefficient very quickly as the number of points increase. The method of advancing front is another technique used to triangulate a region [9]. This method uses a background grid to control the placement of points in a domain through interpolation. The grid is generated by using the discretized boundary definition as an initial front and marching away from the boundaries into the desired triangulated region. A new point is placed in the domain based on certain prescribed criteria.

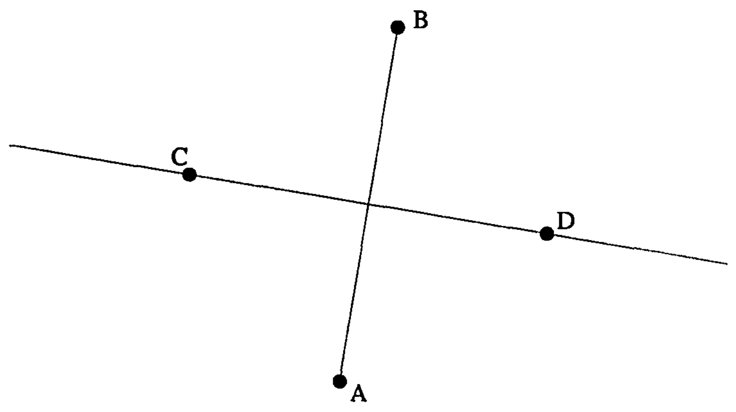


Figure 3.1: The perpendicular bisection of the line connecting the points A and B

These criteria could be the same as those used in the method of Delaunay triangulation which will be described later. The edges that connect to the new point to form a triangle now define the front. The fronts continue to advance into the domain from all boundaries until they coalesce and the entire region has been triangulated. The method of Delaunay triangulation is used in this work and will be described next.

Figure 3.1 shows two points, A and B, placed in a two-dimensional plane. The line that passes through points C and D is defined by the locus of points such that points with x, y coordinates below the line are closest to A and points with x, y coordinates above the line are closest to B. The same geometric construction can be done to a set of points, Fig. 3.2. Here a line is terminated when it intersects another since points crossing the intersection would violate the same geometric construction requirements of another set of points. This is done for every pair of closest points. The result is a region covered with nonoverlapping polygons. This is called a Dirichlet tessellation. An interesting feature of the individual polygons is that their vertices

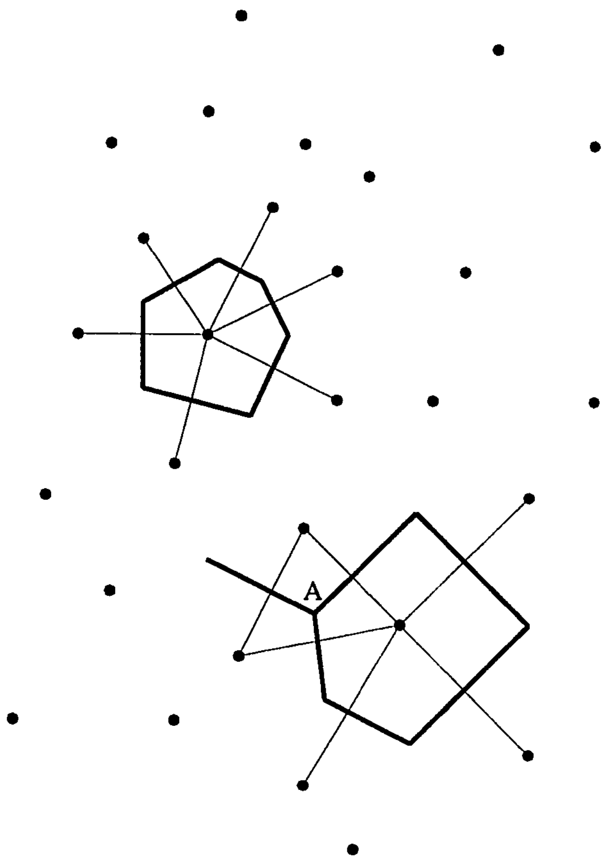


Figure 3.2: Dirichlet polygons

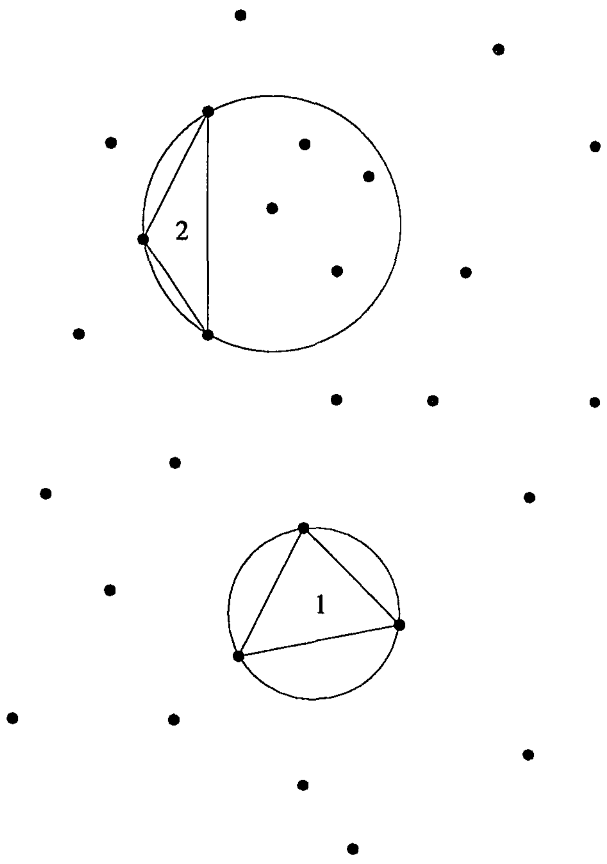


Figure 3.3: Delaunay triangulation

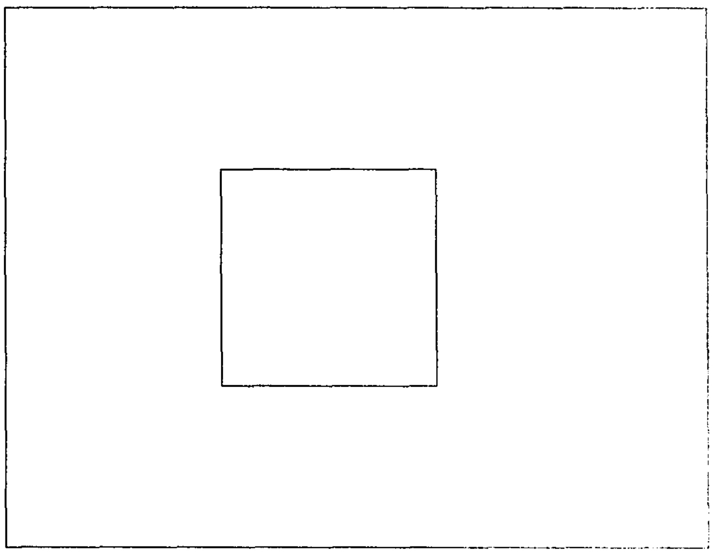


Figure 3.4: Boundary definition of a square hole in a rectangular outer boundary

are the centers of circles that pass through the three points of the triangle whose edges are bisected by the sides of the polygons, such as point A in Fig 3.2. These triangles represent the Delaunay triangulation of the domain. A triangle is considered Delaunay if the circle that passes through the vertices of the triangle contain no other points of the domain. The triangle 1 of Fig. 3.3 is Delaunay. The triangle 2 is not Delaunay since its circumcircle surrounds other points in the region other than its three vertex points. There are many methods of generating a Delaunay triangulation of a region. The method described above starts with a domain that is covered with points. The points are then triangulated according to the Delaunay criterion.

The method used in this work follows a path similar to that of Holmes and Snyder [10] to triangulate a region. First, the boundaries that describe the computational domain are defined, Fig. 3.4. Here a square hole is surrounded by a rectangular outer

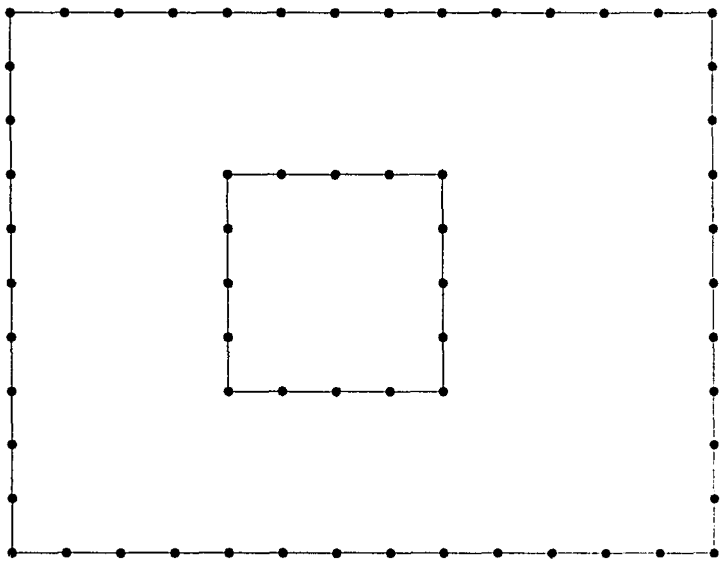


Figure 3.5: Discretization of the boundaries

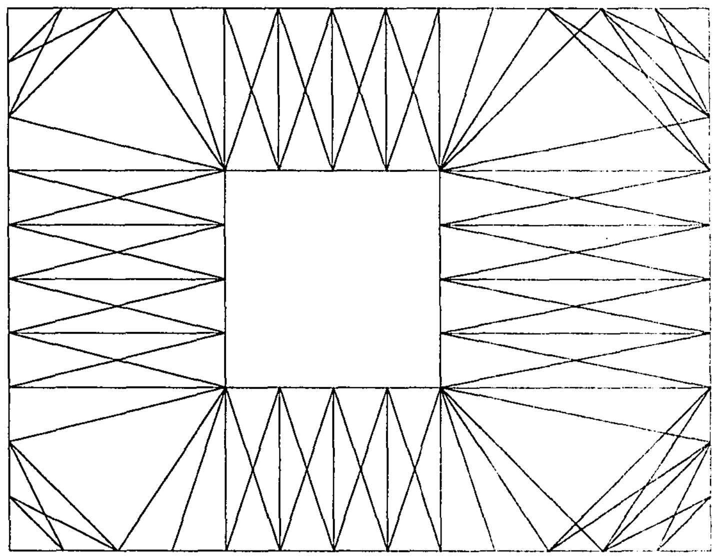


Figure 3.6: Triangulation of boundary points

boundary. The region that is to be triangulated is the area between the inner hole and the outer boundary. Next, the boundaries are discretized in a counter-clockwise direction, Fig. 3.5. These discrete points are then triangulated using the Delaunay criterion(recall that a triangle whose circumcircle contains other points in the computational domain is not Delaunay), Fig. 3.6. The result of this triangulation is not usually desirable. Points must now be added to the domain to obtain a reasonable grid. A new point can be added based on any criteria one chooses. The grid point insertion in the current work is done according to one of the following three geometric criteria: improve the triangle with the smallest aspect ratio, reduce the maximum area triangle, reduce the size of the triangle with the largest circumcircle radius. These geometric constraints can be used in any combination and their definitions will be described below. Some other criteria that could be used for local retriangulation are increase minimum angle, decrease maximum angle, and maintain equal length sides, to name a few.

One refinement criterion used in this work is based on the definition of the aspect ratio of a triangle.

$$AspectRatio = \frac{ri}{2rc} \tag{3.1}$$

where ri is the radius of the inscribed circle of the triangle.

$$ri = \frac{\sqrt{s(s-a)(s-b)(s-c)}}{s}, \tag{3.2}$$

and rc is the radius of the circumscribed circle of the triangle,

$$rc = \frac{abc}{4\sqrt{s(s-a)(s-b)(s-c)}}. \tag{3.3}$$

The variable s is defined as the semiperimeter of a triangle,

$$s = \frac{1}{2}(a+b+c) \tag{3.4}$$

and the quantities a, b, c refer to the lengths of the sides of the triangle. A new point is placed at the circumcenter of the triangle with the smallest aspect ratio.

Another refinement criterion is based on the area of a triangle.

$$Area = \sqrt{s(s-a)(s-b)(s-c)}. \tag{3.5}$$

where s is the semiperimeter defined above. A new point is placed at the circumcenter of the triangle with the largest area.

A third criterion is based on the radius of the circumcircle of a triangle as defined in Eq. (3.3). Again a new point is placed at the circumcenter of the triangle with the largest circumcircle radius. This criterion takes into account both triangles with bad aspect ratios and ones with large areas.

As an example, the aspect ratio criterion was used to put a new point in the previously described domain. First, the triangles were searched for the one with the smallest aspect ratio, Fig. 3.7. A new point was placed at the circumcenter of the triangle. This point actually lies outside the targeted triangle. Recall that a triangle is not Delaunay if any of its three vertices lies within the circumcircle of another triangle. The triangles that violate the Delaunay criterion are now deleted. Fig. 3.8. The points of the old triangulation, however, are not removed. These old points may

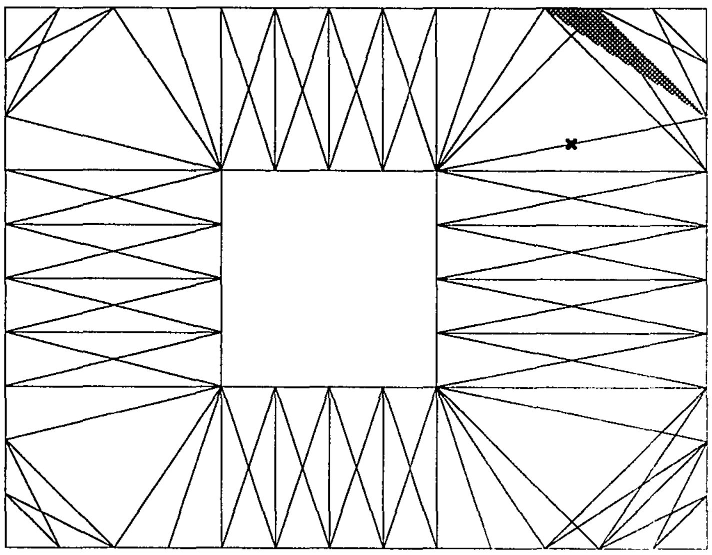


Figure 3.7: Triangle with bad aspect ratio and its circumcenter

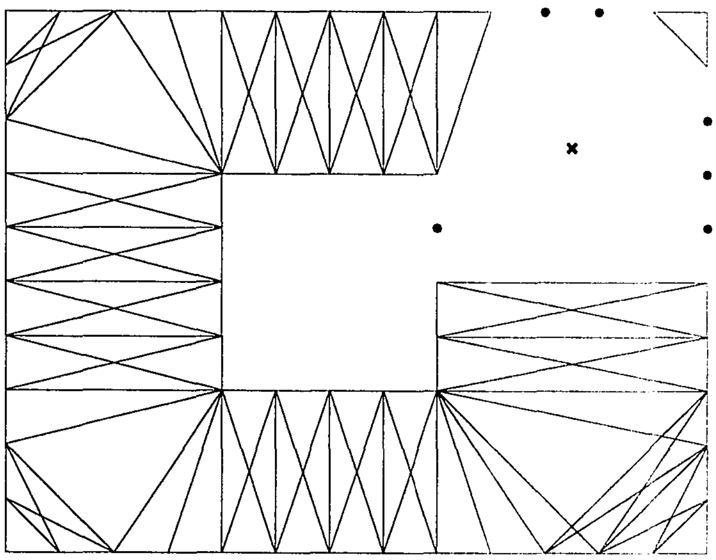


Figure 3.8: Deletion of cells that violate the Delaunay criterion

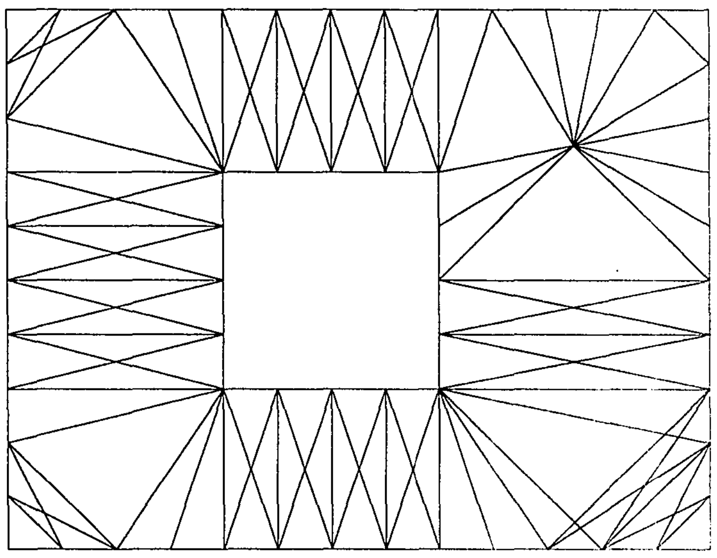


Figure 3.9: Local retriangulation

now simply be connected to the new point to form triangles that automatically satisfy the Delaunay criterion, Fig. 3.9. This refinement then continues until a satisfactory grid is obtained, Fig. 3.10.

The flow code requirements dictate the type of output that the grid generation scheme must provide. A connectivity array must be generated for an unstructured grid so that a cell neighborhood is completely defined for the flow code. The flow code can be a vertex based or a cell center based scheme. The vertex based scheme uses a dual cell as its control volume. The dual cell could be the Dirichlet polygons referred to earlier in this chapter. This scheme requires certain geometric information about the cells that share a given vertex for use to compute the flow domain. The code in the current work is based on a cell centered scheme. Here the triangle itself is the control volume used in the finite volume formulation.

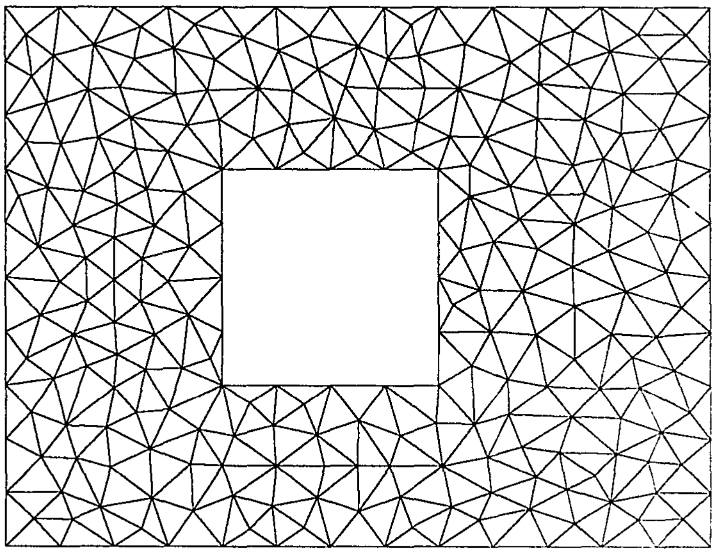
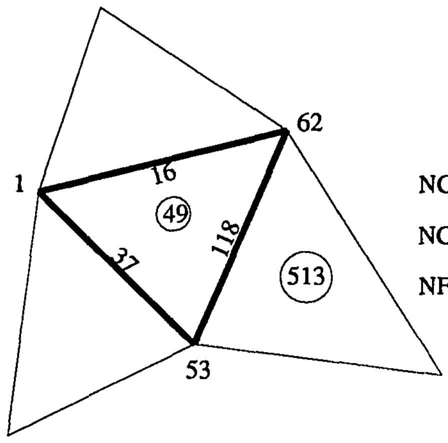


Figure 3.10: Final coarse triangulation of domain



NCELL(1:3,49)=37,118,16

NCELL(4:6,49)=1,53,62

NFACE(1:2,118)=49,513

Figure 3.11: Connectivity requirements for a single cell

Connectivity is determined by cell nodes, cell faces, and face cells. Cell nodes are the nodes at the vertices of a triangle. The cell faces are the edges of a triangle. The face cells are the cells that lie on either side of a given edge. Typically the x, y coordinates of the cell nodes are the only grid floating point numbers required as input by a flow code. The grid connectivity is defined through integer arrays. Two connectivity arrays were needed by the flow code developed in this work to define the discretized geometry of a computational domain, Fig. 3.11. The two-dimensional array $NCELL$ contains the edge and node connectivity for a given cell. The first dimension of $NCELL$ contains 6 elements. The second dimension has a length n, where n is the total number of triangular cells in the computational domain. Consider the cell number i. The first three elements of the first dimension of array $NCELL$ are the edge numbers of cell i. The last three elements of the first dimension of array $NCELL$ are the vertex node numbers of cell i. This allows the access to the edge and node numbers that define a given cell, in this case cell number i. The array $NFACE$ is also two-dimensional. It contains cell connectivity for a specific edge. The first dimension is of length 2. The second dimension is of length m, where m is the total number of edges that make up the computational domain. The second dimension identifies the edge(in this example, j). The first dimension of array $NFACE$ contains the cell numbers that are adjacent to one another sharing the common edge j. These two integer arrays along with the floating point arrays x and y define the geometry for the flow code.

Boundary information must be defined explicitly. Solid wall, exit, and inlet boundaries are implied through the edge connectivity array, $NFACE$. For a solid wall boundary, one of the elements of the first dimension of the array $NFACE$ will contain the value 0. This tells any cell that refers to that edge that it borders a solid wall boundary. Similarly, an exit boundary is adjacent to a cell number of -1, and an inlet boundary borders a cell number of -2. Periodic and symmetric boundaries are handled through special connectivity. This again is done by including the appropriate cell information in array $NFACE$. A symmetry boundary cell will have an edge that borders itself. So the first dimension of array $NFACE$ for the symmetry face will have both elements referring to the same cell number. For a periodic boundary the elements will refer to cell numbers that are separated by one periodic pitch. Boundary conditions will be addressed later.

This unstructured grid generation code can handle both simply and multiply connected regions, Figs. 3.12, 3.13.

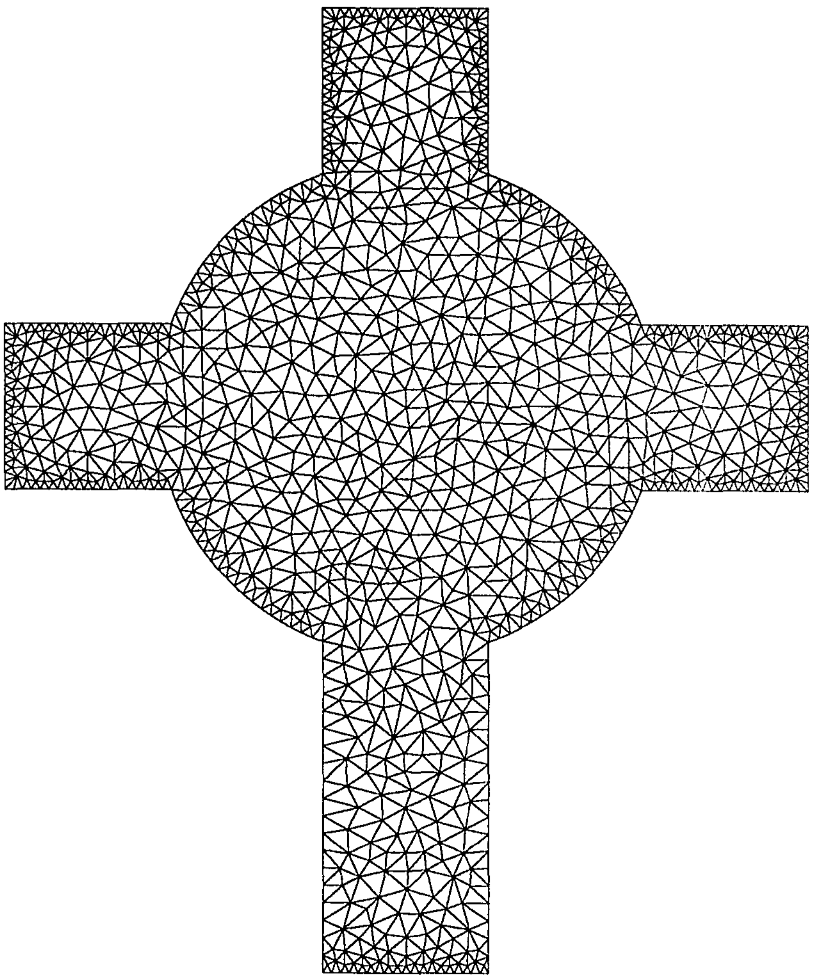


Figure 3.12: Triangulation of a simply connected region

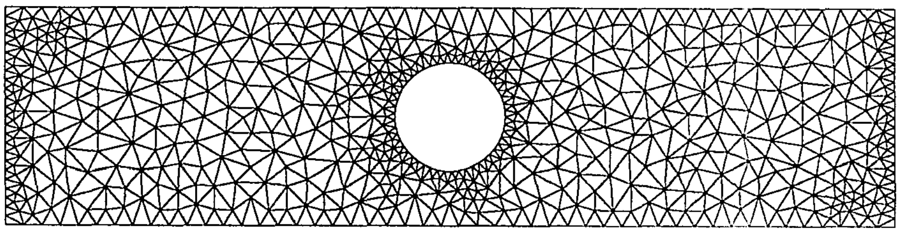


Figure 3.13: Triangulation of a multiply connected region

4. NUMERICAL APPROACH

This chapter describes the finite volume discretization used for the Navier-Stokes equations. A preconditioning technique will be discussed which allows the efficient solution of the compressible flow equations at low Mach numbers. The numerical implementation of the boundary conditions is then examined. Two methods of adding artificial dissipation to the scheme to prevent odd-even decoupling are then discussed. Finally, the methods used to solve the resulting matrix equation are described. This includes a coloring scheme used for vectorization.

4.1 Discretization Technique

The finite volume formulation of the governing equations is well suited for application to an unstructured discretization of the flow domain. The nondimensional Navier-Stokes equation written in differential form Eq. (2.7) is first recast in integral form for an arbitrary volume, Ω as

$$\int_\Omega \frac{\partial Q}{\partial t} d\Omega + \int_\Omega \left(\frac{\partial G}{\partial x} + \frac{\partial H}{\partial y} \right) d\Omega = 0. \tag{4.1}$$

Using Gauss's theorem, the area integral of the flux derivatives can be rewritten as the surface integral of the flux quantities around the area Ω. This allows the Eq. (4.1) to be written as

$$\frac{\partial}{\partial t}\int_{\Omega} Q d\Omega + \oint_{\Gamma} G dy - H dx = 0. \tag{4.2}$$

For each control volume consisting of a triangular element, Eq. (4.2) is evaluated as

$$\frac{\partial}{\partial t}(A_i Q_i) + \sum_{j=1}^{3}(G_j \triangle y_j - H_j \triangle x_j) = 0. \tag{4.3}$$

where Q_i is the vector of conserved quantities in cell i, G_j and H_j are the flux vector quantities across edge j, and Δx_j and Δy_j are the differences in Cartesian nodal coordinates that define edge j. The summation on j proceeds in a counterclockwise manner around the edges of cell i. Also it is understood that Δx_j stands for $x(end) - x(beginning)$ as the evaluation proceeds in a counterclockwise manner around the sides of a control volume. The quantity A_i is the area of cell i defined as

$$A_i = \sqrt{s(s-a)(s-b)(s-c)}. \tag{4.4}$$

The variable s is the semiperimeter of cell i, defined previously by Eq. (3.4), and the quantities a, b, c refer to the lengths of the sides of the cell i. Cell face flow quantities required by Eq. (4.3) for the computation of the inviscid flux terms were approximated by using the average of the cell centered values on both sides of a given cell face. The numerical integration of these quantities around the edges of the cell results in a central difference scheme that is second order accurate in space. The viscous terms require the computation of the derivatives on the faces of the triangle control volume. To compute these terms, the level 2 cells shown in Fig. 4.1 were used and a different path integral was evaluated. The algorithm used to obtain the cell numbers and orientations is discussed in Appendix A. Again this yields a second order accurate scheme in space. A total of 10 cell centered quantities was used in the computation of the viscous quantities of the summation term of Eq. (4.3). Specifically the viscous terms in the x-momentum equation are written as

$$\sum_{j=1}^{3}(-\tau_{xx}\triangle y+\tau_{xy}\triangle x)_j=\sum_{j=1}^{3}\frac{\mu R}{Re}\left[-\frac{2}{3}\left(2\frac{\partial u}{\partial x}-\frac{\partial v}{\partial y}\right)\triangle y+\left(\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}\right)\triangle x\right]_j, \tag{4.5}$$

where the summation is over the three sides of the triangular control volume. The definition of the derivatives

$$\begin{aligned}&\frac{\partial u}{\partial x},\frac{\partial v}{\partial y},\\&\frac{\partial v}{\partial x},\frac{\partial u}{\partial y}\end{aligned} \tag{4.6}$$

must be given on all three edges of the triangular control volume. Consider the evaluation of a typical x and y derivative term on edge number 12 shown in Fig. 4.2. This is the edge that represents the border between cell A and B. Recall that all geometric quantities are computed in a counterclockwise manner. The derivatives can be recast in integral form as

$$\frac{\partial u}{\partial x}=\frac{1}{S'}\oint_{\Gamma'}udy \tag{4.7}$$

and

$$\frac{\partial v}{\partial y}=-\frac{1}{S'}\oint_{\Gamma'}vdx \tag{4.8}$$

where S' is the sum the areas of the two cells across a given edge, and the integral is along the path that traverses the outer boundary of the two cells in a counterclockwise direction. These derivatives are interpreted as mean values over the area S'. The above derivatives can be written for side 12 as

$$\frac{\partial u}{\partial x}|_{12} = \frac{1}{S_{AB}} \Big[\tfrac{1}{2}(u_B + u_E)\,\Delta y_{25} + \tfrac{1}{2}(u_B + u_F)\,\Delta y_{26} + \tfrac{1}{2}(u_A + u_C)\,\Delta y_{13} + \tfrac{1}{2}(u_A + u_D)\,\Delta y_{14}\Big]$$

and

$$\frac{\partial v}{\partial y}|_{12} = -\frac{1}{S_{AB}} \Big[\tfrac{1}{2}(v_B + v_E)\,\Delta x_{25} + \tfrac{1}{2}(v_B + v_F)\,\Delta x_{26} + \tfrac{1}{2}(v_A + v_C)\,\Delta x_{13} + \tfrac{1}{2}(v_A + v_D)\,\Delta x_{14}\Big].$$

Similar terms are computed for sides 13 and 14 of the cell A. These equations are then written in delta form by substituting

$$u = \hat{u} + \Delta u$$

and

$$v = \hat{v} + \Delta v$$

into Eq. (4.5) where $\hat{u}$ is a provisional value. The delta form will be discussed again later when presenting the linearization of the convective terms. The additional cells used in the computation of the viscous terms affect the storage required for the solution of the resulting equation. The storage requirements will be discussed in more detail in the section on solvers.

The system of equations was integrated in time using an implicit scheme written in delta form. Newton linearization was used on nonlinear terms. For example, the terms

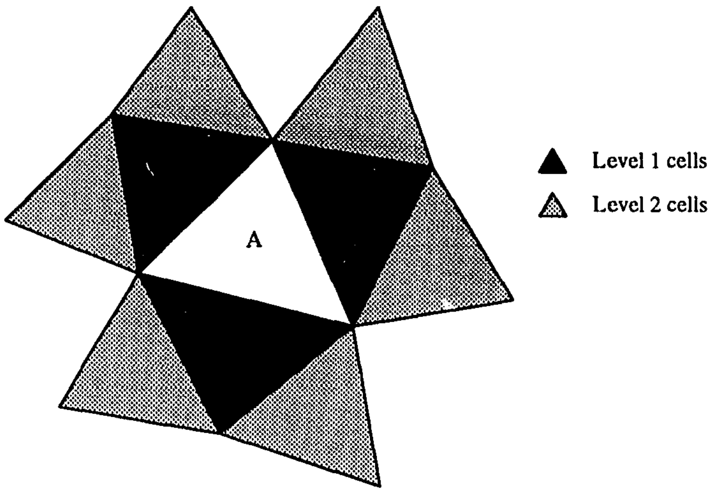


Figure 4.1: Cell level dependence

$$\begin{aligned}
\frac{P}{T} &\simeq \frac{\hat{P}}{\hat{T}} + \frac{1}{\hat{T}} \triangle P - \frac{\hat{P}}{\hat{T}^2} \triangle T, \\
\frac{Pu}{T} &\simeq \frac{\hat{P}\hat{u}}{\hat{T}} + \frac{\hat{P}}{\hat{T}} \triangle u + \frac{\hat{u}}{\hat{T}} \triangle P - \frac{\hat{P}\hat{u}}{\hat{T}^2} \triangle T, \\
\frac{Pv}{T} &\simeq \frac{\hat{P}\hat{v}}{\hat{T}} + \frac{\hat{P}}{\hat{T}} \triangle v + \frac{\hat{v}}{\hat{T}} \triangle P - \frac{\hat{P}\hat{v}}{\hat{T}^2} \triangle T
\end{aligned} \tag{4.9}$$

are substituted into the continuity and momentum equations. The ^ terms take on provisional values of the primitive variables; and the delta quantities represent the differences between values at the new time level and the provisional values, e.g., $\Delta u = u - \hat{u}$. These equations can be iterated at a specific time level until the linearization errors are reduced to a satisfactory value. For steady state problems, this iteration process at each time level is not necessary. Similar terms are used to linearize the energy equation. However, the nonlinear dissipation terms in the energy equation were linearized by evaluating them explicitly in terms of the provisional

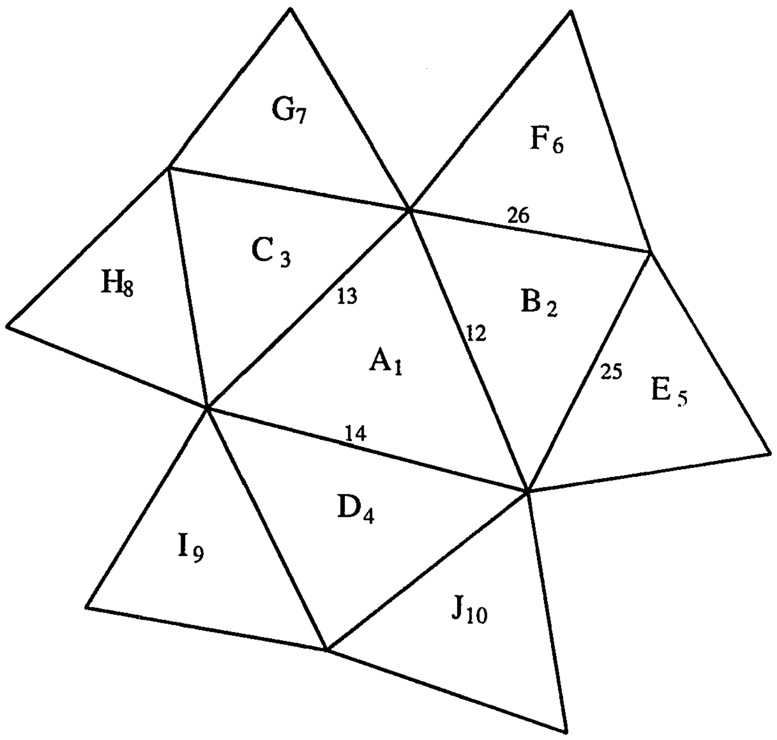


Figure 4.2: Cells used in the computation of viscous terms for cell A

values, rather than using Newton linearization. The result of the above substitutions is the matrix equation

$$\mathbf{A}\vec{x} = \vec{b}. \tag{4.10}$$

The matrix **A** contains the linearized terms which multiply the vector of unknown delta quantities, $\vec{x}$. The vector $\vec{b}$ represents the residual of the equations which should go to zero as the solution approaches convergence. There are four equations written for each cell in the computational domain. For the central difference formulation given here, each cell is dependent on the level one cells through the convective terms as well as the level two cells through its diffusive terms, as illustrated in Fig. 4.1. This along with the unstructured grid yields a sparse matrix whose elements are block 4×4 matrices. The general row of the matrix **A** has ten non-zero blocks of coefficients. An example of a typical row of the A matrix is shown in Appendix B. The number of blocks vary near a boundary.

The solution of Eq. (4.10) is not straight forward. Several methods of solution will be discussed later in the section on sparse matrix solvers. The results from the solution of the matrix equation represent the average of the flow quantities over the entire cell. The values that were computed depends on the accuracy of the scheme(e.g., a second order accurate scheme will give a linear distribution of the flow quantities in the cell). If a solution were to be reconstructed based on the accuracy of the current method on the discretized flow domain, the solution would be linear within a given triangular cell. The global accuracy of the method was second order.

4.2 Preconditioning

Solving the compressible flow equations for low Mach number cases is difficult because the resulting system is stiff due to the large ratio of acoustic and convective velocities. Even the convergence rate of predominantly supersonic flow can be slowed or halted if it contains local regions of low Mach number flow. A temporal preconditioning is used in this work to remove this stiffness.

The approach will be demonstrated by using the Navier-Stokes equations in one dimension. The nondimensional form of these equations is written as

$$\frac{\partial Q(w)}{\partial t} + \frac{\partial G(w)}{\partial x} - \frac{\partial G_v(w)}{\partial x} = 0, \tag{4.11}$$

where

$$w = \begin{bmatrix} P \\ u \\ T \end{bmatrix},$$

$$Q = \begin{bmatrix} \frac{P}{RT} \\ \frac{Pu}{RT} \\ \frac{P}{\gamma R} + \frac{M^2(\gamma-1)}{2}\frac{Pu}{RT} \end{bmatrix},$$

$$G = \begin{bmatrix} \frac{Pu}{RT} \\ \frac{Pu^2}{RT} + P \\ \frac{Pu}{R} + \frac{M^2(\gamma-1)}{2}\frac{Pu^2}{RT} \end{bmatrix},$$

$$G_v = \begin{bmatrix} 0 \\ \frac{4\mu}{3Re}u_x \\ \frac{4\mu}{3Re}uu_x - \frac{\mu}{PrRe}T_x \end{bmatrix}.$$

The quantity M is the reference Mach number, defined as $u_{ref}/\sqrt{\gamma RT_{ref}}$, which appears when the equations are cast in nondimensional form. This equation can be rewritten as

$$\mathbf{A}_t\frac{\partial w}{\partial t} + \mathbf{A}_x\frac{\partial w}{\partial x} = \frac{\partial G_v(w)}{\partial x}, \tag{4.12}$$

where the Jacobian matrices, $\mathbf{A}_t$ and $\mathbf{A}_x$ are defined as

$$\mathbf{A}_t = \begin{bmatrix} \frac{1}{RT} & 0 & -\frac{P}{RT^2} \\ \frac{u}{RT} & \frac{P}{RT} & -\frac{Pu}{RT^2} \\ \frac{1}{\gamma R} + \frac{M^2(\gamma-1)}{2}\frac{u}{RT} & \frac{M^2(\gamma-1)}{2}\frac{P}{RT} & -\frac{M^2(\gamma-1)}{2}\frac{Pu}{RT^2} \end{bmatrix}. \tag{4.13}$$

and

$$\mathbf{A}_x = \begin{bmatrix} \frac{u}{RT} & \frac{P}{RT} & -\frac{Pu}{RT^2} \\ \frac{u^2}{RT} + 1 & \frac{2Pu}{RT} & -\frac{Pu^2}{RT^2} \\ \frac{u}{R} + \frac{M^2(\gamma-1)}{2}\frac{u^2}{RT} & \frac{P}{R} + M^2(\gamma-1)\frac{Pu}{RT} & -\frac{M^2(\gamma-1)}{2}\frac{Pu^2}{RT^2} \end{bmatrix}. \tag{4.14}$$

It is important to note that the quantity P/RT is well behaved as Mach number goes to zero since it is equal to the nondimensional density, ρ/ρ_{ref}. Also note that $R = (\gamma M^2)^{-1}$. The Jacobian matrices, $\mathbf{A}_t$ and $\mathbf{A}_x$ can be rewritten as

$$\mathbf{A}_t = \begin{bmatrix} \frac{\gamma M^2}{T} & 0 & -\frac{P}{RT^2} \\ \frac{\gamma M^2 u}{T} & \frac{P}{RT} & -\frac{Pu}{RT^2} \\ M^2 + \frac{M^4(\gamma-1)u}{2T} & \frac{\gamma M^4(\gamma-1)P}{2T} & -\frac{\gamma M^4(\gamma-1)Pu}{2T^2} \end{bmatrix}. \tag{4.15}$$

and

$$\mathbf{A}_x = \begin{bmatrix} \frac{\gamma M^2 u}{T} & \frac{P}{RT} & -\frac{Pu}{RT^2} \\ \frac{\gamma M^2 u^2}{T} + 1 & \frac{2Pu}{RT} & -\frac{Pu^2}{RT^2} \\ \gamma M^2 u + \frac{\gamma M^4 (\gamma-1) u^2}{2T} & \frac{P}{R} + M^4(\gamma - 1)\frac{Pu}{T} & -\frac{\gamma M^4 (\gamma-1) P u^2}{2T^2} \end{bmatrix}. \tag{4.16}$$

The first column of the Jacobian matrices contains the coefficients associated with pressure. In $\mathbf{A}_t$, these coefficients go to zero as the Mach number goes to zero. This results in the acoustic time scale restriction associated with pressure. As the Mach number approaches zero, a vanishingly small time step is needed to keep the coefficients multiplying pressure finite. More importantly, the system of equations become singular as the Mach number goes to zero, as will be discussed below. For a finite time step, the time derivative of pressure will vanish from the equations. In the limit as the Mach number approaches zero, the equations reduce to their incompressible form. Since the time derivative of pressure does not appear in this form of the incompressible equations, the equations contain no pressure history. There is a more mathematical way of looking at this problem. The system can be rewritten in the form

$$\frac{\partial w}{\partial t} + \mathbf{A}_t{}^{-1}\mathbf{A}_x \frac{\partial w}{\partial x} = \mathbf{A}_t{}^{-1}\frac{\partial G_v(w)}{\partial x}. \tag{4.17}$$

As Mach number, M, becomes small, $\mathbf{A}_t$ becomes ill-conditioned, i.e., the determinant of $\mathbf{A}_t$ becomes small and errors due to round off error will become large when computing $\mathbf{A}_t{}^{-1}$. In the limit as M goes to zero, $\mathbf{A}_t{}^{-1}$ is unbounded and the system is singular. The result is often slow convergence due to this stiffness when using a compressible code to compute a flow at very low Mach number. In addition, the eigenvalues($U + C$, $U - C$, and U)of the Jacobian matrix become farther apart as the Mach number goes to zero. A remedy for this is write the equations in the form

$$\mathbf{A}_p \frac{\partial w}{\partial \tau} + \mathbf{A}_t \frac{\partial w}{\partial t} + \mathbf{A}_x \frac{\partial w}{\partial x} = \frac{\partial G(w)}{\partial x}. \tag{4.18}$$

The preconditioning Jacobian matrix, $\mathbf{A}_p$, is defined as

$$\mathbf{A}_p = \begin{bmatrix} \frac{1}{T} & 0 & -\frac{P}{RT^2} \\ \frac{u}{T} & \frac{P}{RT} & -\frac{Pu}{RT^2} \\ \frac{1}{\gamma} + \frac{M^2(\gamma-1)u}{2\gamma T} & \frac{\gamma M^4(\gamma-1)P}{2T} & -\frac{\gamma M^4(\gamma-1)Pu}{2T^2} \end{bmatrix}. \tag{4.19}$$

This Jacobian matrix used in the preconditioning is of the same form as the matrix $\mathbf{A}_t$, but the dependence of Mach number is removed from the terms that are causing the ill-conditioning. This essentially attempts to cluster the eigenvalues around the convective speed. It may be possible to simplify $\mathbf{A}_p$ by setting some of the nondiagonal terms in columns two and three equal to zero.

The definition of the vector w, the Jacobian matrices $\mathbf{A}_t$ and $\mathbf{A}_x$, and the viscous flux vector remain unchanged. The parameter τ is called the pseudo time.

For a time dependent calculation at low Mach number, the preconditioned equations are advanced in the pseudo time frame as well as the real time frame. At each physical time step, the equations are iterated to convergence in pseudo time. At this point the pseudo time term vanishes and the time dependent Navier-Stokes equations are satisfied. The pseudo time iterations also remove the linearization error from the solution at each physical time level. If the low Mach number flow computation is steady, it is only necessary to integrate the equations in the pseudo time frame. This is done by setting the physical time step to a very large number to remove the effect of the physical time derivative from the preconditioned equations.

At higher Mach numbers the pseudo time term is not needed; although convergence does not appear to deteriorate with its continued use.

Pseudo time terms for the full two-dimensional equations are added to the diagonal blocks of the sparse matrix, **A**, in Eq. (4.10). These terms are formed by a direct extension of the one-dimensional example above.

4.3 Artificial Dissipation

Artificial dissipation was needed in the current implementation of the flow equations to prevent the odd-even decoupling seen in central difference computer flow codes. Two schemes were used in this work. The first scheme was based on the research of Jameson and Mavriplis [12] and was used specifically for inviscid subsonic and transonic test cases. The second version was developed to be used with the low Mach number test cases where preconditioning was employed. Both dissipation schemes were added explicitly to the system of flow equations.

The first dissipation scheme adds the dissipation based on the conserved variables, $\rho, \rho u, \rho v$, and e. The dissipation must be constructed such that it does not effect the accuracy of the numerical scheme in smooth regions of the flow. This can be done by adding a biharmonic operator to the original scheme in a conservative form. This was done to insure that no mass, momentum, or energy was added to the flow field in the sum over the entire flow domain. The flow field could also contain some large pressure gradients as in the region near a shock. Here the biharmonic operator would tend to smear out these discontinuities. A pressure switch was used to detect the location of the shock and the biharmonic operator was turned off. Instead, a Laplacian operator was used near discontinuities to change the overall scheme to be locally first order. The Laplacian and biharmonic operators used in this code correspond to the second and fourth differences added to centrally differenced structured grid codes. Since the main code was written in delta form based on the flow primitives, it was necessary to compute the local conserved variables. The undivided first difference of the variables were then computed on the edges of the triangle and then summed over its three edges to obtain the second difference for the cell. This can be represented as

$$a_{i2} = \sum_{j=1}^{3} \left[U_j\right] - 3U_i, \tag{4.20}$$

where U was the vector of conserved variables. The summation is over the index j which represents the neighboring cell numbers. The index i is the cell number associated with the newly constructed second difference. So the right hand side of Eq. (4.20) shows the computation of the first difference on the edges as well as the summation to give the second difference for cell i. The pressure switch was constructed in the same way except that it was normalized by the average of the three surrounding cells and cell i.

Next the third difference was computed on the edges of all triangle cells. This was then used in conjunction with the first difference and some weighting functions to construct the appropriate combinations of Laplacian and biharmonic dissipation operators,

$$a_{i14} = visc_1 \frac{A_{ij}}{\Delta t_{ij}} \left[\sum_{j=1}^{3} \left[U_j\right] - 3U_i \right] - visc_3 \frac{A_{ij}}{\Delta t_{ij}} \left[\sum_{j=1}^{3} \left[a_{j2}\right] - 3a_{i2} \right]. \qquad (4.21)$$

The fraction $\frac{A_{ij}}{\Delta t_{ij}}$ represents the average of the largest local eigenvalue. The quantity $visc_1$ contains the normalized pressure switch and a constant coefficient of 0.5. It has been found that the use of the biharmonic operator in the presence of a discontinuity was destabilizing, so the weighting function

$$visc_3 = C_3 max(0, visc_3 - visc_1)$$

was used. Here the value of $visc_3$ takes on the value of 0.0 when the weighting coefficient $visc_1$ becomes large. The constant 0.03125 was used for the quantity C_3.

This term was then included explicitly on the right-hand-side of the system of equations. This type of artificial dissipation was not new and has been used extensively by other researchers.

A second type of dissipation was developed to be used with the temporally preconditioned scheme. This form of the dissipation was used for the viscous subsonic flows computed in the present work. Here a biharmonic operator was used on the primitive flow variables. So similar to Eq. (4.20) the second difference of the primitives were computed as

$$ap_{i2} = \sum_{j=1}^{3} \left[w_j\right] - 3w_i,$$

where w was the vector of primitive variables. Again the summation was done over the index j. The result represents the second difference of the variables in cell i. The fourth difference is then computed by summing the third difference over the edges of the cell,

$$ap_{i4} = \sum_{j=1}^{3} \left[ap_{j2}\right] - 3ap_{i2}.$$

The resulting fourth difference was then premultiplied by the preconditioning matrix $\mathbf{A}_t$ shown in Eq. (4.13). Then it was multiplied by the appropriate coefficient to make it consistent with the other terms and included explicitly on the right-hand-side of the system of equations.

4.4 Boundary Conditions

To solve the Navier-Stokes equations on a given computational domain it is necessary to impose the appropriate boundary conditions. The specified numerical boundary conditions will depend on the physics of the real flow that is being predicted. The inlet and exit boundary conditions will be discussed from a characteristics point of view. The eigenvalues of the spatial Jacobian matrix are used to determine how the boundary conditions are imposed at an inlet or exit. Without preconditioning, the eigenvalues of the Jacobian matrix associated with the derivative in the x direction are U, U, $U + C$, and $U - C$, where U is the normal velocity component to the boundary surface, and C is the speed of sound. These eigenvalues are modified when preconditioning is applied to the system. This will be discussed later in this section. Figure 4.3 shows the subsonic inlet and exit boundaries and their respective characteristics. At the inlet, for subsonic flow where C is larger than U, the three characteristics U, U, and $U + C$ come from upstream while the characteristic $U - C$ comes from downstream. This gives the inlet boundary condition that three quantities must be specified, and one quantity extrapolated from the interior of the domain. For the subsonic exit, the same characteristics point at the boundary. To impose this

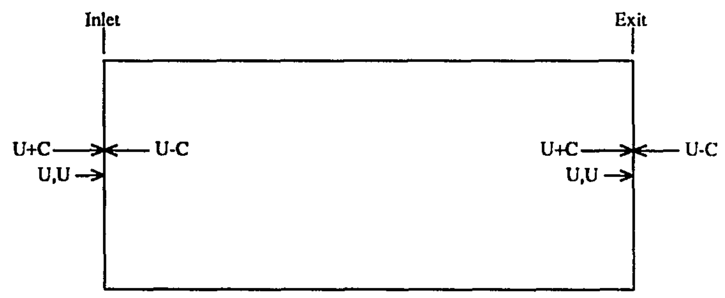


Figure 4.3: Subsonic inlet and exit boundary conditions

boundary condition, three quantities are extrapolated from upstream. i.e. from inside the computational domain; and one must be specified from outside the computational domain. For the specification of supersonic boundary conditions where C is smaller than U, the four characteristics come from upstream. Here it is necessary to specify four quantities at the inlet. At the exit, the four quantities are extrapolated to the exit. Figure 4.4 shows the supersonic inlet and exit boundaries and their respective characteristics. For a cell centered unstructured grid approach this means locating a boundary cell and specifying the necessary flow quantities across its edge in a ghost cell to give the correct conditions at a boundary.

For the subsonic viscous flow cases in this study, the inlet boundary conditions were imposed by specifying the velocity components, u and v, as well as the static temperature, T. The static pressure was extrapolated from the interior to the inlet. At the exit, u, v, and T were extrapolated downstream. The static pressure, P, was specified. Velocity components were specified in the level 2 cells such that there was

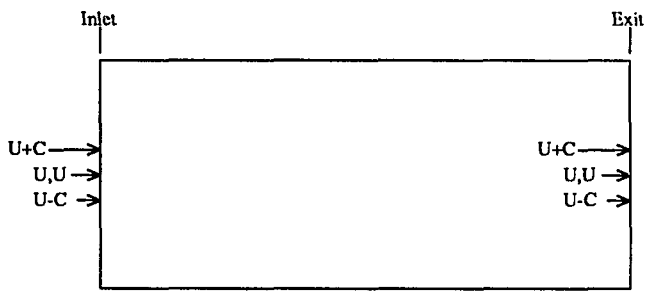


Figure 4.4: Supersonic inlet and exit boundary conditions

no change at the inlet boundary of the computational domain. At the solid wall the static temperature was specified. The u and v velocity components were set to zero to enforce the no-slip condition. The implementation of the no-slip boundary condition is shown in Fig. 4.5. The Cartesian velocity components were specified in cell B such that the average value at face $a - b$ was zero. Static pressure was specified as symmetric in the ghost cell to give a zero normal derivative at the solid surface. The velocities were specified in the level 2 ghost cells antisymmetrically such that the no-slip boundary condition was enforced at the wall. The viscous fluxes were then computed as usual. Symmetry and periodic boundary conditions were imposed by simply specifying the appropriate cell connectivity. At a symmetry boundary cell, values were reflected across the boundary, Fig. 4.6. At a periodic boundary cell, values were transposed by the periodic pitch of the computational domain. Fig. 4.7.

For supersonic viscous flow, boundary conditions at a solid wall remained the same as in subsonic flow. At the inlet, all flow quantities. P. u. v. and T were

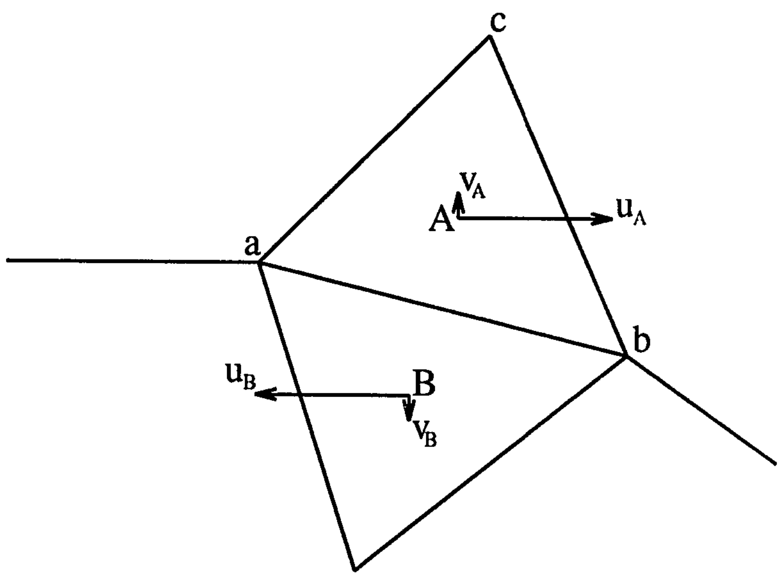


Figure 4.5: Solid wall viscous no-slip boundary condition

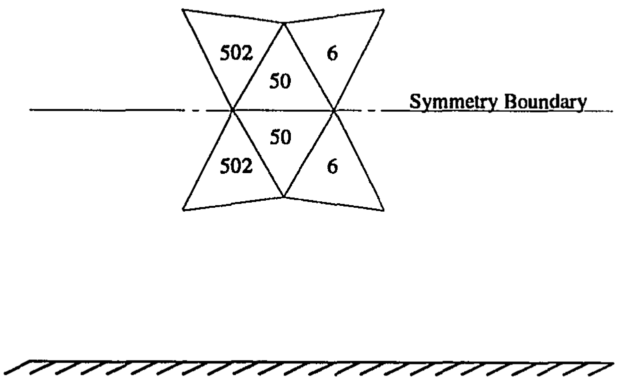


Figure 4.6: Symmetric boundary condition

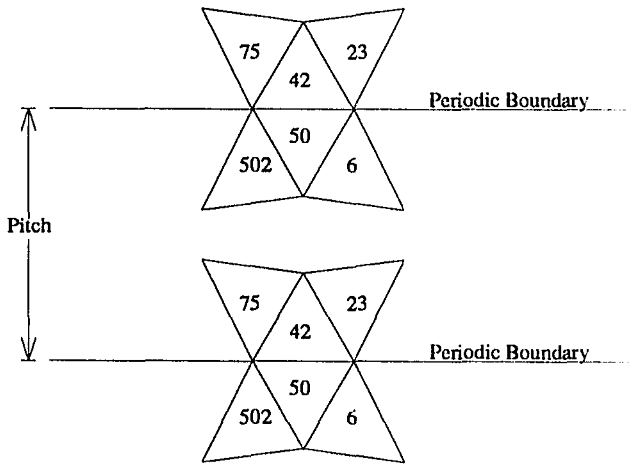


Figure 4.7: Periodic boundary condition

specified. At the exit all the flow quantities were extrapolated.

It is important to note that one of the test cases computed to verify the code was inviscid. For this, the viscous boundary conditions described above were modified appropriately. Since the inflow and outflow were subsonic, it was only necessary to modify the no-slip solid wall boundary condition to a tangency boundary condition. With the present cell centered scheme, this was done by reflecting the u velocity component about the solid wall symmetrically and reflecting the v velocity component antisymmetrically. An example of the enforcement of the wall tangency boundary condition can be seen in Fig. 4.8. The normal and tangential velocity components of cell A, V_{N_A} and V_{T_A} respectively, were computed as

$$V_{N_A} = \frac{u_A \Delta x_{ab} + v_A \Delta y_{ab}}{\sqrt{\Delta x_{ab}{}^2 + \Delta y_{ab}^2}}$$

and

$$V_{T_A} = \frac{-u_A \Delta y_{ab} + v_A \Delta x_{ab}}{\sqrt{\Delta x_{ab}{}^2 + \Delta y_{ab}^2}}$$

where u_A and v_A represent the Cartesian velocity components of the resultant velocity vector V_A shown in Fig. 4.8 associated with cell A. The quantities Δx_{ab} and Δy_{ab} are the geometric differences along side $a - b$ of cell A. The corresponding normal and tangential velocity components were set in cell B as

$$V_{N_B} = -V_{N_A}$$

and

$$V_{T_B} = V_{T_A}.$$

The actual Cartesian velocity components in cell B were then computed as

$$u_B = \frac{V_{T_B}\Delta x_{ab} - V_{N_B}\Delta y_{ab}}{\sqrt{\Delta x_{ab}{}^2 + \Delta y_{ab}^2}}$$

and

$$v_B = \frac{V_{T_B}\Delta y_{ab} + V_{N_B}\Delta x_{ab}}{\sqrt{\Delta x_{ab}{}^2 + \Delta y_{ab}^2}}.$$

These velocities were then used in constructing the matrix equation and enforced the wall tangency condition. Cell B does not exist in memory at the boundary. It is only presented here for clarity. The contributions of cell B were included when computing of the coefficients for cell A. Similarly, the viscous flux contributions in the level 2 ghost cells were included in their complimentary cells that reside in the flow domain.

In the current work, temporal preconditioning was used to compute low Mach number flows as described in the previous section. The preconditioning did not affect how the boundary conditions were specified, but it is interesting to note that the eigenvalues of the system were modified. The eigenvalues must now be obtained from the matrix that results from the product $\mathbf{A}_p{}^{-1}\mathbf{A}_x$. In one dimension

$$\mathbf{A}_p{}^{-1} = \begin{bmatrix} 0 & -\frac{\gamma M^4(\gamma-1)P}{2HT} & \frac{P}{HRT} \\ -\frac{uRT}{P} & \frac{RT}{P} & 0 \\ -\frac{RT^2}{P} & -\frac{\gamma M^4(\gamma-1)P}{2H} & \frac{1}{H} \end{bmatrix}$$

where

$$H = -\frac{\gamma M^4(\gamma-1)Pu}{2T^2} + \frac{M^2(\gamma-1)Pu}{2\gamma RT^2} + \frac{1}{\gamma RT}.$$

The eigenvalues for the preconditioned system can be computed by solving either

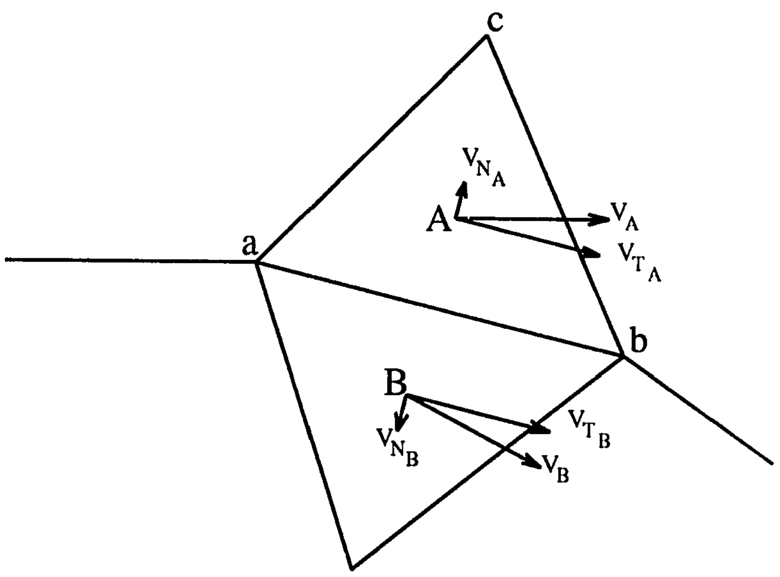


Figure 4.8: Solid wall inviscid tangency boundary condition

$$| \mathbf{A}_p{}^{-1}\mathbf{A}_x - \lambda \mathbf{I} | = 0$$

or

$$| \mathbf{A}_x - \lambda \mathbf{A}_p | = 0$$

for λ. One eigenvalue remains unchanged that is $\lambda_1 = U$. The other two eigenvalues $\lambda_{2,3}$ for the given one-dimensional system take on a similar but yet a more complex form than that described by Withington et al. [31]. However, in the limit as Mach number approaches zero, the eigenvalues of the system are

$$\lambda_1 = U$$

and

$$\lambda_{2,3} = \frac{U \pm \sqrt{U^2 + 4\gamma T}}{2}.$$

By substituting in the appropriate nondimensional quantities the resulting ratio of largest to smallest eigenvalue takes on the value of $(1 + \sqrt{5})/(1 - \sqrt{5})$. This is the same quantity that was obtained by Withington et al. [31] with the ratio of specific heats set equal to one in the present work. The preconditioning essentially allows all of the equations of the system to be integrated at the same pseudo-time rate. This can be compared with a scheme without preconditioning where the ratio of largest to smallest eigenvalues is infinity.

4.5 Sparse Matrix Solvers

The system of algebraic equations being solved in the present implicit unstructured grid formulation is represented by Eq. (4.10). The matrix **A** is sparse. A sparse matrix is one in which most elements are zero. Also, there is usually no particular pattern to the nonzero elements when the matrix arises from an unstructured grid formulation. However, the blocks on the main diagonal of this sparse matrix always have some nonzero entries. Figure 4.9 shows a representative form of the sparse matrix **A**. The solid squares represent 4×4 blocks with at least the diagonal elements of the block being nonzero. The remaining blocks contain zeros. In the present method, a two-dimensional viscous flow computation requires a maximum of ten 4×4 blocks in each row of the **A** matrix. The block locations in a given row represent the connectivity matrix for the level 1 and level 2 cells that surround the cell which requires the solution. The block matrix that is identified with this cell is the diagonal element for that row. A given row of the **A** matrix consists of six blocks which contain the convective flux information, while viscous flux information can be contained in all ten blocks. Some rows of the **A** matrix may contain fewer blocks since boundary condition information can be included only through blocks that represent cells that reside inside the computational domain. Recall that ghost cells were not included explicitly as part of the solution. In contrast, an implicit structured solver can be written such that the resulting **A** matrix on the left-hand side has some special structure that allows the matrix equation to be solved by some well established methods. The equations can often be cast in a form that results in a block bidiagonal or block tridiagonal matrix. This structure is not generally available to the solution of the flow equations written for an unstructured grid. The method for solving the sparse matrix equation that results from the unstructured formulation can be direct or iterative. The direct method is usually not chosen since it requires a large computational effort compared with most iterative methods. In addition, if the size of **A** is large, a solution is difficult to obtain by a direct method due to roundoff errors. Specialized direct solvers have been developed which take advantage of the sparseness of the matrix **A**. The Yale Sparse Matrix package [34] is an example of this type of technology.

Several iterative methods were examined in this work. The first was a point Gauss-Seidel scheme where only the diagonal elements of the diagonal blocks of matrix **A** were retained on the left-hand side as unknowns. This scheme was successful for many of the simpler problems but was prone to divergence when starting with poor initial conditions. It seemed to be very sensitive to lack of diagonal dominance.

Another iterative scheme used was the point block Gauss-Seidel method. Here the diagonal 4×4 blocks of matrix **A** were retained on the left hand side. The remaining matrix equation was solved using **LU** decomposition. The **L** and **U** matrices refer to the lower and upper triangular decomposition of the diagonal block of the matrix **A**. This was found to be more robust than the previous scheme.

Even though the full sparse block matrix is $N \times N$, it is only necessary to store the nonzero blocks. This gives a maximum block matrix of $10 \times N$ for a viscous code. However, the bandwidth could still be the maximum, N.

Since the grid is unstructured, the boundary conditions can be scattered throughout the entire matrix. They are not clustered at the top or bottom of the matrix as in structured codes.

The commercially available sparse iterative solver, SITRSOL [35], which resides on the Cray YMP as a callable subroutine was also evaluated for solving the above matrix equation. SITRSOL takes advantage of the matrix sparseness by only storing the nonzero entries. The package makes available to the user several iterative meth-

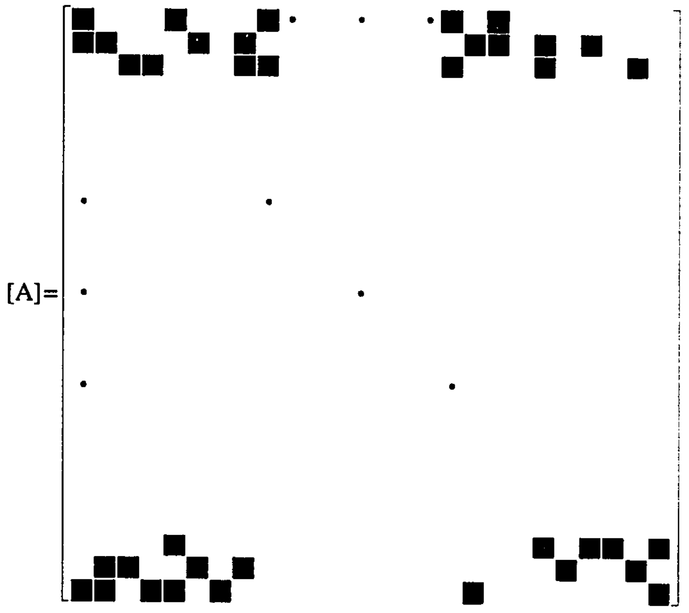


Figure 4.9: Form of sparse matrix **A**

ods as well as preconditioners for solving non-symmetric positive indefinite sparse linear systems. In the present work the bi-conjugate gradient method, the generalized minimal residual method, and the generalized conjugate residual method were considered. An incomplete **LU** preconditioner was also used. These three iterative methods are of the preconditioned conjugate gradient type. A general description of the conjugate gradient method with and without a preconditioner will be given below based on the works of Golub and Van Loan [36], Saad and Schultz [37], and Press et al. [38].

The conjugate gradient method is a technique that minimizes a function along vector paths that are linearly independent. These vector paths are called conjugate directions. The method is based on the method of steepest descent used in problems of optimization where a function needs to be either minimized or maximized. A function is required that, when minimized, results in the same solution as the matrix equation $\mathbf{A}\vec{x} = \vec{b}$. Here $\mathbf{A}$ is assumed to be symmetric positive definite. The minimization of the function

$$\phi(\vec{x}) = \frac{1}{2}\vec{x}^T\mathbf{A}\vec{x} - \vec{x}^T\vec{b} \tag{4.22}$$

is the equation

$$\mathbf{A}\vec{x} = \vec{b}$$

where $\vec{x}^T$ is defined as the transpose of $\vec{x}$. The gradient at a point $\vec{x}_p$ is defined as

$$\nabla\phi(\vec{x}_p) = \mathbf{A}\vec{x}_p - \vec{b}.$$

The function ϕ decreases fastest in the $-\nabla\phi$ direction. At a point p this defines the residual

$$\vec{r}_p = \vec{b} - \mathbf{A}\vec{x}_p.$$

A new direction is now taken orthogonal to the previous residual direction such that

$$\phi(\vec{x}_p + \alpha\vec{r}_p) < \phi(\vec{x}_p).$$

An α is chosen such that $\phi(\vec{x}_p + \alpha\vec{r}_p)$ is minimized.

$$\frac{\partial\phi}{\partial\alpha} = \vec{r}_p^{\,T}\mathbf{A}\vec{x}_p + \alpha\vec{r}_p^{\,T}\mathbf{A}\vec{r}_p - \vec{r}_p^{\,T}\vec{b}$$

gives the value

$$\alpha = \frac{\vec{r}_p^{\,T}\vec{r}_p}{\vec{r}_p^{\,T}\mathbf{A}\vec{r}_p}$$

which minimizes the function ϕ along the path of the residual $\vec{r}$. The steepest descent method becomes very inefficient for functions that are elliptic in shape with a high aspect ratio.

A more efficient method can be obtained by making the search directions more general. Again the function in Eq.(4.22) is minimized. Here the minimization will be required along the general search direction $\vec{z}$. So the function to be minimized is

$$\phi(\vec{x} + \alpha\vec{z}) = \frac{1}{2}(\vec{x} + \alpha\vec{z})^T\mathbf{A}(\vec{x} + \alpha\vec{z}) - (\vec{x} + \alpha\vec{z})^T\vec{b}$$

resulting in the quantity

$$\alpha_k = \frac{\vec{z}_k^{\,T}\vec{r}_{k-1}}{\vec{z}_k^{\,T}\mathbf{A}\vec{z}_k}.$$

The subscript k refers to the current vector path. An appropriate vector $\vec{z}$ must now be chosen.

The conjugate gradient method is used to solve the matrix equation $\mathbf{A}\vec{x} = \vec{b}$ by choosing the $\vec{z}_k$ vector such that it is close to the $k-1$ residual vector and is conjugate to all previous vectors, $\vec{z}_1, \vec{z}_2, ...\vec{z}_{k-1}$. The $\vec{z}_k$ vector is obtained by solving a least squares problem. The result is that

$$\vec{z}_k = \vec{r}_{k-1} + \beta_k \vec{z}_{k-1}$$

where

$$\beta_k = -\frac{\vec{z}^T_{k-1}\mathbf{A}\vec{r}_{k-1}}{\vec{z}^T_{k-1}\mathbf{A}\vec{z}_{k-1}}.$$

This is used to iteratively determine the solution $\vec{x}$ by

$$\vec{x}_k = \vec{x}_{k-1} + \alpha_k \vec{z}_k$$

where

$$\alpha_k = \frac{\vec{r}^T_{k-1}\vec{r}_{k-1}}{\vec{z}^T_k \mathbf{A}\vec{z}_k}.$$

This is the essential algorithm for the conjugate gradient method.

The conjugate gradient method was developed for a symmetric positive definite matrix. The matrix resulting from the discretized flow equations, in general, does not necessarily possess either of these properties. However, a matrix can be made symmetric positive definite by simply multiplying by its transpose. So this changes the problem from

$$\mathbf{A}\vec{x} = \vec{b}$$

to

$$\mathbf{A}^T\mathbf{A}\vec{x} = \mathbf{A}^T\vec{b}.$$

The multiplication $\mathbf{A}^T\mathbf{A}$ can be computationally very expensive when $\mathbf{A}$ is large. A consequence of this multiplication is that the condition number of the original matrix is squared. This results in a considerably slower convergence rate.

The sparse matrix solvers from SITRSOL used in the present work were conjugate gradient like methods. These methods relax the need for the matrix $\mathbf{A}$ to be symmetric positive definite. Differences in these methods were described in Saad and Schultz [37] and Wigton et al. [39]. The generalized minimal residual method seemed to be the most efficient method when the requirements of storage and operation count in reaching a solution were considered.

A preconditioning matrix can be used to accelerate the solution convergence rate of the conjugate gradient method described above. In general, a preconditioning matrix $\mathbf{P}$ should approximate $\mathbf{A}^{-1}$. Basically $\mathbf{P}$ should drive the condition number of the product $\mathbf{PA}$ toward the ideal value of one. This clusters the eigenvalues of the product, and results in the preconditioned matrix equation

$$\mathbf{PA}\vec{x} = \mathbf{P}\vec{b}.$$

Several preconditioners are available to the user of SITRSOL. In the current work the incomplete **LU** preconditioner was found to be the most effective. The **L** and **U** refer to lower and upper triangular matrices respectively. These are incomplete in the .ense that they do not represent the true **LU** decomposition of the matrix **A**. The form of these matrices retain the sparseness of the original **A** matrix.

The iterative solver SITRSOL was used on one of the test cases to be shown in the results section and its effectiveness was compared with that of the point block Gauss-Seidel method. The conclusions shown in this work are provisional. More experience needs to be obtained to make a true evaluation of the various solvers and preconditioners.

The solution of Eq. (4.10) using a point block Gauss-Seidel method suffers from recurrence. The penalty is seen in vectorization. This recurrence can be eliminated with a minimum effect on the solution convergence rate by using a coloring scheme. The idea comes from a problem which arose in graph theory. A theorem states that a map can be colored with only four colors such that no two regions of the same color share a border. The conjecture was proven through exhaustive computation by Appel and Haken [33] in 1976. This theorem was implemented by first coloring the unstructured grid according to the theorem and storing all cell numbers of given color in an integer array. The scheme was most efficient when the number of cells in each color integer array was about equal.

The actual coloring in the present application was done exhaustively. First in the order of blue, green, red, and yellow every cell was visited with a level 2 restriction on neighboring cells of the same color. The grid cells were initially set as uncolored. A cell was colored blue if both the level 1 and level 2 cells surrounding that cell were all uncolored. If a cell was already colored blue at level 2 or level 1. that particular cell was left uncolored and the search continued until all cells in the computational domain were visited. Then the cells were again queried for the color green. Again a cell was only colored green if the level 1 and level 2 cells did not contain a green cell. However a neighboring blue cell was acceptable. This search continued through all four colors. Then the color order was reversed and a level 1 restriction was placed on the cell color. Here only the level 1 cells were checked for a like color. The cell numbers of the same color were then stored in four integer arrays. The effect of the color reversal was to equalize the length of the arrays for vectorization. Each cell was then checked to make certain that it did not border a cell with the same color. This gave a color map that was then used as input to the flow code. The Gauss-Seidel algorithm was then written to contain four loops corresponding to the four colors of the colored grid. Each single colored loop contained no level 1 cell recurrence, so it was vectorized. On a typical problem in the present study, the solution time of the algebraic system(the Gauss-Seidel subroutine) was reduced by a factor of 7.6 times by using this four color partitioning. Recurrence is still present but only through the level 2 cells, illustrated in Fig. 4.1, required in the viscous terms. The result is that the quantities in the level 2 cells are lagged from the previous iteration time step. However, this does not seem to effect the convergence rate.

5. RESULTS

The results presented in this chapter will be used to demonstrate two conclusions. The initial results will show the validity of the code. And later results will indicate the versatility of the unstructured grid over the structured grid formulation. Comparisons will be made with data available from other investigators.

5.1 Bump on Wall

Inviscid flow over a bump in a channel was computed at two values of Mach number. A Mach number of 0.5 was used for the first test case. Figure 5.1 shows Mach number contours. The flow was subsonic so the inviscid flow was symmetric about the middle of the bump. This could be seen more clearly when the upper and lower wall Mach number distributions were plotted. The results of this subsonic case compared well with those reported by Ni, [40].

A second test case was computed at a Mach number of 0.675 at the inlet. Here the flow was transonic over the bump. The grid used for this test case can be seen in Fig. 5.3. A supersonic bubble was formed on the bump. Fig. 5.4. The location of the shock was shown clearly in the plot of upper and lower wall Mach number distributions, Fig. 5.5. The location of the shock compared well with the results of Ni, [40] as well as that of Chima et al, [41]. The sonic line that impinged on the aft

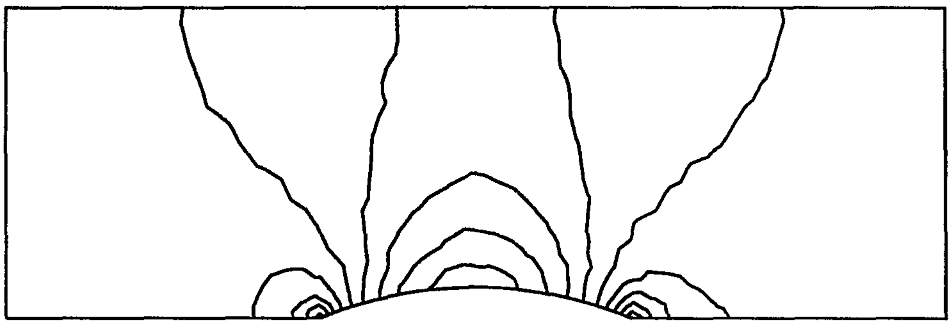


Figure 5.1: Constant Mach number contours for flow over a symmetrical bump in a channel, $M_{in} = 0.5$

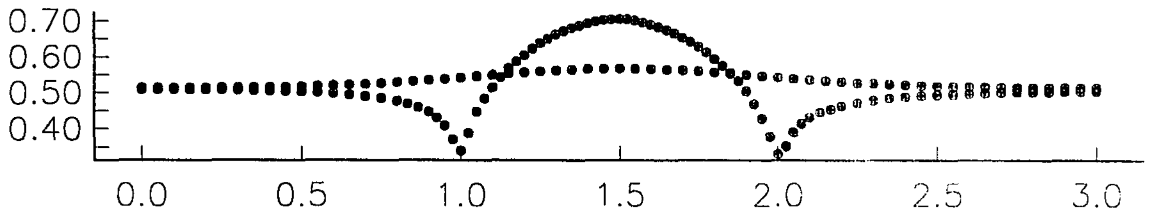


Figure 5.2: Upper and lower wall Mach number distribution

side of the bump was at a distance of 72 percent of the chord length from the head of the bump in the above cases. The present case locates the sonic line at 73 percent of the chord length.

These inviscid test cases required the addition of artificial dissipation. For the subsonic case, a fourth difference was added to prevent the odd-even decoupling of the solution seen in central difference schemes. The transonic case also required the additional second difference to prevent oscillations from occurring about the discontinuity. In both cases the dissipation model was similar to that of Jameson et al. [3]. Later Jameson and Mavriplis [12] implemented this type of dissipation model for an explicit unstructured grid flow solver.

5.2 Developing Channel Flow

Developing flow in a channel was used to validate the code for viscous flows. It also served the purpose of testing the preconditioning used for computing low Mach number flows. Comparisons were made between the Gauss-Seidel method and the solver SITRSOL for solving the sparse matrix equation.

The code was validated on four developing channel flow test cases. A low inlet Mach number flow of 0.05 was used to compute flows at Reynolds numbers of 1, 20, 150, and 1500 based on the inlet uniform velocity, density, and full channel height. Because the inlet Mach number was held constant, the channel height was varied to obtain the appropriate Reynolds number. Unstructured grids of 1114,1969,4800, and 4800 cells were used for the Reynolds number flows of 1, 20, 150, and 1500 respectively. Uniform flow enters the channel with a nondimensional uniform velocity of one and accelerates to a nondimensional centerline velocity of 1.5. In order to compare with

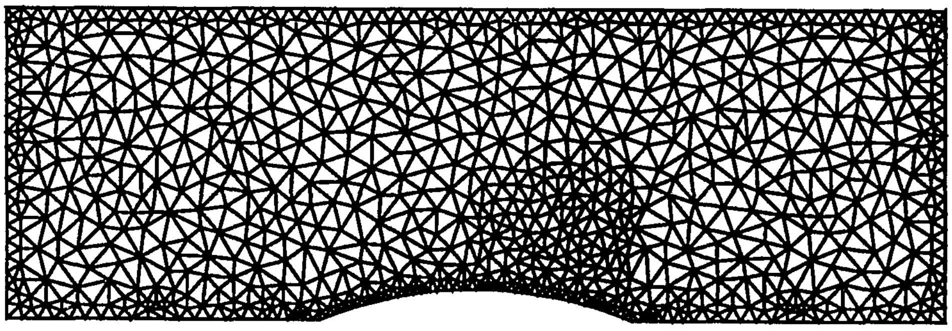


Figure 5.3: Computational grid for the symmetrical bump in a channel test case, $M_{in} = 0.675$

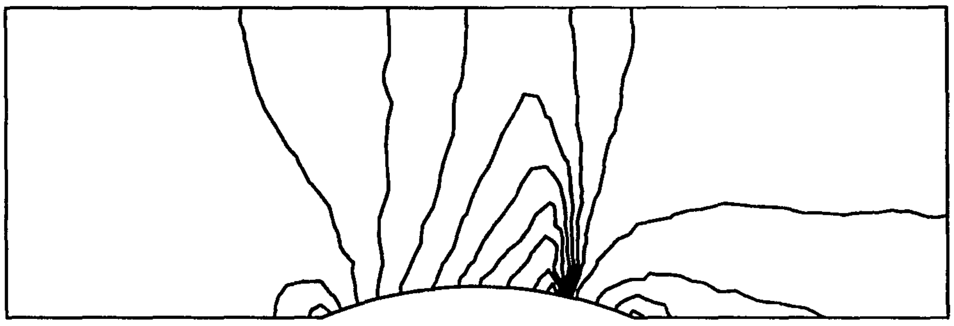


Figure 5.4: Constant Mach number contours for flow over a symmetrical bump in a channel, $M_{in} = 0.675$

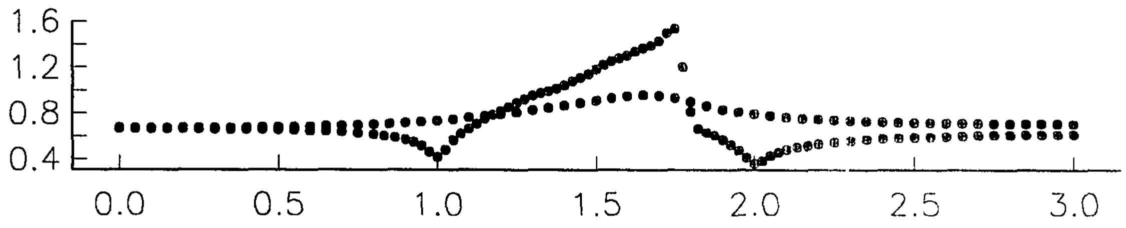


Figure 5.5: Upper and lower wall Mach number distribution

published results for incompressible flows, it was necessary to make a correction to the centerline velocity at the low Reynolds numbers due to the larger density variation from the inlet to the exit of the channel. Figure 5.6 shows the centerline velocity of the channel flow at various Reynolds numbers. These results were compared with other computations by Tenpas and Pletcher [42], Morihara and Cheng [43], and Chilukuri and Pletcher [44]. At a Reynolds number of 20 the centerline velocity of the current study slightly under predicted the centerline velocities obtained by the other investigators near the exit of the channel. At a Reynolds number of 1500, the results of the present study show a more rapid acceleration of the flow than indicated by the solution of the partially parabolized Navier-Stokes equations.

Typical convergence histories for the code are shown in Fig. 5.7. The convergence criteria was based on the residual of the continuity equation in delta form which should approach machine zero as the solution goes to a steady state. The solution of the matrix equation was done by the block Gauss-Seidel method. In general, the solution converged at nearly the same rate over a wide range of Mach numbers holding the Reynolds number equal to 20 for the four flow test cases. This illustrates the benefits of the preconditioning. Without preconditioning, it was necessary to run the code at a much smaller time step thus decreasing the rate of convergence. At Mach numbers lower than 0.1 the code without preconditioning did not converge.

The sparse matrix solver SITRSOL was used for comparison with the Gauss-Seidel method. The convergence history for three different conjugate gradient like methods with the **ILU** used as a preconditioner is shown in Fig. 5.8. The Gauss-Seidel method without the coloring algorithm took 13.5 minutes on the Cray YMP. The same computation with a color map supplied for vectorization of the Gauss-Seidel

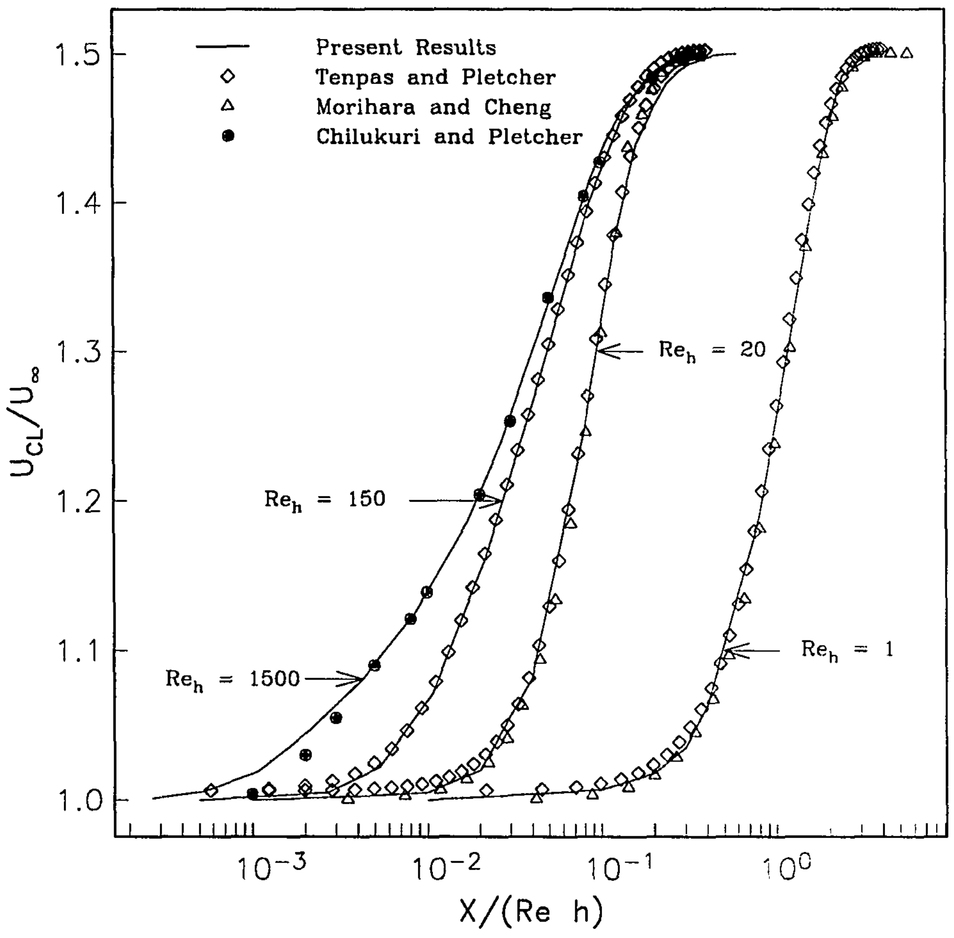


Figure 5.6: Centerline velocity profiles for developing flow in a channel at $M_{in} = 0.05$ with $Re_h = 1, 20, 150, 1500$

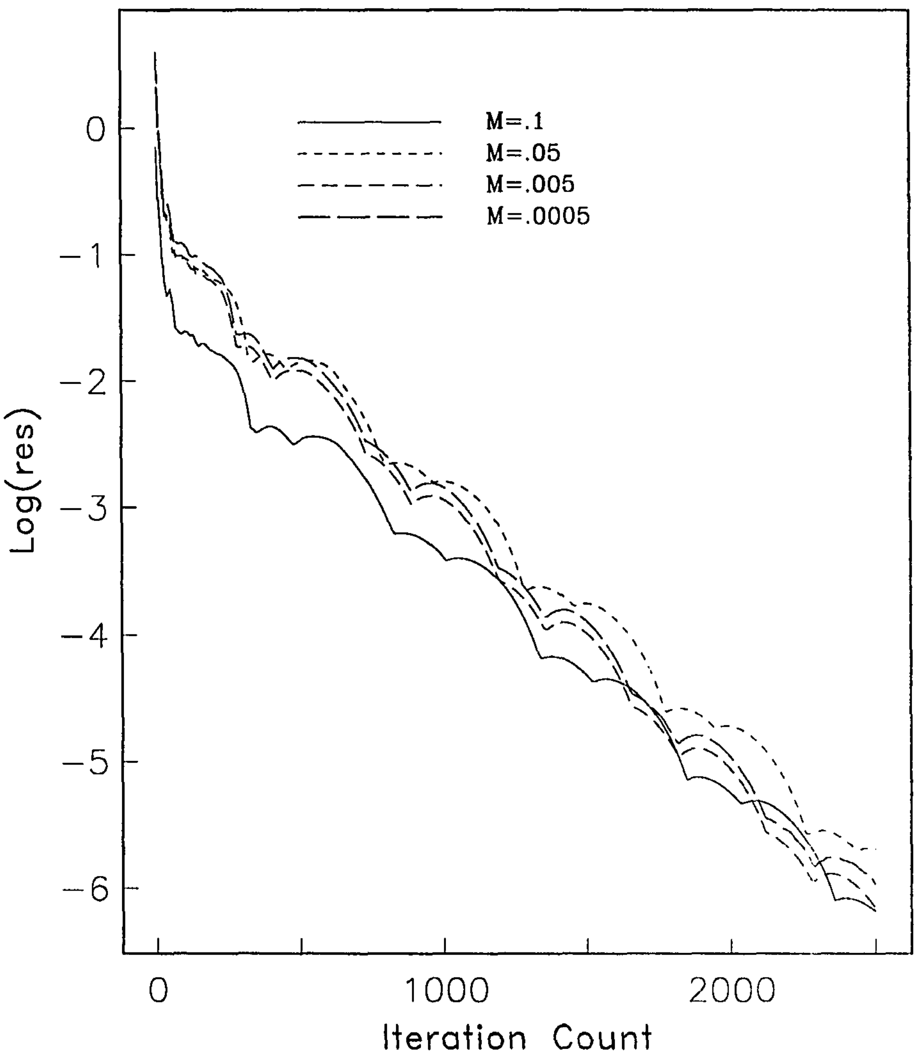


Figure 5.7: Convergence history for developing channel flow over a range of Mach numbers at $Re_h = 20$ using the block Gauss-Seidel solver

algorithm gave a speedup factor of 7.6 over the standard Gauss-Seidel matrix solver. The overall computer time was reduced to 11.4 minutes, or a speed up of 15.5 percent. This suggests that more attention should be given to the vectorization of other parts of the flow code. The coloring scheme did not have much effect on the convergence history of this viscous calculation. The same grid was used to make comparisons with SITRSOL. The bi-conjugate gradient method took 9.3 minutes of computer time to reach about the same level of convergence as the Gauss-Seidel method. The generalized conjugate gradient residual method required 10.23 minutes of computer time. About the same level of convergence was obtained by the generalized minimum residual method in 5.5 minutes.

5.3 Sudden Expansion

The previous test cases could have easily been computed using a structured grid approach. The sudden expansion test case demonstrates the capability of the unstructured grid generation and its ability to obtain a grid in a domain that would otherwise need a patched or masked grid to work for a structured flow code. The results from this computation were compared with the experimental results obtained by Durst et. al. [45]. They noted that though at lower Reynolds numbers the flow was symmetric about the centerline of the expansion, there were three-dimensional effects near the separated regions. A plane symmetric sudden expansion with a downstream channel height to step height ratio of 3:1 was computed. The Reynolds number for this flow was 56 based on the upstream channel height and the centerline upstream velocity. A fully developed parabolic profile was prescribed at the inlet which was located one step height upstream of the expansion. The Reynolds number was com-

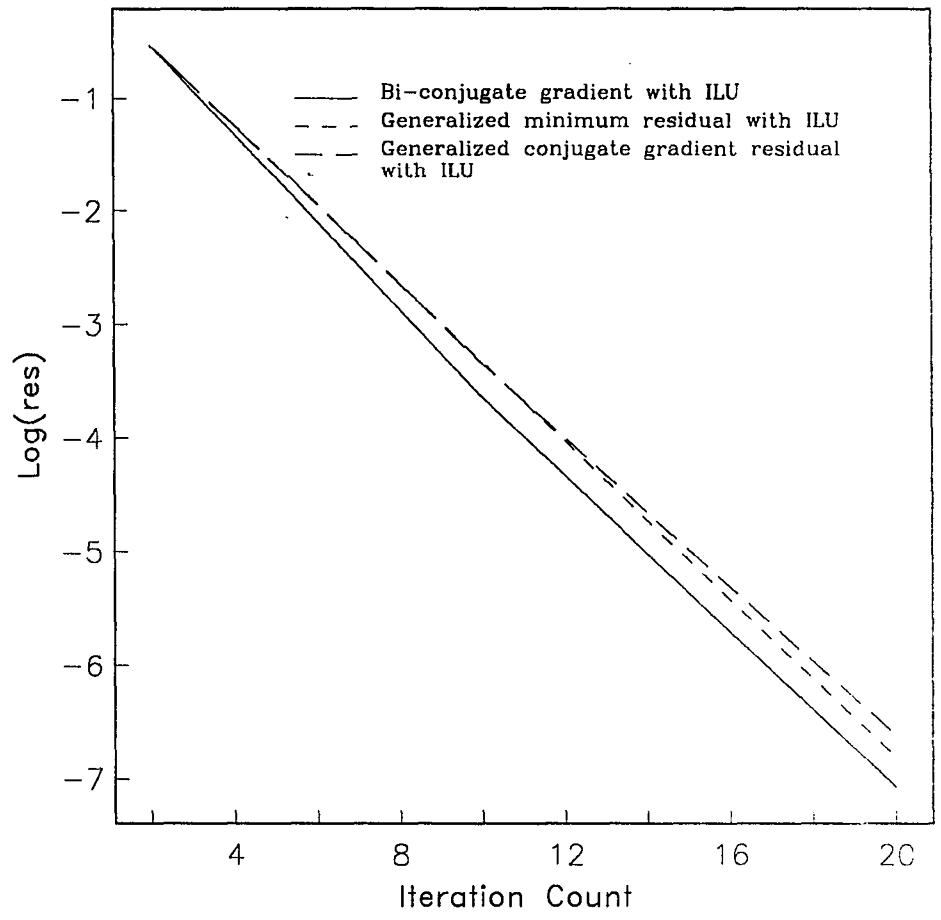


Figure 5.8: Convergence history of developing flow in a channel, $Re_h = 20$ computed with sparse matrix solver

puted at a streamwise location 0.25 step heights upstream of the expansion. It was interesting to note that the profile at this location was already anticipating the expansion corner. The flow near the wall begins to accelerate; and to conserve mass, the centerline velocity decreases. The flow separates at the step, reattaches downstream, and returns to a fully developed profile about ten step heights from the expansion. Figure 5.9 is a velocity vector plot of the recirculating region. The streamwise velocity component at six specific channel locations are shown in Fig. 5.10. Comparisons were made with the laser anemometer experimental data presented by Durst et al. [45]. The centerline velocity distribution was compared with the laser anemometer data and with the viscous-inviscid interaction computational method of Kwon et al. [46] and is shown in Fig. 5.11. The predicted centerline velocity appears larger than the experimental values downstream, but the correct value of one-third the upstream fully developed centerline velocity was obtained in the present calculation.

5.4 Periodic Tandem Circular Cylinders in Cross Flow

The flow was computed over a cascade of tandem circular cylinders. This computation should be of practical interest in that geometries of this sort are encountered when modeling flow through heat exchangers. These tube heat exchangers can be found in automobile radiators, room heaters and gas and air heaters. With the unstructured grid formulation, it was easier to generate a computational grid about in-line as well as staggered cascades of tubes. Some of the geometric quantities that affect the flow characteristics of the heat exchanger are the size and shape of the tubes as well as their vertical and horizontal spacing. This type of parametric study is ideal for the unstructured grid formulation.

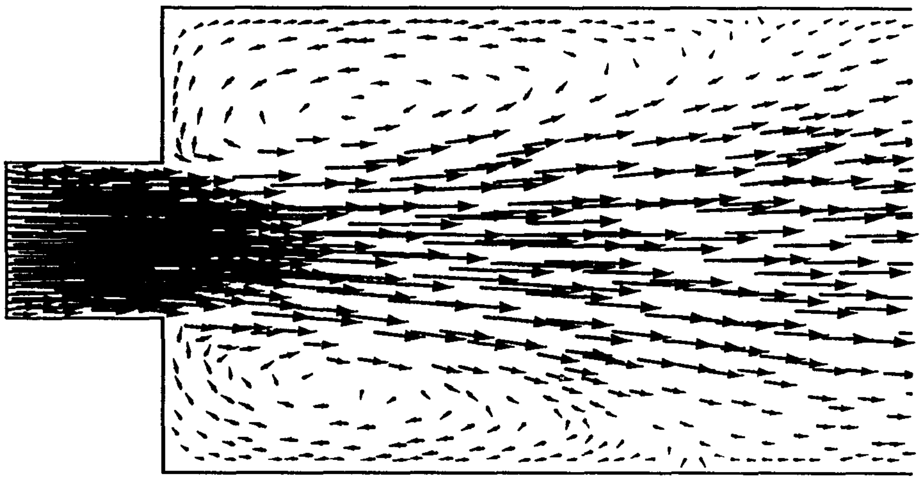


Figure 5.9: Symmetric sudden expansion, $Re_h = 56$

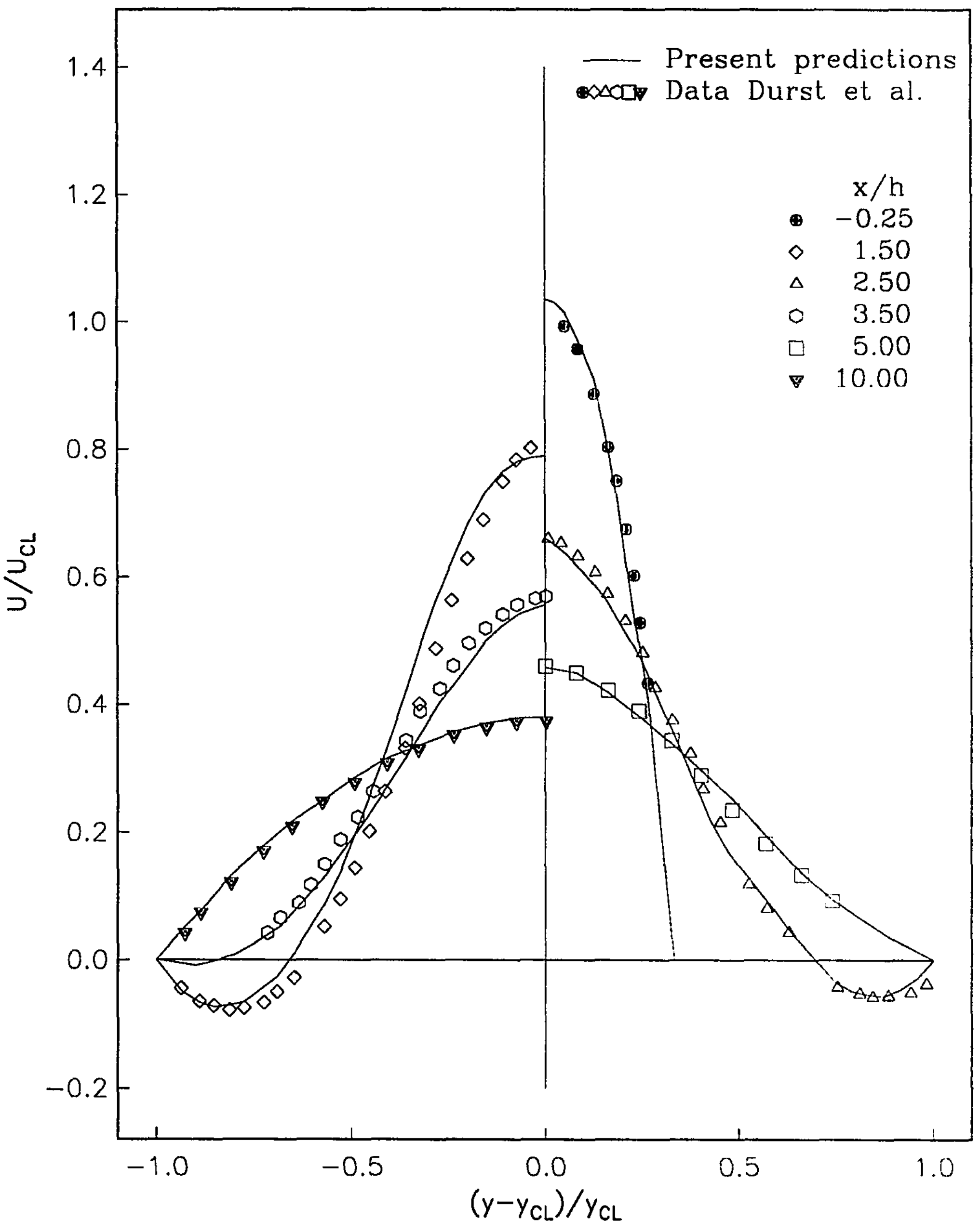


Figure 5.10: Velocity profiles for a laminar flow in a channel with a 3:1 symmetric sudden expansion, $Re_h = 56$

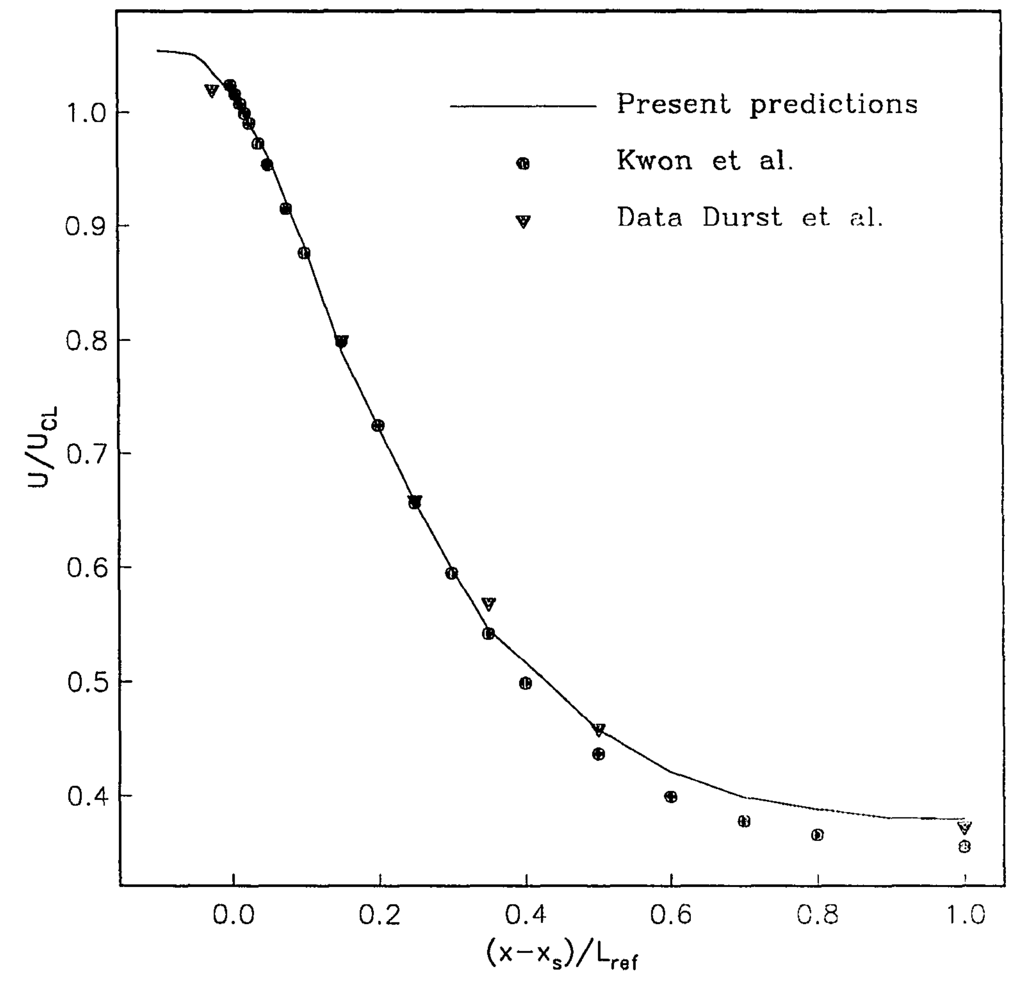


Figure 5.11: Centerline velocity distribution for a laminar flow in a channel with a 3:1 symmetric sudden expansion, $Re_h = 56$

The model problems presented here were compared with the incompressible numerical results of Gordon [47]. Uniform flow conditions were prescribed perpendicular to the cascade upstream of the first bank of tubes. Periodic boundary conditions were imposed at the upper and lower geometric boundaries to simulate an infinite number of parallel rows. The tubes in this case were in-line. An upstream Mach of 0.05 was used for both computed test cases. Both cases were computed on the same geometry. The geometry used in both test cases and the grid used for the second test case is shown in Fig. 5.12.

The first test case was computed at a Reynolds number of 1.0 based on the upstream conditions and cylinder radius. The velocity vectors are shown in Fig. 5.13 and compare well with the streamlines computed by Gordon [47]. At this Reynolds number the flow was nearly symmetric about both cylinders indicating that there was minimal influence of one bank of cylinders on the other.

The second test case was computed at a Reynolds number of 20.0 based on the same conditions as the test case above. Here the flow separates behind both cylinder banks. An enlargement of the velocity vectors near the cylinders is shown in Fig. 5.14 and the length of the separated regions behind both of the cylinders compares well with those computed by Gordon [47]. As noted by Gordon [47], the length of the separation behind the second bank of tubes is slightly smaller than that behind the first bank. The first set of cylinders accelerates the flow in the freestream so the slower wake flow impacts the second set of cylinders. The flow was symmetric about an imaginary horizontal line that passed through the centers of the cylinders. There does not seem to be any difference in the angular location of the actual separation point on either cylinder.

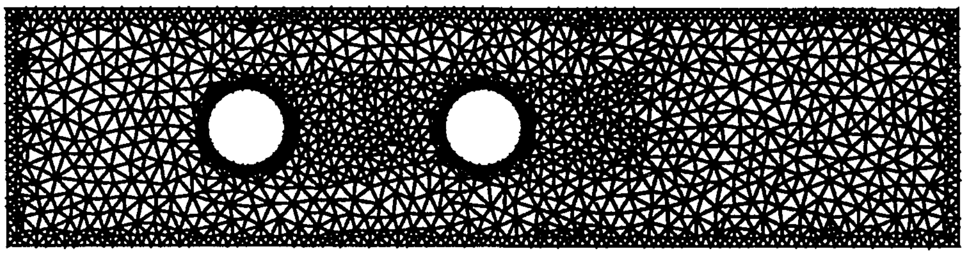


Figure 5.12: Grid about periodic tandem circular cylinders

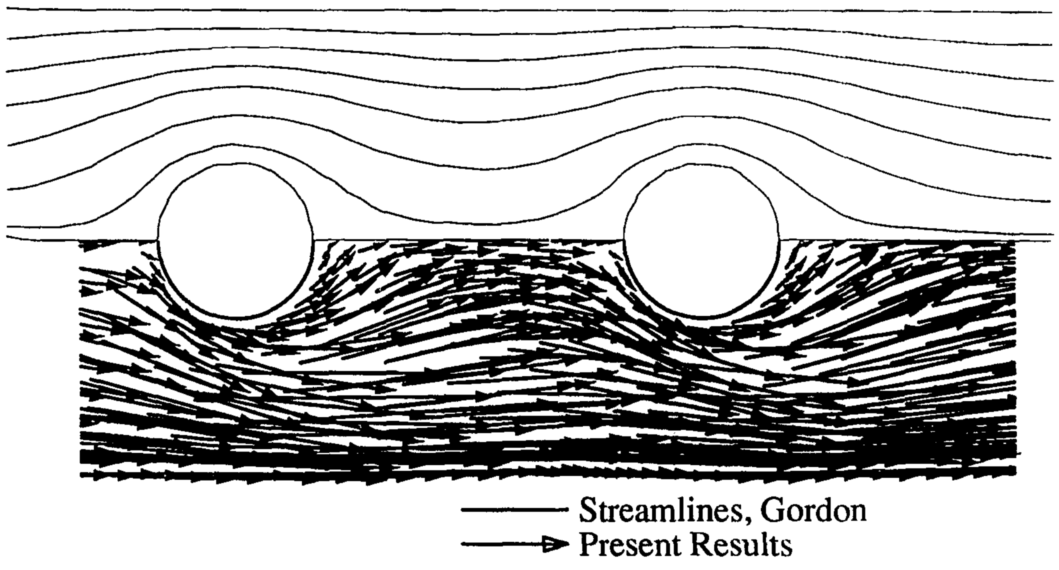


Figure 5.13: Periodic tandem circular cylinders in cross flow. $Re_r = 1$

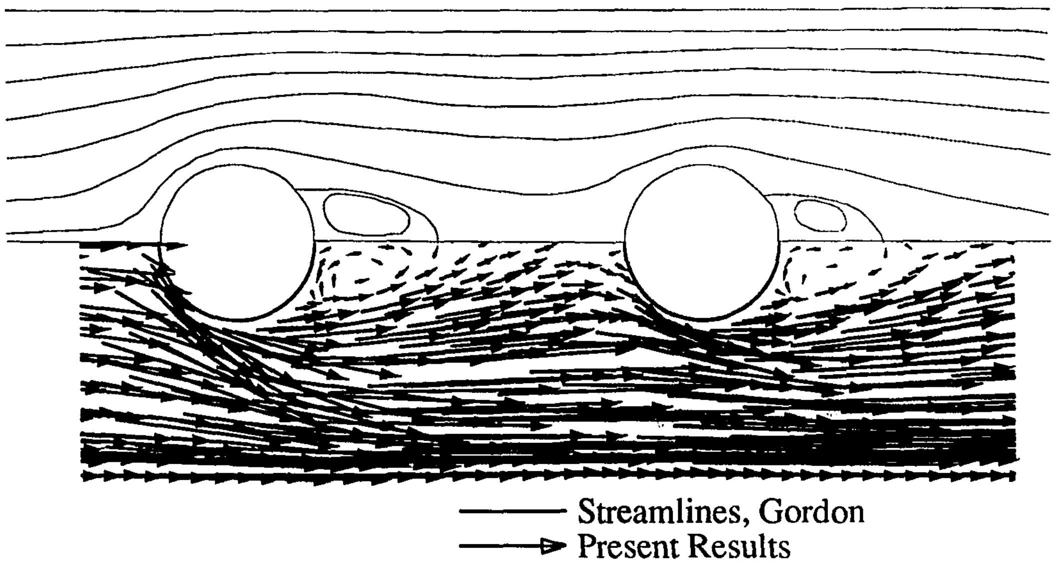


Figure 5.14: Periodic tandem circular cylinders in cross flow, $Re_T = 20$

5.5 Four Port Valve

The final results are presented to show the versatility of applying boundary conditions when using an unstructured grid flow solver. The geometry represents a two-dimensional valve with four ports where inlet or outlet flow boundary conditions can be specified. The grid used for this test case was shown in Fig. 3.12. A reference was made to such a flow geometry in an article by Ackert [48]. The actual flow conditions were not given. Here the flow was computed at two Reynolds numbers. Fluid enters through a channel on the left, enters the circular cavity, and exits through a channel at the bottom. For both test cases fully developed conditions were prescribed at the inlet. The flow redevelops along the open channel. Both test cases were computed with an inlet Mach number of 0.05. An interesting aspect of the geometry was that the closed valve ports acted as driven cavities. The unstructured grid formulation allows the application of exit boundary conditions at any or all of the three remaining ports. This type of valve geometry can be found in an application like fluid networks.

Fluid flow was first simulated in the valve geometry at a Reynolds number of 10 based on inlet conditions and channel height. The velocity vectors of this steady state flow are shown in Fig. 5.15. Here the fluid near the wall of the inlet channel was accelerated as the corner of the cavity was anticipated. A clockwise rotation of the fluid was followed through the circular volume. The fluid in the closed cavity ports was driven in a counterclockwise rotation. The band of fluid then enters the open lower exit channel and again becomes fully developed.

Next the same valve geometry was used to simulate the fluid flow at a Reynolds number of 50 based on inlet conditions and channel height. The velocity vectors of this computation are shown in Fig. 5.16. Again a fully developed velocity profile was specified at the inlet to the channel. The velocity of the fluid near the wall accelerates as it approaches the entrance to the circular chamber. Contrary to the previous case, the banded fluid actually drove a large volume of fluid in a counterclockwise direction. This had the effect of driving the closed valve ports in an opposite rotation direction from that of the lower Reynolds number test case. Also, the band of fluid did not diffuse as much across the circular volume. In the open exit channel the fluid redevelops into a fully developed parabolic profile.

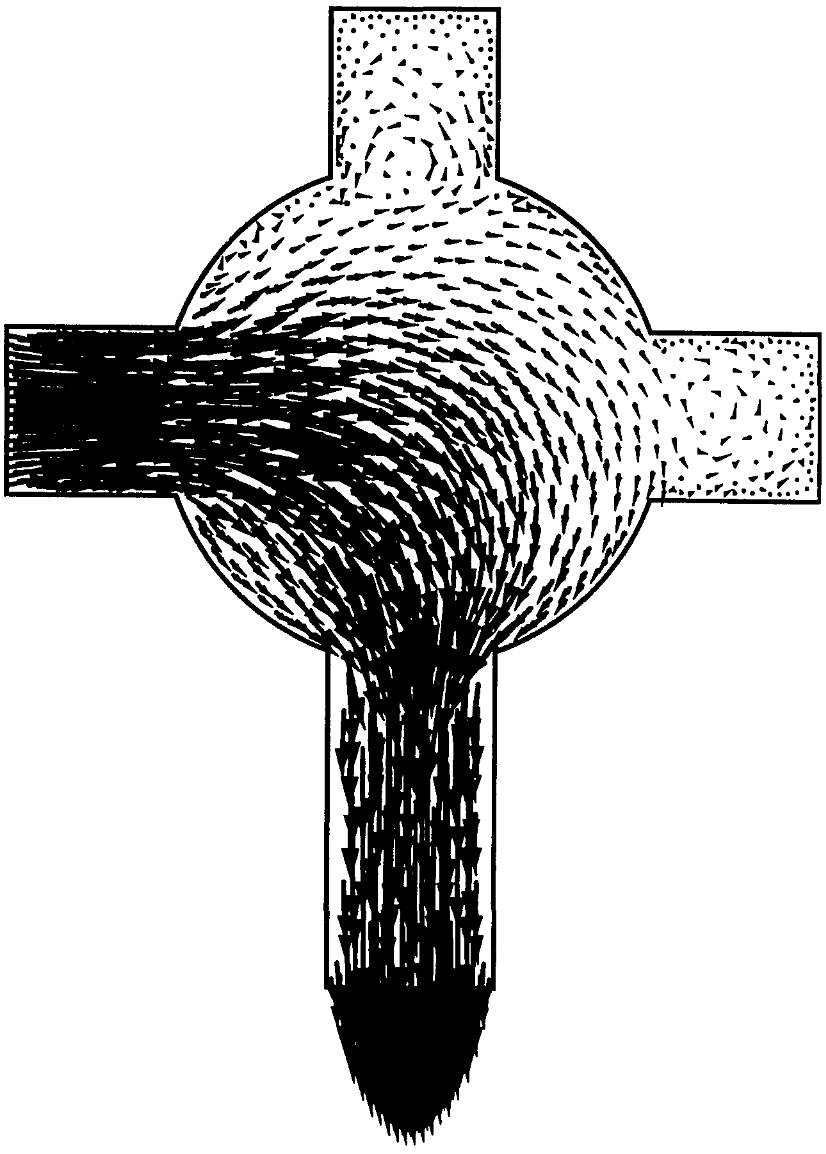


Figure 5.15: Four port valve, $Re_h = 10$

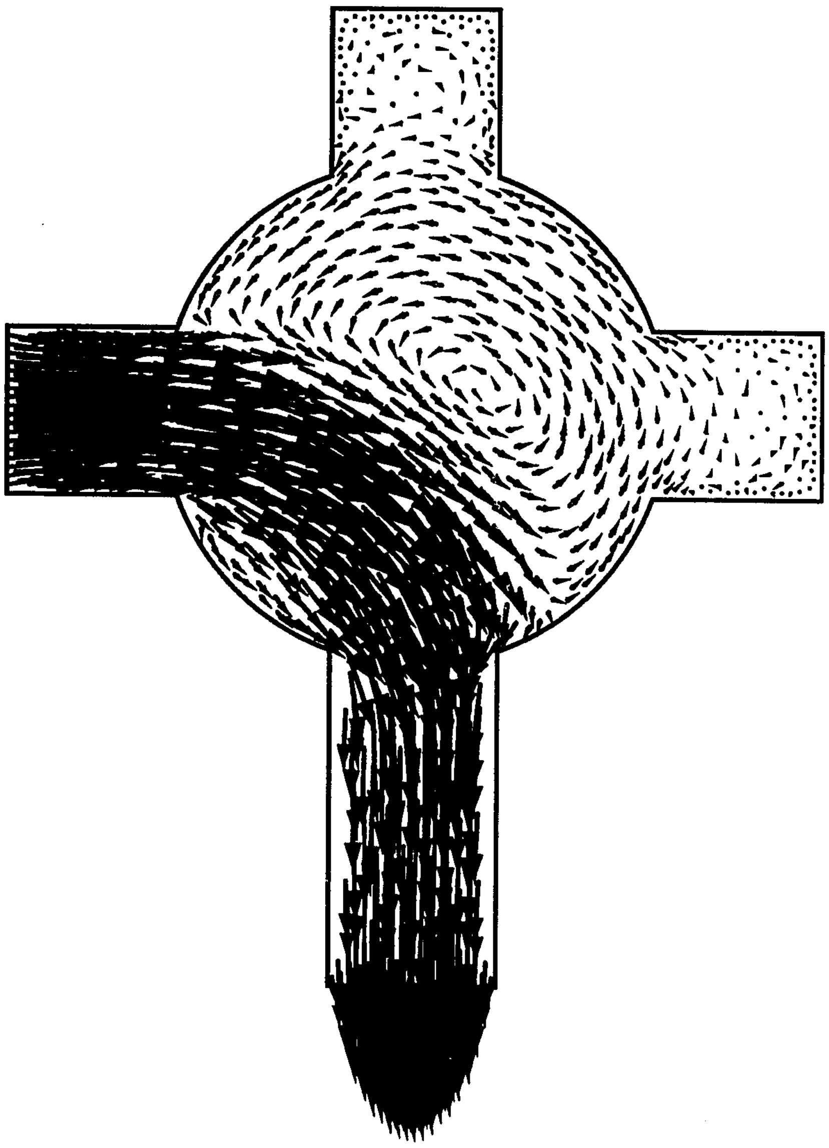


Figure 5.16: Four port valve, $Re_h = 50$

6. CONCLUSIONS

A two-dimensional unstructured grid implicit flow solver was described. Although only internal flow problems were considered in this study. the method is believed to be applicable to external flows as well.

The compressible flow equations were discretized in finite volume form. They were integrated in time by an implicit Gauss-Seidel relaxation procedure. Diagonal point and the diagonal block Gauss-Seidel solvers were investigated. A sparse matrix iterative solver, SITRSOL, was also evaluated as a way of obtaining a solution. Results were presented for inviscid flow over a channel bump at subsonic and transonic conditions. The subsonic case produced nearly symmetric flow over the bump as was observed in the results of another researcher. The location of the sonic line on the aft portion of the bump for the transonic case compared well with results obtained by several investigators.

The main thrust of this study was the computation of viscous flows. First the code was verified on some standard laminar test cases. Developing flow in a channel was computed at several Reynolds numbers. The results compared well with available experimental data and with numerical results. A temporal preconditioner was used to enable the code to run at very low Mach numbers. Flows were computed at a Mach number of 0.0005 without much effect on the convergence history of the solution. The different solvers were tested and compared on the developing channel flow test cases. Then a symmetric sudden expansion test case was run to show the capability of the code to compute laminar separation. Also the geometry of the sudden expansion was much easier to resolve and the solution was more straight forward since it was unnecessary to do any grid patching or grid overlaying that would be required if a structured grid code was used. The numerical results compared well with the available experimental data. A geometrically more complex test case was then computed. A periodic cross flow over periodic tandem circular cylinders was solved numerically at Reynolds numbers of 1 and 20. Comparisons were made with the streamlines computed by another investigator. The predicted shape and size of the separation bubbles behind the cylinders at a Reynolds number of 20 were in good agreement. Finally the flows were computed in a four port valve at Reynolds numbers of 10 and 50. This showed the versatility in imposing boundary condition offered by an unstructured flow code. No data were available for comparison. The solution revealed some very interesting flow characteristics.

Several conclusions can be drawn from the present study.

1. A triangular unstructured grid can be generated about very complex geometries where the use of a single structured grid cannot be considered in most cases. This gives the advantage that a single computer code can be used in a wide variety of flow geometry applications. However, this advantage is somewhat dampened by the complexity of coding required for solving a system of differential equations on this unstructured grid. Details such as boundary conditions are more difficult to implement on an unstructured grid.

2. It was found that the diagonal block Gauss-Seidel solver was more robust than

the point diagonal Gauss-Seidel version of solver. The diagonal point solver seemed sensitive to initial conditions and diagonal dominance.

3. A coloring scheme was used to take advantage of the vectorization of the implicit Gauss-Seidel solver. A minimum of extra storage was necessary for a significant reduction in computer time. The time spent in the solver was decreased by a factor of 7.6. It was found that the recurrence in the viscous fluxes had little affect on the convergence of the solution to a steady state.

4. The use of the sparse matrix iterative solver allowed a much larger time step to be used than that of the Gauss-Seidel solver. However, every time step using the sparse matrix solver was significantly more expensive. Even so, the sparse solver ran at 2 to 2.5 times faster than the block Gauss-Seidel solver. Several different conjugate gradient like solvers were tested with the matrix preconditioners available with SITRSOL. Some of the preconditioners did not allow a solution to the equations. The incomplete **LU** preconditioner was found to be the best. The solvers all exhibited the same basic convergence rate. The difference in the solvers was in the time that was required to obtain the same convergence level. The generalized minimum residual method was found to be the fastest for the particular test case that was being computed.

5. A temporal preconditioning was added to the flow equations to allow solutions at very low Mach numbers. The preconditioner was implemented such that both steady state and time accurate flows could be computed. Steady state solutions were considered in this study. Mach number flows as low as 0.0005 were computed without degradation to the convergence rate of the solution

procedure. Without the preconditioning, convergence was either very slow due to the necessity of running at a much smaller time step, or the equations could not be converged to a solution. Preconditioning was relatively easy to add to the numerical code.

There are several topics of research in the area of unstructured grid flow solvers that deserve more attention. Several of the items mentioned below are being investigated by other researchers for their specific computer codes and applications.

1. Several discretization methods need to be investigated. In the current work the central difference formulation was used. Upwind methods could also be considered. This area could most expeditiously be investigated in the current code by implementing the upwinding through the artificial dissipation. This idea also extends to how artificial dissipation is used in conjunction with preconditioning.

2. Grid adaptation was one of the reasons for using a triangular unstructured grid formulation. In the current code, the adaptation was not done automatically while the code was running. Instead, a grid was input to the flow code with interesting areas of the flow regime already resolved. A more appropriate method would be to allow the flow to cause the grid to automatically adapt to regions of high gradients or large error.

3. More work needs to be done on finding ways of vectorizing an implicit unstructured grid flow code. In the current formulation this seems to be especially important in the implementation of the boundary conditions. This may require the use of more computer storage. Each cell may need a small square matrix that triggers the appropriate boundary condition for every cell edge.

4. The parallelization of the unstructured computer code is becoming more important as three-dimensional effects in a flow field require modeling. The flow domain needs to be divided between several computers since the number of cells or grid points in the computational domain becomes too large for one computer to solve effectively. Many of the ideas can be tested in two dimensions to determine their viability.

5. Different sparse matrix solvers need to be investigated and compared for speed and stability. Another concern is how well the solver will parallelize.

6. One version of a temporal preconditioner was investigated in the current work. Other preconditioners can be used to determine if they have a positive or negative effect on the convergence characteristics or speed of the scheme.

BIBLIOGRAPHY

[1] Steger, J., and Warming, R. F., "Flux Vector Splitting of the Inviscid Gasdynamics Equations with Application to Finite-Difference Methods," NASA TM-78605, 1978.

[2] MacCormack, R. W., "A Numerical Method for Solving the Equations of Compressible Viscous Flow," AIAA 81-0110, 1981.

[3] Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes," AIAA 81-1259, 1981.

[4] MacCormack, R. W., "Current Status of Numerical Solutions of the Navier-Stokes Equations," AIAA 85-0032, 1985.

[5] Jameson, A., "Successes and Challenges in Computational Aerodynamics," AIAA 87-1184, 1987.

[6] Steger, J. L., Dougherty, F. C., Benek, J. A., "A Chimera Grid Scheme," *Advances in Grid Generation,* ASME FED-5, pp. 59-69, 1983.

[7] Rai, M. M., "A Relaxation Approach to Patched-Grid Calculations with the Euler Equations," AIAA 85-0295, 1985.

[8] Sibson, R., "Locally equiangular triangulations," *The Computer Journal,* 21, No. 3, pp. 243-245, 1978.

[9] Peraire, J., Vahdati, M., Morgan, K, and Zienkiewicz, O. C.. "Adaptive Remeshing for Compressible Flow Computations," *Journal of Computational Physics,* 72, No. 2, pp. 449-466, 1987.

[10] Holmes, D.G., and Snyder, D. D., "The Generation of Unstructured Triangular Meshes Using Delaunay Triangulation," *Numerical Grid Generation in Computational Fluid Mechanics '88,* Pineridge Press, Miami, 1988. pp. 643-652.

[11] Holmes, D.G., and Connel, S. D., "Solution of the 2D Navier-Stokes Equations on Unstructured Adaptive Grids," AIAA 89-1932, 1989.

[12] Jameson, A., Mavriplis, D., "Finite Volume Solution of the Two Dimensional Euler Equations on a Regular Triangular Mesh," AIAA 85-0435. 1985.

[13] Lindquist, D. R., and Giles, M. B, "A Comparison of Numerical Schemes on Triangular and Quadrilateral Meshes," *Proceedings of 11th International Conference on Numerical Methods in Fluid Dynamics.* Springer-Verlag, New York, 1988.

[14] Mavriplis, D., Jameson, A., Martinelli, L., "Multigrid Solution of the Navier-Stokes Equations on Triangular Meshes," AIAA 89-0120, 1989.

[15] Venkatakrishnan, V., Barth, T. J., "Application of Direct Solvers to Unstructured Meshes for the Euler and Navier-Stokes Equations Using Upwind Schemes," AIAA 89-0364, 1989.

[16] Barth, T. J., Jespersen, D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA 89-0366, 1989.

[17] Whitaker, D. L., and Grossman, B., "Two-Dimensional Euler Computations on a Triangular Mesh Using an Upwind Finite-Volume Scheme," AIAA 89-0470, 1989.

[18] Caruso, S., "Development of an Unstructured Mesh/Navier-Stokes Method for Aerodynamics of Aircraft with Ice Accretions," AIAA 90-0758. 1990.

[19] Batina, J. T., "Development of Unstructured Grid Methods for Steady and Unsteady Aerodynamic Analysis," Presented at the 17th Congress of the International Council of the Aeronautical Sciences, Stockholm, Sweden. 1990.

[20] Hase, J. E., Anderson, D. A., Parpia, I., "A Delaunay Triangulation Method and Euler Solver for Bodies in Relative Motion." AIAA 91-1590. 1991.

[21] Usab, W. J., Jiang, Y. T., "Development of a Solution Adaptive Unstructured Scheme for Quasi-Three-Dimensional Inviscid Flows Through Advanced Turbomachinery Cascades," AIAA 91-0132, 1991.

[22] Barth, R. J., "Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes," AIAA 91-0721, 1991.

[23] Venkatakrishnan, V., Mavriplis, D. J., "Implicit Solvers for Unstructured Meshes," ICASE Report No. 91-40, 1991.

[24] Baker, T. J., "Unstructured Meshes and Surface Fidelity for Complex Shapes," AIAA 91-1591, 1991.

[25] Woodard, P. R., Batina, J. T., "Quality Assessment of Two- and Three-Dimensional Unstructured Meshes and Validation of an Upwind Euler Flow Solver," AIAA 92-0444, 1992.

[26] Batina, J. T., "A Fast Implicit Upwind Solution Algorithm for Three-Dimensional Unstructured Dynamic Meshes," AIAA 92-0447, 1992.

[27] Das, R., Mavriplis, D. J., Saltz, J., Gupta, S., Ponnusamy, R., "The Design and Implementation of a Parallel Unstructured Euler Solver using Software Primitives," ICASE Report No. 92-12, 1992.

[28] Choi, D., Merkel, C. L., "Application of Time-Iterative Schemes to Incompressible Flows," AIAA 84-1638, 1984.

[29] Turkel, E, "Preconditioned Methods for Solving the Incompressible and Low Speed Compressible Equations," *Journal of Computational Physics*, 72, pp. 277-298, 1987.

[30] Feng, J., Merkel, C. L., "Evaluation of Preconditioning Methods for Time-Marching Systems," AIAA 90-0016, 1990.

[31] Withington, J. P., Shuen, J. S., Yang, V., "A Time Accurate, Implicit Method for Chemically Reacting Flows at All Mach Numbers," AIAA 91-0581, 1991.

[32] Choi, Y., Merkel, C. L., "Time-Derivative Preconditioning for Viscous Flows," AIAA 91-1652, 1991.

[33] Appel,K and Haken, W, "Every planer map is 4-colorable," *Bull. Am. Math. Soc.* 82, pp. 711-712, 1976.

[34] Eisenstadt, S. C., Gursky, M. C., Schultz, M. H., Sherman, A. H., "Yale Sparse Matrix Package: II The Nonsymmetric Codes," Research Report No. 114, Yale University, New Haven, 1977.

[35] Cray Research, Inc., "Volume 3: UNICOS Math and Scientific Library Reference Manual," SR-2081 6.0, pp. 227-243, 1991.

[36] Golub, G. H., Van Loan, C. F., *Matrix Computations,* The John Hopkins University Press, Baltimore, 1989.

[37] Saad, Y., Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing,* 7, pp. 856-869, 1986.

[38] Press W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., *Numerical Recipes,* Cambridge University Press, Cambridge, 1989.

[39] Wigton, L. B., Yu, N. J., and Young, D. P., "GMRES Acceleration of Computational Fluid Dynamics Codes," AIAA 85-1494, 1985.

[40] Ni, R., "A Multi-Grid Scheme for Solving the Euler Equations," AIAA 81-1025, 1981.

[41] Chima, R. V., Turkel, E., Schaffer, S., "Comparison of Three Explicit Multigrid Methods for the Euler and Navier-Stokes Equations," AIAA 87-0602, 1987.

[42] Tenpas, P. W. and Pletcher, R. H., "Solution of the Navier-Stokes equations for subsonic flows using a coupled space-marching method," AIAA 87-1173-cp, 1987.

[43] Morihara, H. and Cheng, R. T., "Numerical solution of the viscous flow in the entrance region of parallel plates," *Journal of Computational Physics,* 11, No. 4, pp. 550-572, 1973.

[44] Chilukuri, R. and Pletcher, R. H., "Numerical solution to the partially parabolized Navier-Stokes equations for developing flow in a channel," *Numerical Heat Transfer,* 3, No. 2, pp. 169-187, 1980.

[45] Durst, F., Melling, A., Whitelaw, J. H., "Low Reynolds number flow over a plane symmetric sudden expansion," *Journal of Fluid Mechanics*, 64. pp. 441-428, 1974.

[46] Kwon, O. K., Pletcher, R. H., and Lewis, J. P., "Prediction of sudden expansion flows using the boundary-layer equations," *Trans. ASME. J. Fluids Engineering,* 106, pp. 285-291, 1984.

[47] Gordon, D., "Numerical Calculations on Viscous Flow Fields through Cylinder Arrays," *Computers and Fluids,* 6, pp. 1-13, 1978.

[48] Ackeret, J., "Aspects of Internal Flow," Proceedings of the Symposium on the Fluid Mechanics of Internal Flows, General Motors Research Laboratories, Warren, MI, 1965.

APPENDIX A. LOCATING LEVEL 2 CELLS

To compute the viscous fluxes it was necessary to determine the cell numbers and orientation locally at level 2. A general description of how these cells were identified and some details of the algorithm will be presented.

Figure A.1 shows cell A and its level $1(B, C, D)$ and level $2(E, F, G, H, I, J)$ cell neighbors. The subscripts of the cell labels refer to the numbers associated with the cells. In this case the cells A through J are numbered 1 through 10, respectively. The connectivity matrix described at the end of the chapter on grid generation gives the numbers associated with the level 1 cells of cell A. The level 2 cell numbers can also be easily obtained by shifting to the adjacent neighbors of cell A and finding their level 1 cell numbers. Note that the cell number associated with cell A will be one of those numbers. This can most easily be explained by referring to Fig. A.1. First, cell A has a cell number of 1 and its level 1 cells are given by the connectivity matrix as 2, 3, and 4 for cells B, C, and D respectively. Next the level 2 cells need to be determined. So, by shifting to cell B its level 1 cells reveal the level 2 cells that are adjacent to cell B. In this case cell B has cells 5, 6, and 1 as its level 1 cells. The real problem is not finding the cell numbers of the level 2 cells but instead how these cells can be oriented such that they can be traversed in a counterclockwise manner when doing a numerical integration. This orientation procedure should result in cell

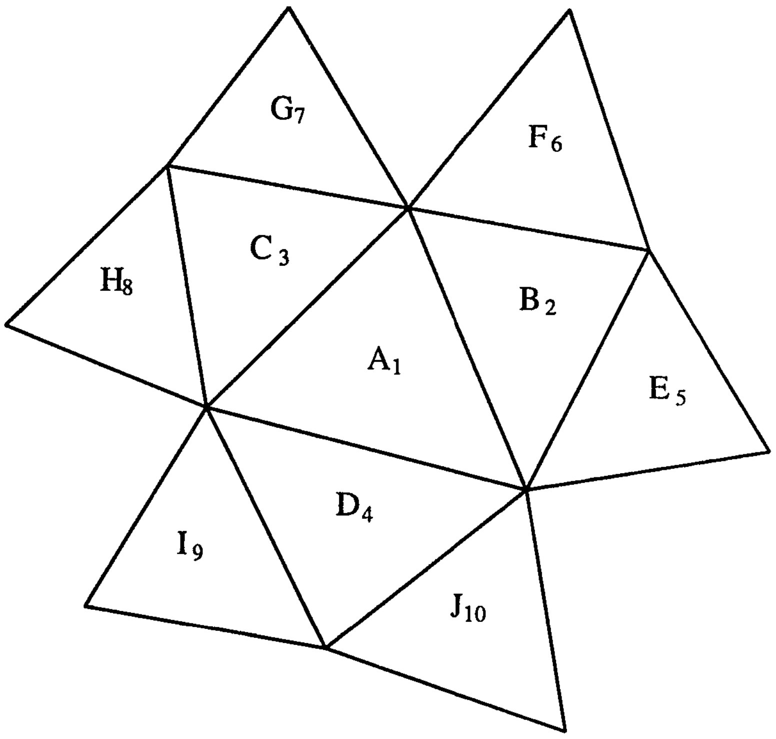


Figure A.1: Level 1 and level 2 cells about cell A with subscript cell numbers

numbers that are in the order E, F, G, H, I and J. Note that the level 1 cells are always numbered locally in a counterclockwise manner. This reduces the number of permutations required to obtain the correct orientation. A permutation matrix is used along with the available connectivity matrix to accomplish the ordering.

An algorithm can be written to accomplish the above local ordering of cells. First the permutation matrix is defined as

$$nperm(k,l) = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}.$$

The three faces of cell A(cell number i in this example) are determined from the connectivity matrix as

$$\begin{aligned} nf1 &= NCELL(1,i) \\ nf2 &= NCELL(2,i) \\ nf3 &= NCELL(3,i). \end{aligned}$$

It is now necessary to find the cell numbers for B, C, and D by

$$\begin{aligned} icb &= NFACE(1,nf1) + NFACE(2,nf1) - i \\ icc &= NFACE(1,nf2) + NFACE(2,nf2) - i \\ icd &= NFACE(1,nf3) + NFACE(2,nf3) - i. \end{aligned}$$

This algorithm simply looks at the cell numbers on both sides of a given face, adds the cell numbers together, and subtracts the cell number associated with cell A. This gives the level 1 cell number across a given face adjacent to cell A. This is done for all three faces of each local triangular cell. Next, the faces of cell B are obtained from its local connectivity matrix as

$$nfs1 = NCELL(1, icb)$$
$$nfs2 = NCELL(2, icb)$$
$$nfs3 = NCELL(3, icb).$$

Now the cells that surround cell b are temporarily stored in the order in which they are oriented with respect to cell B. This is done by setting

$$ics(1) = NFACE(1, nfs1) + NFACE(2, nfs1) - icb$$
$$ics(2) = NFACE(1, nfs2) + NFACE(2, nfs2) - icb$$
$$ics(3) = NFACE(1, nfs3) + NFACE(2, nfs3) - icb.$$

This is the same type of coding that gave the level 1 cells for A shown above. Now the hyperbolic cosine and integer conversion functions are used to locate cell E and F with respect to cell A. The permutation matrix defined above is used to give the appropriate orientation of cell E and F. So,

$$nfa = INT(1./COSH(FLOAT(ics(1) - ica)))$$
$$nfb = INT(1./COSH(FLOAT(ics(2) - ica)))$$
$$nfc = INT(1./COSH(FLOAT(ics(3) - ica)))$$
$$lab = nfa + nfb * 2 + nfc * 3$$
$$lad = nperm(lab, 2)$$
$$lda = nperm(lab, 3).$$

The integers nfa, nfb, and nfc take on values of zero or one. The cell numbers for E and F can now be found by

$$ice = ics(lad)$$
$$icf = ics(lab).$$

Here *ice* and *icf* are the cell numbers of E and F respectively.

The cell numbers for G, H, I, and J are determined in a similar fashion. These cell numbers can either be stored as an extended connectivity matrix or recomputed for every cell at every iteration. In the present study these level 1 and level 2 cell numbers were stored locally for each cell. The above algorithm was accessed as a preprocessor and the cell numbers were stored as part of the connectivity matrix..

APPENDIX B. CONSTRUCTING THE MATRIX EQUATION

The matrix equation $\mathbf{A}\vec{x} = \vec{b}$ was constructed using the connectivity matrix and the viscous flow equations written in discrete delta form. This can be most easily shown by looking at a typical row of the the matrix equation. Consider the cells that are shown in Fig. A.1. The system of equations are written in block form where the **A** matrix is given by

$$\begin{bmatrix} [A_{1,1}] & [A_{1,2}] & [A_{1,3}] & [A_{1,4}] & [A_{1,5}] & [A_{1,6}] & [A_{1,7}] & [A_{1,8}] & [A_{1,9}] & [A_{1,10}] & \cdots [0] \cdots \\ & & \vdots & & & & & & & & \\ & & & & & \vdots & & & & & \\ & & & & & & & & \vdots & & \\ & & & & & & & & & & [A_{n,n}] \end{bmatrix}$$

and the vectors are written as

$$\vec{x} = \begin{bmatrix} [x_1] \\ [x_2] \\ [x_3] \\ [x_4] \\ [x_5] \\ [x_6] \\ [x_7] \\ [x_8] \\ [x_9] \\ [x_{10}] \\ \vdots \\ [x_n] \end{bmatrix}, \qquad \vec{b} = \begin{bmatrix} [b_1] \\ [b_2] \\ [b_3] \\ [b_4] \\ [b_5] \\ [b_6] \\ [b_7] \\ [b_8] \\ [b_9] \\ [b_{10}] \\ \vdots \\ [b_n] \end{bmatrix}$$

respectively. The subscript n refers to the total number of triangular control volumes in the computational domain. The vector x contains the unknowns ΔP, Δu, Δv, and ΔT for each block of the matrix **A**. The vector b is the column of known quantities from the discretization of the equations in delta form.

As an example, the x-momentum equation will be used to show how the coefficients of the second row of the block matrices are computed. First the inviscid terms will be considered and then the viscous terms described earlier in the section on discretization will be added.

Only the level 1 cells are used to compute the inviscid terms. These are cells B through D or cell numbers 2 through 4 respectively. By referring to Eqs. (2.7, 4.3, and 4.9) the x-momentum equation can be written in discrete form as

$$\frac{S_1}{\Delta t}\left[\frac{P_1}{T_1}\Delta u_1 + \frac{u_1}{T_1}\Delta P_1 - \frac{P_1 u_1}{{T_1}^2}\Delta T_1\right] + \frac{1}{2}\left[\left(\frac{2P_1u_1}{T_1}\Delta u_1 + \frac{2P_2u_2}{T_2}\Delta u_2\right)\Delta y_{12}\right.$$
$$+\left(\frac{2P_1u_1}{T_1}\Delta u_1 + \frac{2P_3u_3}{T_3}\Delta u_3\right)\Delta y_{13} + \left(\frac{2P_1u_1}{T_1}\Delta u_1 + \frac{2P_4u_4}{T_4}\Delta u_4\right)\Delta y_{14}$$
$$+\left(\frac{{u_1}^2}{T_1}\Delta P_1 + \frac{{u_2}^2}{T_2}\Delta P_2\right)\Delta y_{12} + \left(\frac{{u_1}^2}{T_1}\Delta P_1 + \frac{{u_3}^2}{T_3}\Delta P_3\right)\Delta y_{13}$$
$$+\left(\frac{{u_1}^2}{T_1}\Delta P_1 + \frac{{u_4}^2}{T_4}\Delta P_4\right)\Delta y_{14} - \left(\frac{P_1{u_1}^2}{{T_1}^2}\Delta T_1 + \frac{P_2{u_2}^2}{{T_2}^2}\Delta T_2\right)\Delta y_{12}$$
$$-\left(\frac{P_1{u_1}^2}{{T_1}^2}\Delta T_1 + \frac{P_3{u_3}^2}{{T_3}^2}\Delta T_3\right)\Delta y_{13} - \left(\frac{P_1{u_1}^2}{{T_1}^2}\Delta T_1 + \frac{P_4{u_4}^2}{{T_4}^2}\Delta T_4\right)\Delta y_{14}$$
$$+R(\Delta P_1 + \Delta P_2)\,\Delta y_{12} + R(\Delta P_1 + \Delta P_3)\,\Delta y_{13} + R(\Delta P_1 + \Delta P_4)\,\Delta y_{14}$$
$$-\left(\frac{P_1v_1}{T_1}\Delta u_1 + \frac{P_2v_2}{T_2}\Delta u_2\right)\Delta x_{12} - \left(\frac{P_1v_1}{T_1}\Delta u_1 + \frac{P_3v_3}{T_3}\Delta u_3\right)\Delta x_{13}$$
$$-\left(\frac{P_1v_1}{T_1}\Delta u_1 + \frac{P_4v_4}{T_4}\Delta u_4\right)\Delta x_{14} - \left(\frac{P_1u_1}{T_1}\Delta v_1 + \frac{P_2u_2}{T_2}\Delta v_2\right)\Delta x_{12}$$
$$-\left(\frac{P_1u_1}{T_1}\Delta v_1 + \frac{P_3u_3}{T_3}\Delta v_3\right)\Delta x_{13} - \left(\frac{P_1u_1}{T_1}\Delta v_1 + \frac{P_4u_4}{T_4}\Delta v_4\right)\Delta x_{14}$$
$$-\left(\frac{u_1v_1}{T_1}\Delta P_1 + \frac{u_2v_2}{T_2}\Delta P_2\right)\Delta x_{12} - \left(\frac{u_1v_1}{T_1}\Delta P_1 + \frac{u_3v_3}{T_3}\Delta P_3\right)\Delta x_{13}$$
$$-\left(\frac{u_1v_1}{T_1}\Delta P_1 + \frac{u_4v_4}{T_4}\Delta P_4\right)\Delta x_{14} + \left(\frac{P_1u_1v_1}{{T_1}^2}\Delta T_1 + \frac{P_2u_2v_2}{{T_2}^2}\Delta T_2\right)\Delta x_{12}$$
$$+\left(\frac{P_1u_1v_1}{{T_1}^2}\Delta T_1 + \frac{P_3u_3v_3}{{T_3}^2}\Delta T_3\right)\Delta x_{13}$$
$$\left.+\left(\frac{P_1u_1v_1}{{T_1}^2}\Delta T_1 + \frac{P_4u_4v_4}{{T_4}^2}\Delta T_4\right)\Delta x_{14}\right] = R.H.S.,$$

where the subscripts 1 through 4 and 12 through 14 refer to the cell number and cell edges respectively. The variable S_1 is the area of cell number 1. The $R.H.S.$ is the right-hand side of the x-momentum equation and is given by

$$R.H.S. = -\frac{S_1}{\Delta t}\left(\frac{P_1 u_1}{T_1} - \frac{\hat{P}_1 \hat{u}_1}{\hat{T}_1}\right) - \frac{1}{2}\left[\left(\frac{P_1 {u_1}^2}{T_1} + \frac{P_2 {u_2}^2}{T_2}\right)\Delta y_{12}\right.$$
$$+\left(\frac{P_1 {u_1}^2}{T_1} + \frac{P_3 {u_3}^2}{T_3}\right)\Delta y_{13} + \left(\frac{P_1 {u_1}^2}{T_1} + \frac{P_4 {u_4}^2}{T_4}\right)\Delta y_{14}$$
$$+R(P_1 + P_2)\Delta y_{12} + R(P_1 + P_3)\Delta y_{13} + R(P_1 + P_4)\Delta y_{14}$$
$$+\left(\frac{P_1 u_1 v_1}{T_1} + \frac{P_2 u_2 v_2}{T_2}\right)\Delta x_{12} + \left(\frac{P_1 u_1 v_1}{T_1} + \frac{P_3 u_3 v_3}{T_3}\right)\Delta x_{13}$$
$$\left.+\left(\frac{P_1 u_1 v_1}{T_1} + \frac{P_4 u_4 v_4}{T_4}\right)\Delta x_{14}\right],$$

where the $\hat{}$ terms are quantities from the previous time level. The coefficients multiplying similar delta unknown quantities are collected and written in matrix form. The geometric factors

$$\Delta x_{12} + \Delta x_{13} + \Delta x_{14}$$

and

$$\Delta y_{12} + \Delta y_{13} + \Delta y_{14}$$

are identically zero.

The inviscid coefficients of the delta quantities of the x-momentum equation are then stored in the **A** matrix as the second row of the first block row in this implementation. The viscous terms make contributions through both the level 1 and level 2 cells. As with the inviscid terms described above the terms that multiply similar delta unknown quantities are collected and added to the **A** matrix. The viscous terms will require additional storage for the contribution given through cells 5 through 10. The exception to this storage requirement is where boundary conditions are imposed. The viscous terms are added to the inviscid *R.H.S.* of the x-momentum equation as

$$
\begin{aligned}
R.H.S. = R.H.S. - \frac{\mu R}{2Re}\Bigg\{ &-\frac{4}{3}\Bigg[\frac{1}{S_{12}}\left[(u_2+u_5)\triangle y_{25}+(u_2+u_6)\triangle y_{26}\right.\\
&\left.+(u_1+u_3)\triangle y_{13}+(u_1+u_4)\triangle y_{14}\right]\triangle y_{12}+\frac{1}{S_{13}}\left[(u_1+u_2)\triangle y_{12}\right.\\
&\left.+(u_3+u_7)\triangle y_{37}+(u_3+u_8)\triangle y_{38}+(u_1+u_4)\triangle y_{14}\right]\triangle y_{13}\\
&+\frac{1}{S_{14}}\left[(u_1+u_2)\triangle y_{12}+(u_1+u_3)\triangle y_{13}+(u_4+u_9)\triangle y_{49}\right.\\
&\left.+(u_4+u_{10})\triangle y_{410}\right]\triangle y_{14}\Bigg]-\frac{2}{3}\Bigg[\frac{1}{S_{12}}\left[(v_2+v_5)\triangle x_{25}+(v_2+v_6)\triangle x_{26}\right.\\
&\left.+(v_1+v_3)\triangle x_{13}+(v_1+v_4)\triangle x_{14}\right]\triangle y_{12}+\frac{1}{S_{13}}\left[(v_1+v_2)\triangle x_{12}\right.\\
&\left.+(v_3+v_7)\triangle x_{37}+(v_3+v_8)\triangle x_{38}+(v_1+v_4)\triangle x_{14}\right]\triangle y_{13}\\
&+\frac{1}{S_{14}}\left[(v_1+v_2)\triangle x_{12}+(v_1+v_3)\triangle x_{13}+(v_4+v_9)\triangle x_{49}\right.\\
&\left.+(v_4+v_{10})\triangle x_{410}\right]\triangle y_{14}\Bigg]-\frac{1}{S_{12}}\left[(u_2+u_5)\triangle x_{25}+(u_2+u_6)\triangle x_{26}\right.\\
&\left.+(u_1+u_3)\triangle x_{13}+(u_1+u_4)\triangle x_{14}\right]\triangle x_{12}-\frac{1}{S_{13}}\left[(u_1+u_2)\triangle x_{12}\right.\\
&\left.+(u_3+u_7)\triangle x_{37}+(u_3+u_8)\triangle x_{38}+(u_1+u_4)\triangle x_{14}\right]\triangle x_{13}\\
&-\frac{1}{S_{14}}\left[(u_1+u_2)\triangle x_{12}+(u_1+u_3)\triangle x_{13}+(u_4+u_9)\triangle x_{49}\right.\\
&\left.+(u_4+u_{10})\triangle x_{410}\right]\triangle x_{14}\Big]+\frac{1}{S_{12}}\left[(v_2+v_5)\triangle y_{25}+(v_2+v_6)\triangle y_{26}\right.\\
&\left.+(v_1+v_3)\triangle y_{13}+(v_1+v_4)\triangle y_{14}\right]\triangle x_{12}+\frac{1}{S_{13}}\left[(v_1+v_2)\triangle y_{12}\right.\\
&\left.+(v_3+v_7)\triangle y_{37}+(v_3+v_8)\triangle y_{38}+(v_1+v_4)\triangle y_{14}\right]\triangle x_{13}\\
&+\frac{1}{S_{14}}\left[(v_1+v_2)\triangle y_{12}+(v_1+v_3)\triangle y_{13}+(v_4+v_9)\triangle y_{49}\right.\\
&\left.+(v_4+v_{10})\triangle y_{410}\right]\triangle x_{14}\Bigg\}.
\end{aligned}
$$

The subscripts 1 through 10 and 12 through 410 refer to the cell numbers and edge numbers, respectively. The edge number 410 is the boundary between cell 4 and cell 10, for example. Geometric quantities are computed in a counterclockwise manner.

The coefficients stored in the **A** matrix that multiply the delta quantities in the continuity, y-momentum, and energy equations are obtained in a similar fashion. The preconditioning terms are added to the diagonal blocks of the **A** matrix. The artificial dissipation is explicit and is added to the term on the right-hand side.

APPENDIX C. GRID GENERATION COMPUTER CODE

This computer program generates a grid based on the Delaunay triangulation criteria described in the chapter on grid generation. The input parameters are described in the subroutine INPUT. Boundary points are read in as input in the subroutine BOUNDI. The computational domain is triangulated and the node point coordinates and cell connectivity arrays are written to output files. A four color map is also generated and written to a file.

```
      PROGRAM GRID5
$INCLUDE grid5.common
      OPEN(UNIT=10,STATUS='FORMATTED',FILE='grid5.input')
      OPEN(UNIT=15,STATUS='FORMATTED',FILE='grid5.bndpts')
      OPEN(UNIT=20,STATUS='FORMATTED',FILE='grid5.rst')
      OPEN(UNIT=31,STATUS='FORMATTED',FILE='grid.plot1')
      OPEN(UNIT=32,STATUS='FORMATTED',FILE='grid.plot2')
      OPEN(UNIT=33,STATUS='FORMATTED',FILE='grid.plot3')
      OPEN(UNIT=40,STATUS='FORMATTED',FILE='facell.data')
      OPEN(UNIT=45,STATUS='FORMATTED',FILE='node.data')
      OPEN(UNIT=50,STATUS='FORMATTED',FILE='grid.plot')
      OPEN(UNIT=55,STATUS='FORMATTED',FILE='color.map')
      CALL INPUT
C     INPUT BOUNDARY POINTS
      CALL BOUNDI
      IF(IRST.GE.1)THEN
       NRW=1
       CALL RESTRT(NRW)
      ELSE
C     Triangulate boundary points
       CALL TRIAN
      ENDIF
C     Add points to the original boundary triangulation
      IF(NPA.GT.0) CALL ADDPT
```

```
C     Check orientation of triangle
       CALL ORIENT
C     Check for duplicate cells
       CALL REMDUP
C     Make certain boundary edges correspond to side A-B
       IF(NCCP.EQ.1)CALL SIDEAB
C     Determine face-cell connectivity
       IF(NCCP.EQ.1)CALL FACELL
C     Check for cells that have more than one face on a boundary
        IF(NCCP.EQ.1)CALL BFC
C     Make four color map
        IF(NCCP.EQ.1)CALL FCM
C     OUTPUT BOUNDARY GEOMETRY FOR PLOTTING
       CALL BPLOT
       CALL OUTPUT
C     Output Restart file
       NRW=2
       IF(IRST.LT.2)CALL RESTRT(NRW)
       STOP
       END


       SUBROUTINE INPUT
$INCLUDE grid5.common
C     NCB = Number of Closed Bodies
C     IRE = Type of REfinement
C     NPA = Number of Points to be Added
C     IRST= Restart file?
C     NCCP= Switch diagonal of cells with 2 faces on a boundary
C     NSWB,NSWE= node solid wall begin and end
C     NEXB,NEXE= node exit begin and end
C     NINB,NINE= node inlet begin and end
C     NSYB,NSYE= node symmetry begin and end
C     NSYP     = node # that is periodic with the first index # NSYB
       NAMELIST/INPUT1/NCB,IRE,NPA,IRST,NCCP
       NAMELIST/INPUT2/NSB,NEB,NIB,NYB
       READ(10,NML=INPUT1)
       READ(10,NML=INPUT2)
       DO 1 I=1,NSB
        READ(10,*)NSWB(I),NSWE(I)
     1 CONTINUE
       DO 2 I=1,NEB
        READ(10,*)NEXB(I),NEXE(I)
     2 CONTINUE
       DO 3 I=1,NIB
        READ(10,*)NINB(I),NINE(I)
     3 CONTINUE
```

```
      DO 4 I=1,NYB
       READ(10,*)NSYB(I),NSYE(I),NSYP(I)
    4 CONTINUE
      RETURN
      END



      SUBROUTINE BOUNDI
$INCLUDE grid5.common
      N=0
      DO 1 K=1,NCB
       READ(15,*)NPB(K)
       NPBT=NPB(K)
       print*,'Points on boundary',k,' =',npbt
        DO 1 I=1,NPBT
         N=N+1
         READ(15,*)NCO(N),XB(K,I),YB(K,I)
    1 CONTINUE
      NPTB=N
      print*,'Total number of initial points=',nptb
      IF(IRST.EQ.0)THEN
      NPTT=0
      DO 2 K=1,NCB
       NPBT=NPB(K)-1
       DO 2 I=1,NPBT
        NPTT=NPTT+1
        X(NPTT)=XB(K,I)
        Y(NPTT)=YB(K,I)
    2 CONTINUE
      ENDIF
      RETURN
      END



      SUBROUTINE TRIAN
$INCLUDE grid5.common
C   Triangulate the initial boundary nodes
      NEL=0
      DO 1 I=1,NPTB-1
       I1=NCO(I)
       I2=NCO(I+1)
       DO 1 I3=1,NPTT
       IF(I3.EQ.I1.OR.I3.EQ.I2) GO TO 1
C    Determine if points lie in a straight line
        CALL LINE
        IF(IFLAGL.EQ.1) THEN
C    Check Delaunay triangulation criterion
         CALL DELAUNY
         IF(IFLAGD.EQ.1) THEN
```

```
C     Store cell node point numbers
          NEL=NEL+1
          NCELL(4,NEL)=I1
          NCELL(5,NEL)=I2
          NCELL(6,NEL)=I3
           PRINT*,'The following nodes have been triangulated'
           PRINT*,'Cell',nel,I1,I2,I3
         ENDIF
        ENDIF
    1 CONTINUE

      RETURN
      END



      SUBROUTINE LINE
$INCLUDE grid5.common
C    Check if points are in a straight line.  A, B, and C are the
C    lengths of the sides of the triangle.  If the sum of two of
C    the sides is equal to the third, the triangle is flat.
      IFLAGL=1
      A=SQRT((X(I1)-X(I2))**2+(Y(I1)-Y(I2))**2)
      B=SQRT((X(I2)-X(I3))**2+(Y(I2)-Y(I3))**2)
      C=SQRT((X(I3)-X(I1))**2+(Y(I3)-Y(I1))**2)
      CK1=ABS((A+B)/C-1.0)
      CK2=ABS((B+C)/A-1.0)
      CK3=ABS((C+A)/B-1.0)
      IF(CK1.LT.1.E-5.OR.CK2.LT.1.E-5.OR.CK3.LT.1.E-5)IFLAGL=0
      RETURN
      END



      SUBROUTINE DELAUNY
$INCLUDE grid5.common
C    Compute distance from center of circumcircle to all other
C    points in the domain.
C    Calculate the radius of the circumcircle
      A=SQRT((X(I1)-X(I2))**2+(Y(I1)-Y(I2))**2)
      B=SQRT((X(I2)-X(I3))**2+(Y(I2)-Y(I3))**2)
      C=SQRT((X(I3)-X(I1))**2+(Y(I3)-Y(I1))**2)
      S=.5*(A+B+C)
      SS=4.*SQRT(S*(S-A)*(S-B)*(S-C))
      RC=A*B*C/SS
C    Center of circumcircle.
      A11=X(I1)-X(I2)
      A12=Y(I1)-Y(I2)
      A21=X(I1)-X(I3)
      A22=Y(I1)-Y(I3)
      B1=.5*(X(I1)**2-X(I2)**2+Y(I1)**2-Y(I2)**2)
      B2=.5*(X(I1)**2-X(I3)**2+Y(I1)**2-Y(I3)**2)
      XC=(B1*A22-B2*A12)/(A11*A22-A21*A12)
      YC=(B2*A11-B1*A21)/(A11*A22-A21*A12)

C    Check if center of circumcircle lies within the domain of
C    interest
      CALL CHKD
      IF(IFLAGC.EQ.0) THEN
       IFLAGD=0
       RETURN
      ENDIF

      IFLAGD=1
      DO 1 I=1,NPTT
       IF(I.EQ.I1.OR.I.EQ.I2.OR.I.EQ.I3) GO TO 1
C    Check for violation of Delaunay criterion.

       RP=SQRT((XC-X(I))**2 + (YC-Y(I))**2)
c     PRINT*,'I=',i,'RC=',RC,'RP=',RP
       IF(RP.LT..98*RC)THEN
C     Point lies within circumcircle.
        IFLAGD=0
        RETURN
       ENDIF

    1 CONTINUE

      RETURN
      END



      SUBROUTINE CHKD
$INCLUDE grid5.common
C    Checks to see if new point is within the domain of interest.
C    Sum of angles on inside of domain = 360.  Sum of angles outside
C    domain = 0.
C    If IANGLE(1) = 1 and ANGLE(>1) = 0 then the new point is ok.
C    If IANGLE(1) = 0                    then the new point is not ok.
C    If IANGLE(1) = 1 and ANGLE(>1) = 1 then the new point is not ok.

      RTOD=180./3.141592654

C    Initialize point type

      DO 1 K=1,NCB
       IANGLE(K)=0
    1 CONTINUE

C    Sum all angles and set the point type: 1 or 0.

      DO 2 K=1,NCB
       NPTS = NPB(K)-1
       SUM=0.0
       DO 3 I=1,NPTS
        AI=XB(K,I)-XC
        AJ=YB(K,I)-YC
        BI=XB(K,I+1)-XC
        BJ=YB(K,I+1)-YC
        AIJSQ=SQRT(AI*AI+AJ*AJ)
        BIJSQ=SQRT(BI*BI+BJ*BJ)
        ANGLE=(AI*BI+AJ*BJ)/(AIJSQ*BIJSQ)
        IF(ANGLE.GT.1.)THEN
         THEAB=0.0
        ELSE
         THEAB=ACOS(ANGLE)
        ENDIF
        THEAB=THEAB*RTOD
        CROSS=AI*BJ-AJ*BI
        SUM=SUM+SIGN(THEAB,CROSS)
    3 CONTINUE
       IF(ABS(SUM-360.0).LT.1.0) IANGLE(K)=1
    2 CONTINUE

C    Determine if point is in domain of interest.
      IFLAGC=1
      IF(IANGLE(1).EQ.0) THEN
       IFLAGC=0
       RETURN
      ENDIF
      DO 4 K=2,NCB
       IF(IANGLE(K).EQ.1) THEN
        IFLAGC=0
        RETURN
       ENDIF
    4 CONTINUE

      RETURN
      END



      SUBROUTINE ADDPT
$INCLUDE grid5.common
      NPAC=0
    1 CONTINUE
C    Determine cell number with largest aspect ratio
      IF(IRE.EQ.1)CALL ASPECT(IPA)
C    Determine cell number with largest area
      IF(IRE.EQ.2)CALL AREA(IPA)
C    Determine cell number with largest circumcircle
      IF(IRE.EQ.3)CALL CIRCUM(IPA)
C    Determine cell number with largest side ratio
      IF(IRE.EQ.4)CALL SIDE(IPA)
C    Determine cell number with largest side ratio
      IF(IRE.EQ.5)CALL EPI(IPA)
C    Increment number of points total and store coordinates of
C    new point
```

```
      NPTT=NPTT+1
      X(NPTT)=XC
      Y(NPTT)=YC

      print*,'Point number',nptt,' added in cell',ipa
      print*,'at location',xc,yc
C     Determine cells whose circumcircles include the new point
C     and delete those triangles
      CALL DELETE
C     Reconnect the remaining sides to new point
      CALL RECON
      NPAC=NPAC+1
C     Remove bogus cells from local refinement
      CALL RBC

      IF(NPAC.LT.NPA)GO TO 1

      RETURN
      END




      SUBROUTINE ASPECT(IN)
$INCLUDE grid5.common
C---COMPUTE ASPECT RATIO OF EVERY TRIANGLE AND
C---PLACE NEW GRID POINT IN TRIANGLE WITH LARGEST
C---ASPECT RATIO
      AR=0.0
      DO 1 I=1,NEL
      I1=NCELL(4,I)
      I2=NCELL(5,I)
      I3=NCELL(6,I)
      X1=X(I1)
      X2=X(I2)
      X3=X(I3)
      Y1=Y(I1)
      Y2=Y(I2)
      Y3=Y(I3)
      A=SQRT((X1-X2)**2+(Y1-Y2)**2)
      B=SQRT((X2-X3)**2+(Y2-Y3)**2)
      C=SQRT((X3-X1)**2+(Y3-Y1)**2)
      S=.5*(A+B+C)
      ARN=A*B*C/(8.*(S-A)*(S-B)*(S-C))
      IF(ARN.GT.AR)THEN
       A11=X1-X2
       A12=Y1-Y2
       A21=X1-X3
       A22=Y1-Y3
       B1=.5*(X1*X1-X2*X2+Y1*Y1-Y2*Y2)
       B2=.5*(X1*X1-X3*X3+Y1*Y1-Y3*Y3)
       XC=(B1*A22-B2*A12)/(A11*A22-A21*A12)
       YC=(B2*A11-B1*A21)/(A11*A22-A21*A12)
C--CHECK XC AND YC TO BE WITHIN BOUNDARY LIMITS
c      if(xc.gt.2.9)go to 1
       CALL CHKD
```

```
         IF(IFLAGC.EQ.1) THEN
          AR=ARN
          IN=I
          XCC=XC
          YCC=YC
         ENDIF
        ENDIF
    1   CONTINUE

        XC=XCC
        YC=YCC

        RETURN
        END




        SUBROUTINE AREA(IN)
$INCLUDE grid5.common
C---COMPUTE AREA OF EVERY TRIANGLE AND
C---PLACE NEW GRID POINT IN TRIANGLE WITH LARGEST
C---AREA
        TAREA=0.0
        DO 1 I=1,NEL
        I1=NCELL(4,I)
        I2=NCELL(5,I)
        I3=NCELL(6,I)
        X1=X(I1)
        X2=X(I2)
        X3=X(I3)
        Y1=Y(I1)
        Y2=Y(I2)
        Y3=Y(I3)
        A=SQRT((X1-X2)**2+(Y1-Y2)**2)
        B=SQRT((X2-X3)**2+(Y2-Y3)**2)
        C=SQRT((X3-X1)**2+(Y3-Y1)**2)
        S=.5*(A+B+C)
        TAREAN=SQRT(S*(S-A)*(S-B)*(S-C))
        IF(TAREAN.GT.TAREA)THEN
         A11=X1-X2
         A12=Y1-Y2
         A21=X1-X3
         A22=Y1-Y3
         B1=.5*(X1*X1-X2*X2+Y1*Y1-Y2*Y2)
         B2=.5*(X1*X1-X3*X3+Y1*Y1-Y3*Y3)
         XC=(B1*A22-B2*A12)/(A11*A22-A21*A12)
         YC=(B2*A11-B1*A21)/(A11*A22-A21*A12)
C--CHECK XC AND YC TO BE WITHIN BOUNDARY LIMITS
c        if(xc.gt.2.0)go to 1
         CALL CHKD
         IF(IFLAGC.EQ.1) THEN
          TAREA=TAREAN
          IN=I
          XCC=XC
          YCC=YC
         ENDIF
        ENDIF
    1   CONTINUE

        XC=XCC
        YC=YCC
```

```
      RETURN
      END



      SUBROUTINE CIRCUM(IN)
$INCLUDE grid5.common
C---COMPUTE CIRCUMCIRCLE RADIUS OF EVERY TRIANGLE AND
C---PLACE NEW GRID POINT IN TRIANGLE WITH LARGEST
C---CIRCUMCIRCLE RADIUS
      RADC=0.0
      DO 1 I=1,NEL
      I1=NCELL(4,I)
      I2=NCELL(5,I)
      I3=NCELL(6,I)
      X1=X(I1)
      X2=X(I2)
      X3=X(I3)
      Y1=Y(I1)
      Y2=Y(I2)
      Y3=Y(I3)
      A=SQRT((X1-X2)**2+(Y1-Y2)**2)
      B=SQRT((X2-X3)**2+(Y2-Y3)**2)
      C=SQRT((X3-X1)**2+(Y3-Y1)**2)
      S=.5*(A+B+C)
      RADCN=A*B*C/(4.*SQRT(S*(S-A)*(S-B)*(S-C)))
      IF(RADCN.GT.RADC)THEN
       A11=X1-X2
       A12=Y1-Y2
       A21=X1-X3
       A22=Y1-Y3
       B1=.5*(X1*X1-X2*X2+Y1*Y1-Y2*Y2)
       B2=.5*(X1*X1-X3*X3+Y1*Y1-Y3*Y3)
       XC=(B1*A22-B2*A12)/(A11*A22-A21*A12)
       YC=(B2*A11-B1*A21)/(A11*A22-A21*A12)
C--CHECK XC AND YC TO BE WITHIN BOUNDARY LIMITS
c       if(xc.lt.3.0.or.xc.gt.9.0)go to 1
c       if(yc.lt.-0.7.or.yc.gt.0.7)go to 1
c       if(xc.gt.0.55)go to 1
c       if(xc.lt.-.55)go to 1
c       if(yc.gt..55)go to 1
c       if(yc.lt.-.55)go to 1
       if(xc.gt.12.0)go to 1
       if(xc.lt.-12.0)go to 1
       if(yc.gt.12.0)go to 1
       if(yc.lt.-12.0)go to 1
       CALL CHKD
       IF(IFLAGC.EQ.1) THEN
        RADC=RADCN
        IN=I
        XCC=XC
        YCC=YC
       ENDIF
      ENDIF
    1 CONTINUE
```

```
      XC=XCC
      YC=YCC

      RETURN
      END



      SUBROUTINE SIDE(IN)
$INCLUDE grid5.common
C---COMPUTE LENGTH OF TRIANGLE SIDES.  CHOOSE THE TRIANGLE WITH
C---THE GREATEST RATIO OF TOTAL LENGTH TO EACH SIDE.
      SD=0.0
      DO 1 I=1,NEL
      I1=NCELL(4,I)
      I2=NCELL(5,I)
      I3=NCELL(6,I)
      X1=X(I1)
      X2=X(I2)
      X3=X(I3)
      Y1=Y(I1)
      Y2=Y(I2)
      Y3=Y(I3)
      A=SQRT((X1-X2)**2+(Y1-Y2)**2)
      B=SQRT((X2-X3)**2+(Y2-Y3)**2)
      C=SQRT((X3-X1)**2+(Y3-Y1)**2)
      S=A+B+C
      SD3=S/AMIN1(A,B,C)
      IF(SD3.GT.SD)THEN
       A11=X1-X2
       A12=Y1-Y2
       A21=X1-X3
       A22=Y1-Y3
       B1=.5*(X1*X1-X2*X2+Y1*Y1-Y2*Y2)
       B2=.5*(X1*X1-X3*X3+Y1*Y1-Y3*Y3)
       XC=(B1*A22-B2*A12)/(A11*A22-A21*A12)
       YC=(B2*A11-B1*A21)/(A11*A22-A21*A12)
C--CHECK XC AND YC TO BE WITHIN BOUNDARY LIMITS
       CALL CHKD
       IF(IFLAGC.EQ.1) THEN
        SD=SD3
        IN=I
        XCC=XC
        YCC=YC
       ENDIF
      ENDIF
    1 CONTINUE

      XC=XCC
      YC=YCC

      RETURN
      END



      SUBROUTINE EPI(IN)
```

```
$INCLUDE grid5.common
C---Explicit point input from terminal.
    1 CONTINUE
       PRINT*,"Input X coordinate of new point."
       READ(5,*)XC
       PRINT*,"Input Y coordinate of new point."
       READ(5,*)YC
C--CHECK XC AND YC TO BE WITHIN BOUNDARY LIMITS
       CALL CHKD
       IF(IFLAGC.EQ.1) THEN
       PRINT*,"Point is OK."
       ELSE
       PRINT*,"Point lies outside domain; enter new point."
       GO TO 1
       ENDIF

      RETURN
      END




      SUBROUTINE DELETE
$INCLUDE grid5.common
C    Delete cells whose circumcircle includes the new point
      NCE=0
      DO 1 I=1,NEL
C    Compute circumcircle radius of all current triangles
C    and the circumcenter coordinates, RCI, XCI, YCI
      I1=NCELL(4,I)
      I2=NCELL(5,I)
      I3=NCELL(6,I)
      X1=X(I1)
      X2=X(I2)
      X3=X(I3)
      Y1=Y(I1)
      Y2=Y(I2)
      Y3=Y(I3)
      A=SQRT((X1-X2)**2+(Y1-Y2)**2)
      B=SQRT((X2-X3)**2+(Y2-Y3)**2)
      C=SQRT((X3-X1)**2+(Y3-Y1)**2)
      S=.5*(A+B+C)
      RCI=A*B*C/(4.*SQRT(S*(S-A)*(S-B)*(S-C)))
      A11=X1-X2
      A12=Y1-Y2
      A21=X1-X3
      A22=Y1-Y3
      B1=.5*(X1*X1-X2*X2+Y1*Y1-Y2*Y2)
      B2=.5*(X1*X1-X3*X3+Y1*Y1-Y3*Y3)
      XCI=(B1*A22-B2*A12)/(A11*A22-A21*A12)
      YCI=(B2*A11-B1*A21)/(A11*A22-A21*A12)
C    Compute distance from circumcenter of cell I to new point
C    Store the cell number for elimination and reconnection
C    if R < RCI
      R=SQRT((XCI-XC)**2 + (YCI-YC)**2)
      IF(R.LT.RCI)THEN
       NCE=NCE+1
       NCST(NCE)=I
c      print*,"cell to be deleted=",i
      ENDIF
    1 CONTINUE
      RETURN
      END



      SUBROUTINE RECON
$INCLUDE grid5.common
      RTOD=180./3.141592654
C    Divide eliminated region into new triangular cells
C    Store all points from eliminated triangles
      NCR=0
      DO 1 N=1,NCE
       I=NCST(N)
       NCR=NCR+1
       NTR(NCR)=NCELL(4,I)
       NCR=NCR+1
       NTR(NCR)=NCELL(5,I)
       NCR=NCR+1
       NTR(NCR)=NCELL(6,I)
    1 CONTINUE
C   Remove points with repeated indexes
      DO 2 I=1,NCR-1
       IF(NTR(I).NE.-999)THEN
        DO 3 J=I+1,NCR
         IF(NTR(I).EQ.NTR(J))NTR(J)=-999
    3   CONTINUE
       ENDIF
    2 CONTINUE
      NRI=0
      DO 4 I=1,NCR
       IF(NTR(I).NE.-999)THEN
        NRI=NRI+1
        NTR(NRI)=NTR(I)
       ENDIF
    4 CONTINUE
C    Compute the angle that the vector connecting the sorted
C    points with the new point makes with the horizontal
      DO 5 I=1,NRI
       N=NTR(I)
       HYP=SQRT((X(N)-X(NPTT))**2+(Y(N)-Y(NPTT))**2)
       ALPHA(I)=ACOS((X(N)-X(NPTT))/HYP)
       ALPHA(I)=ALPHA(I)*RTOD
       IF(Y(N)-Y(NPTT).LT.0.0)ALPHA(I)=360.-ALPHA(I)
    5 CONTINUE
C    Reconnect points into triangles by connecting the new point and
C    the sorted points with its nearest neighbor in the counter-
C    clockwise direction.  A correction must be made if the nearest
C    neighbor liesabove the 0 degree axis by adding 360 degrees
C    to its angle.
```

```
      DO 6 I=1,NRI
       DELAL1=-360.
       ALPHAI=ALPHA(I)
       DO 7 J=1,NRI
        ALPHAJ=ALPHA(J)
        IF(ALPHAI-ALPHAJ.GT.180.)ALPHAJ=ALPHAJ+360.
        DELAL=ALPHAI-ALPHAJ
        IF(DELAL.LT.0.0)THEN
         IF(DELAL.GT.DELAL1)THEN
          K=J
          DELAL1=DELAL
         ENDIF
        ENDIF
    7  CONTINUE
       NEL=NEL+1
       NCELL(4,NEL)=NTR(I)
       NCELL(5,NEL)=NTR(K)
       NCELL(6,NEL)=NPTT
    6 CONTINUE

      RETURN
      END



      SUBROUTINE RBC
$INCLUDE grid5.common
C     Remove bogus cells

C     Flag bogus cells
      DO 1 N=1,NCE
       I=NCST(N)
       NCELL(4,I)=-999
    1 CONTINUE

C     Restack cells without bogus cells
      NT=0
      DO 2 N=1,NEL
       IF(NCELL(4,N).NE.-999) THEN
        NT=NT+1
        NCELL(4,NT)=NCELL(4,N)
        NCELL(5,NT)=NCELL(5,N)
        NCELL(6,NT)=NCELL(6,N)
       ENDIF
    2 CONTINUE

      NEL=NT
      print*,'Total number of elements=',nel

      RETURN
      END



      SUBROUTINE ORIENT
$INCLUDE grid5.common

C     Check triangle orientation.  Node numbering should be in a
```

```
C    counterclockwise direction(i.e. cross product of edges should
C    be positive).  If cross product is negative exchange any two
C    indexes.
      DO 1 I=1,NEL
       I1=NCELL(4,I)
       I2=NCELL(5,I)
       I3=NCELL(6,I)
       AI=X(I2)-X(I1)
       BI=X(I3)-X(I1)
       AJ=Y(I2)-Y(I1)
       BJ=Y(I3)-Y(I1)
       ACROSSB=AI*BJ - BI*AJ
       IF(ACROSSB.LT.0.0)THEN
        I1C=I2
        I2C=I1
        NCELL(4,I)=I1C
        NCELL(5,I)=I2C
       ENDIF
    1 CONTINUE

      RETURN
      END


      SUBROUTINE REMDUP
$INCLUDE grid5.common
C    Determine duplicate cells and CALL RBC to delete them
      N=0
      DO 1 I=1,NEL-1
       I1=NCELL(4,I)
       I2=NCELL(5,I)
       I3=NCELL(6,I)
       DO 2 IC=I+1,NEL
        I1C=NCELL(4,IC)
        I2C=NCELL(5,IC)
        I3C=NCELL(6,IC)
        IF(I1.EQ.I1C.AND.I2.EQ.I2C.AND.I3.EQ.I3C)THEN
         N=N+1
         NCST(N)=IC
        ELSE
        IF(I1.EQ.I2C.AND.I2.EQ.I3C.AND.I3.EQ.I1C)THEN
         N=N+1
         NCST(N)=IC
        ELSE
        IF(I1.EQ.I3C.AND.I2.EQ.I1C.AND.I3.EQ.I2C)THEN
         N=N+1
         NCST(N)=IC
        ENDIF
        ENDIF
        ENDIF
    2  CONTINUE
    1 CONTINUE

      NCE=N
```

```
C     Call Subroutine RBC to remove flagged cells
       CALL RBC
       RETURN
       END



       SUBROUTINE SIDEAB
$INCLUDE grid5.common
C     Side A-B of solid wall boundary connectivity
       DO 1 K=1,NSB
       DO 2 I=NSWB(K),NSWE(K)-1
        I1=NCO(I)
        I2=NCO(I+1)
        NC=0
        DO 3 IC=1,NEL
         IF(NC.EQ.1) GO TO 2
         I1C=NCELL(4,IC)
         I2C=NCELL(5,IC)
         I3C=NCELL(6,IC)
         IF(I1.EQ.I1C.AND.I2.EQ.I2C.OR.I1.EQ.I2C.AND.I2.EQ.I1C)THEN
          NC=NC+1
         ELSE
         IF(I1.EQ.I2C.AND.I2.EQ.I3C.OR.I1.EQ.I3C.AND.I2.EQ.I2C)THEN
          NC=NC+1
          NCELL(4,IC)=I2C
          NCELL(5,IC)=I3C
          NCELL(6,IC)=I1C
         ENDIF
         IF(I1.EQ.I3C.AND.I2.EQ.I1C.OR.I1.EQ.I1C.AND.I2.EQ.I3C)THEN
          NC=NC+1
          NCELL(4,IC)=I3C
          NCELL(5,IC)=I1C
          NCELL(6,IC)=I2C
         ENDIF
         ENDIF
     3  CONTINUE
     2 CONTINUE
     1 CONTINUE
C     Side A-B of exit boundary connectivity
       DO 4 K=1,NEB
       DO 5 I=NEXB(K),NEXE(K)-1
        I1=NCO(I)
        I2=NCO(I+1)
        NC=0
        DO 6 IC=1,NEL
         IF(NC.EQ.1) GO TO 5
         I1C=NCELL(4,IC)
         I2C=NCELL(5,IC)
         I3C=NCELL(6,IC)
         IF(I1.EQ.I1C.AND.I2.EQ.I2C.OR.I1.EQ.I2C.AND.I2.EQ.I1C)THEN
          NC=NC+1
         ELSE
         IF(I1.EQ.I2C.AND.I2.EQ.I3C.OR.I1.EQ.I3C.AND.I2.EQ.I2C)THEN
          NC=NC+1
          NCELL(4,IC)=I2C
          NCELL(5,IC)=I3C
          NCELL(6,IC)=I1C
         ENDIF
         IF(I1.EQ.I3C.AND.I2.EQ.I1C.OR.I1.EQ.I1C.AND.I2.EQ.I3C)THEN
          NC=NC+1
          NCELL(4,IC)=I3C
          NCELL(5,IC)=I1C
          NCELL(6,IC)=I2C
         ENDIF
         ENDIF
    6   CONTINUE
    5  CONTINUE
    4  CONTINUE
C     Side A-B of inlet boundary connectivity
       DO 7 K=1,NIB
       DO 8 I=NINB(K),NINE(K)-1
        I1=NCO(I)
        I2=NCO(I+1)
        NC=0
        DO 9 IC=1,NEL
         IF(NC.EQ.1) GO TO 8
         I1C=NCELL(4,IC)
         I2C=NCELL(5,IC)
         I3C=NCELL(6,IC)
         IF(I1.EQ.I1C.AND.I2.EQ.I2C.OR.I1.EQ.I2C.AND.I2.EQ.I1C)THEN
          NC=NC+1
         ELSE
         IF(I1.EQ.I2C.AND.I2.EQ.I3C.OR.I1.EQ.I3C.AND.I2.EQ.I2C)THEN
          NC=NC+1
          NCELL(4,IC)=I2C
          NCELL(5,IC)=I3C
          NCELL(6,IC)=I1C
         ENDIF
         IF(I1.EQ.I3C.AND.I2.EQ.I1C.OR.I1.EQ.I1C.AND.I2.EQ.I3C)THEN
          NC=NC+1
          NCELL(4,IC)=I3C
          NCELL(5,IC)=I1C
          NCELL(6,IC)=I2C
         ENDIF
         ENDIF
    9   CONTINUE
    8  CONTINUE
    7  CONTINUE
       RETURN
       END


       SUBROUTINE FACELL
$INCLUDE grid5.common
C     Creates connectivity between cell numbers and face numbers
       N=0
C     Side A-B
       DO 1 I=1,NEL-1
        NC=0
```

```
          I1=NCELL(4,I)
          I2=NCELL(5,I)
          DO 2 IC=I+1,NEL
           IF(NC.EQ.2) GO TO 1
            IF(I1.EQ.NCELL(5,IC).AND.I2.EQ.NCELL(4,IC))THEN
             NC=NC+1
             N=N+1
             NCELL(1,I)=N
             NCELL(1,IC)=N
             NFACE(1,N)=I
             NFACE(2,N)=IC
            ELSE
            IF(I1.EQ.NCELL(6,IC).AND.I2.EQ.NCELL(5,IC))THEN
             NC=NC+1
             N=N+1
             NCELL(1,I)=N
             NCELL(2,IC)=N
             NFACE(1,N)=I
             NFACE(2,N)=IC
            ELSE
            IF(I1.EQ.NCELL(4,IC).AND.I2.EQ.NCELL(6,IC))THEN
             NC=NC+1
             N=N+1
             NCELL(1,I)=N
             NCELL(3,IC)=N
             NFACE(1,N)=I
             NFACE(2,N)=IC
            ENDIF
            ENDIF
            ENDIF
      2   CONTINUE
      1 CONTINUE
C      Side B-C
        DO 3 I=1,NEL-1
          NC=0
          I2=NCELL(5,I)
          I3=NCELL(6,I)
          DO 4 IC=I+1,NEL
           IF(NC.EQ.2) GO TO 3
            IF(I2.EQ.NCELL(5,IC).AND.I3.EQ.NCELL(4,IC))THEN
             NC=NC+1
             N=N+1
             NCELL(2,I)=N
             NCELL(1,IC)=N
             NFACE(1,N)=I
             NFACE(2,N)=IC
            ELSE
            IF(I2.EQ.NCELL(6,IC).AND.I3.EQ.NCELL(5,IC))THEN
             NC=NC+1
             N=N+1
             NCELL(2,I)=N
             NCELL(2,IC)=N
             NFACE(1,N)=I
             NFACE(2,N)=IC
            ELSE
            IF(I2.EQ.NCELL(4,IC).AND.I3.EQ.NCELL(6,IC))THEN
             NC=NC+1
          N=N+1
          NCELL(2,I)=N
          NCELL(3,IC)=N
          NFACE(1,N)=I
          NFACE(2,N)=IC
         ENDIF
         ENDIF
         ENDIF
    4  CONTINUE
    3 CONTINUE
C     Side C-A
      DO 5 I=1,NEL-1
       NC=0
       I3=NCELL(6,I)
       I4=NCELL(4,I)
       DO 6 IC=I+1,NEL
        IF(NC.EQ.2) GO TO 5
         IF(I3.EQ.NCELL(5,IC).AND.I4.EQ.NCELL(4,IC))THEN
          NC=NC+1
          N=N+1
          NCELL(3,I)=N
          NCELL(1,IC)=N
          NFACE(1,N)=I
          NFACE(2,N)=IC
         ELSE
         IF(I3.EQ.NCELL(6,IC).AND.I4.EQ.NCELL(5,IC))THEN
          NC=NC+1
          N=N+1
          NCELL(3,I)=N
          NCELL(2,IC)=N
          NFACE(1,N)=I
          NFACE(2,N)=IC
         ELSE
         IF(I3.EQ.NCELL(4,IC).AND.I4.EQ.NCELL(6,IC))THEN
          NC=NC+1
          N=N+1
          NCELL(3,I)=N
          NCELL(3,IC)=N
          NFACE(1,N)=I
          NFACE(2,N)=IC
         ENDIF
         ENDIF
         ENDIF
    6  CONTINUE
    5 CONTINUE
C     Side A-B of solid wall boundary connectivity
      DO 107 K=1,NSB
      DO 7 I=NSWB(K),NSWE(K)-1
       I1=NCO(I)
       I2=NCO(I+1)
       NC=0
       DO 8 IC=1,NEL
        IF(NC.EQ.1) GO TO 7
        I1C=NCELL(4,IC)
        I2C=NCELL(5,IC)
        IF(I1.EQ.I1C.AND.I2.EQ.I2C.OR.I1.EQ.I2C.AND.I2.EQ.I1C)THEN
         NC=NC+1
```

```
          N=N+1
          NCELL(1,IC)=N
          NFACE(1,N)=IC
          NFACE(2,N)=0
         ENDIF
    8   CONTINUE
    7  CONTINUE
  107  CONTINUE
C     Side A-B of exit boundary connectivity
       DO 109 K=1,NEB
       DO 9 I=NEXB(K),NEXE(K)-1
        I1=NCO(I)
        I2=NCO(I+1)
        NC=0
        DO 10 IC=1,NEL
         IF(NC.EQ.1) GO TO 9
         I1C=NCELL(4,IC)
         I2C=NCELL(5,IC)
         IF(I1.EQ.I1C.AND.I2.EQ.I2C)THEN
          NC=NC+1
          N=N+1
          NCELL(1,IC)=N
          NFACE(1,N)=IC
          NFACE(2,N)=-2
         ENDIF
   10   CONTINUE
    9  CONTINUE
  109  CONTINUE
C     Side A-B of inlet boundary connectivity
       DO 111 K=1,NIB
       DO 11 I=NINB(K),NINE(K)-1
        I1=NCO(I)
        I2=NCO(I+1)
        NC=0
        DO 12 IC=1,NEL
         IF(NC.EQ.1) GO TO 11
         I1C=NCELL(4,IC)
         I2C=NCELL(5,IC)
         IF(I1.EQ.I1C.AND.I2.EQ.I2C)THEN
          NC=NC+1
          N=N+1
          NCELL(1,IC)=N
          NFACE(1,N)=IC
          NFACE(2,N)=-1
         ENDIF
   12   CONTINUE
   11  CONTINUE
  111  CONTINUE
C     Side A-B of symmetric boundary connectivity
C     NSYP is the node number that is periodic with the first
C     index number, NSYB.
       DO 113 K=1,NYB
       NP=NSYP(K)
       DO 13 I=NSYB(K),NSYE(K)-1
        I1=NCO(I)
        I2=NCO(I+1)
```

```
      NP=NP-1
      I1P=NCO(NP)
      I2P=NCO(NP+1)
      NC=0
      DO 14 IC=1,NEL
       IF(NC.EQ.1) GO TO 15
       I1C=NCELL(4,IC)
       I2C=NCELL(5,IC)
       I3C=NCELL(6,IC)
       IF(I1.EQ.I1C.AND.I2.EQ.I2C)THEN
        NC=NC+1
        N=N+1
        NCELL(1,IC)=N
        NFACE(1,N)=IC
       ELSE
       IF(I1.EQ.I2C.AND.I2.EQ.I3C)THEN
        NC=NC+1
        N=N+1
        NCELL(2,IC)=N
        NFACE(1,N)=IC
       ELSE
       IF(I1.EQ.I3C.AND.I2.EQ.I1C)THEN
        NC=NC+1
        N=N+1
        NCELL(3,IC)=N
        NFACE(1,N)=IC
       ENDIF
       ENDIF
       ENDIF
 14   CONTINUE
 15   CONTINUE
      DO 16 IC=1,NEL
       IF(NC.EQ.2) GO TO 13
       I1C=NCELL(4,IC)
       I2C=NCELL(5,IC)
       I3C=NCELL(6,IC)
       IF(I1P.EQ.I1C.AND.I2P.EQ.I2C)THEN
        NC=NC+1
        NCELL(1,IC)=N
        NFACE(2,N)=IC
       ELSE
       IF(I1P.EQ.I2C.AND.I2P.EQ.I3C)THEN
        NC=NC+1
        NCELL(2,IC)=N
        NFACE(2,N)=IC
       ELSE
       IF(I1P.EQ.I3C.AND.I2P.EQ.I1C)THEN
        NC=NC+1
        NCELL(3,IC)=N
        NFACE(2,N)=IC
       ELSE
       ENDIF
       ENDIF
       ENDIF
 16   CONTINUE
 13  CONTINUE
113  CONTINUE

     NFT=N
```

```
      RETURN
      END



      SUBROUTINE BPLOT
$INCLUDE grid5.common
      DO 1 I=1,NPB(1)
      WRITE(31,*)XB(1,I),YB(1,I)
    1 CONTINUE
      DO 2 I=1,NPB(2)
      WRITE(32,*)XB(2,I),YB(2,I)
    2 CONTINUE
      DO 3 I=1,NPB(3)
      WRITE(33,*)XB(3,I),YB(3,I)
    3 CONTINUE
C---Output elements for gridpl plotting
      WRITE(50,40)NEL
      DO 20 I=1,NEL
       NAB=NCELL(1,I)
       NBC=NCELL(2,I)
       NCA=NCELL(3,I)
       NBTAB=NFACE(1,NAB)+NFACE(2,NAB)-I
       NBTBC=NFACE(1,NBC)+NFACE(2,NBC)-I
       NBTCA=NFACE(1,NCA)+NFACE(2,NCA)-I
       I1=NCELL(4,I)
       I2=NCELL(5,I)
       I3=NCELL(6,I)
      WRITE(50,50)X(I1),Y(I1),X(I2),Y(I2),X(I3),Y(I3)
      WRITE(50,55)NBTAB,NBTBC,NBTCA
   20 CONTINUE
   40 FORMAT(3I7)
   50 FORMAT(6E13.6)
   55 FORMAT(3I6)
      RETURN
      END



      SUBROUTINE RESTRT(NRW)
$INCLUDE grid5.common
      REWIND(20)
      IF(NRW.EQ.1)THEN
      READ(20,*)NPTT,NEL
      DO 1 I=1,NPTT
      READ(20,*)X(I),Y(I)
    1 CONTINUE
      DO 2 I=1,NEL
      READ(20,*)(NCELL(K,I),K=4,6)
    2 CONTINUE
      ENDIF

      REWIND(20)

      IF(NRW.EQ.2)THEN
      WRITE(20,*)NPTT,NEL
      DO 3 I=1,NPTT
      WRITE(20,*)X(I),Y(I)
    3 CONTINUE
      DO 4 I=1,NEL
      WRITE(20,*)(NCELL(K,I),K=4,6)
    4 CONTINUE
      ENDIF

      REWIND(20)

      RETURN
      END



      SUBROUTINE OUTPUT
$INCLUDE grid5.common

C  Write out face data

      WRITE(40,10)
      WRITE(40,11)

      WRITE(40,12)NFT
      WRITE(40,13)
      DO 1 I=1,NFT
       WRITE(40,14)I,(NFACE(K,I),K=1,2)
    1 CONTINUE

C  Write out cell data

      WRITE(40,15)
      WRITE(40,16)
      WRITE(40,17)NEL
      WRITE(40,18)
      DO 2 I=1,NEL
       WRITE(40,19)I,(NCELL(K,I),K=1,6)
    2 CONTINUE

C  Write out node data

      WRITE(45,20)
      WRITE(45,21)
      WRITE(45,22)NPTT
      WRITE(45,23)
      DO 3 I=1,NPTT
       WRITE(45,24)I,X(I),Y(I)
    3 CONTINUE
```

```
   10 FORMAT(2X,'FACE DATA')
   11 FORMAT(2X,'TOTAL NUMBER OF FACES')
   12 FORMAT(I10)
   13 FORMAT(4X,'FACE',4X,'CELL1',4X,'CELL2')
   14 FORMAT(2X,I5,5X,I5,5X,I5)
   15 FORMAT(2X,'CELL DATA')
   16 FORMAT(2X,'TOTAL NUMBER OF CELLS')
   17 FORMAT(I10)
   18 FORMAT(5X,'CELL',13X,'FACE NUMBERS',19X,
     .'NODE NUMBERS')
   19 FORMAT(7I10)
   20 FORMAT(2X,'NODES')
   21 FORMAT(2X,'TOTAL NUMBER OF NODES=')
   22 FORMAT(I10)
   23 FORMAT(3X,'NODE',8X,'X',10X,'Y')
   24 FORMAT(2X,I4,2F13.6)
      RETURN
      END



      SUBROUTINE BFC
$INCLUDE grid5.common
      DIMENSION NF(4),NCEFF(2)
      NCEF=0
      DO 1 N=1,NEL
      NBCC=0
       DO 2 K=1,3
       NFC=NCELL(K,N)
       NCC=NFACE(1,NFC)+NFACE(2,NFC)-N
       IF(NCC.LE.0)THEN
        NBCC=NBCC+1
       ENDIF
    2  CONTINUE
C   Switch diagonal face of cells with more than one boundary face.
      IF(NBCC.GT.1)THEN
      NCEF=NCEF+1
      NF(1)=NCELL(4,N)
      NF(2)=NCELL(5,N)
      NF(3)=NCELL(6,N)
      NF(4)=NCELL(4,N)
C   Determine common face and node numbers
      DO 3 K=1,3
      NFC=NCELL(K,N)
      NCC=NFACE(1,NFC)+NFACE(2,NFC)-N
      IF(NCC.GT.0)THEN
       NFCOM=NFC
       NCCOM=NCC
       NODE1=NF(K)
       NODE2=NF(K+1)
      ENDIF
    3 CONTINUE
      NCEFF(1)=N
      NCEFF(2)=NCCOM
      NODE3=NCELL(4,NCCOM)+NCELL(5,NCCOM)+NCELL(6,NCCOM)-NODE1-NODE2
C  Renumber the cells with the new diagonal
      M=0
      DO 4 K=1,3
      NFC=NCELL(K,N)
      NCC=NFACE(1,NFC)+NFACE(2,NFC)-N
      IF(NCC.LE.0)THEN
       M=M+1
       NCEX=NCEFF(M)
       NCELL(4,NCEX)=NF(K)
       NCELL(5,NCEX)=NF(K+1)
       NCELL(6,NCEX)=NODE3
      ENDIF
    4 CONTINUE

      ENDIF
    1 CONTINUE
      PRINT*,'NUMBER OF CELLS THAT EXCHANGED FACES=',NCEF
C   Recheck cell orientation and generate connectivity arrays if any
C   cells exchanged diagonal faces.
      IF(NCEF.GT.0)THEN
       CALL ORIENT
       CALL FACELL
      ENDIF
      RETURN
      END



      SUBROUTINE FCM
$INCLUDE grid5.common
C       Input ordering array for ordering extra viscous triangles
      NPERM(1,1)=1
      NPERM(1,2)=2
      NPERM(1,3)=3
      NPERM(2,1)=2
      NPERM(2,2)=3
      NPERM(2,3)=1
      NPERM(3,1)=3
      NPERM(3,2)=1
      NPERM(3,3)=2
C   Initialize color
      DO 1 K=1,4
      DO 1 I=1,NEL
       NCOLOR(K,I)=0
    1 CONTINUE
      DO 2 I=1,NEL
       NCELL(7,I)=0
    2 CONTINUE
C   Blue
      NBLUE=0
      K=1
      DO 3 I=1,NEL
C   Faces of cell I
      NF1=NCELL(1,I)
      NF2=NCELL(2,I)
      NF3=NCELL(3,I)
C   Adjacent cells to cell I
      ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
      ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
      ICD=NFACE(1,NF3) + NFACE(2,NF3) - I

      ICA=I

C-----Find Cell #'s of E, F, G, H, I, J
C------Faces of Cell B
      IF(ICB.GT.0)THEN
      NFS1=NCELL(1,ICB)
      NFS2=NCELL(2,ICB)
      NFS3=NCELL(3,ICB)
C------Cells surrounding cell B (E, F)
C------Determine cell numbers in the order A-E-F
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICB
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICB
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICB
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICE=ICS(LAD)
      ICF=ICS(LDB)

      ELSE
       ICE=ICB
       ICF=ICB
      ENDIF

C------Faces of Cell C
      NFS1=NCELL(1,ICC)
      NFS2=NCELL(2,ICC)
      NFS3=NCELL(3,ICC)
C------Cells surrounding cell C (G, H)
C------Determine cell numbers in the order A-G-H
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICC
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICC
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICC
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICG=ICS(LAD)
      ICH=ICS(LDB)

C------Faces of Cell D
      NFS1=NCELL(1,ICD)
      NFS2=NCELL(2,ICD)
      NFS3=NCELL(3,ICD)
C------Cells surrounding cell D (I, J)
C------Determine cell numbers in the order A-I-J
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICD
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICD
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICD
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICI=ICS(LAD)
      ICJ=ICS(LDB)

      NCOLA=1

      IF(ICB.LE.0)THEN
       NCOLB=0
       NCOLE=0
       NCOLF=0
      ELSE
       NCOLB=NCELL(7,ICB)
       NCOLE=NCELL(7,ICE)
       NCOLF=NCELL(7,ICF)
      ENDIF
      IF(ICE.LE.0)THEN
       NCOLE=0
      ELSE
       NCOLE=NCELL(7,ICE)
      ENDIF
      IF(ICF.LE.0)THEN
       NCOLF=0
      ELSE
       NCOLF=NCELL(7,ICF)
      ENDIF
       NCOLC=NCELL(7,ICC)
       NCOLG=NCELL(7,ICG)
       NCOLH=NCELL(7,ICH)
       NCOLD=NCELL(7,ICD)
       NCOLI=NCELL(7,ICI)
       NCOLJ=NCELL(7,ICJ)

      IF(NCOLA.NE.NCOLB)THEN
       IF(NCOLA.NE.NCOLC)THEN
        IF(NCOLA.NE.NCOLD)THEN
        IF(NCOLA.NE.NCOLE)THEN
        IF(NCOLA.NE.NCOLF)THEN
        IF(NCOLA.NE.NCOLG)THEN
        IF(NCOLA.NE.NCOLH)THEN
        IF(NCOLA.NE.NCOLI)THEN
        IF(NCOLA.NE.NCOLJ)THEN
        NBLUE=NBLUE+1
        NCOLOR(K,NBLUE)=I
        NCELL(7,I)=1
        ENDIF
        ENDIF
```

```
            ENDIF
            ENDIF
            ENDIF
            ENDIF
            ENDIF
           ENDIF
          ENDIF
     3 CONTINUE
          NGREEN=0
          NRED=0
          NYELLOW=0
          DO 5 I=1,NEL
          IF(NCELL(7,I).EQ.0)THEN
C    Faces of cell I
          NF1=NCELL(1,I)
          NF2=NCELL(2,I)
          NF3=NCELL(3,I)
C    Adjacent cells to cell I
          ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
          ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
          ICD=NFACE(1,NF3) + NFACE(2,NF3) - I
          ICA=I
C-----Find Cell #'s of E, F, G, H, I, J
C------Faces of Cell B
          IF(ICB.GT.0)THEN
          NFS1=NCELL(1,ICB)
          NFS2=NCELL(2,ICB)
          NFS3=NCELL(3,ICB)
C------Cells surrounding cell B (E, F)
C------Determine cell numbers in the order A-E-F
          ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICB
          ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICB
          ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICB
           NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
           NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
           NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
           LAB=NFA*1 + NFB*2 + NFC*3
           LAD=NPERM(LAB,2)
           LDB=NPERM(LAB,3)
          ICE=ICS(LAD)
          ICF=ICS(LDB)
          ELSE
           ICE=ICB
           ICF=ICB
          ENDIF
C------Faces of Cell C
          NFS1=NCELL(1,ICC)
          NFS2=NCELL(2,ICC)
          NFS3=NCELL(3,ICC)
C------Cells surrounding cell C (G, H)
C------Determine cell numbers in the order A-G-H
          ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICC
          ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICC
```

```
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICC
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICG=ICS(LAD)
      ICH=ICS(LDB)

C------Faces of Cell D
      NFS1=NCELL(1,ICD)
      NFS2=NCELL(2,ICD)
      NFS3=NCELL(3,ICD)
C------Cells surrounding cell D (I, J)
C------Determine cell numbers in the order A-I-J
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICD
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICD
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICD
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICI=ICS(LAD)
      ICJ=ICS(LDB)

      NCOLA=2

      IF(ICB.LE.0)THEN
       NCOLB=0
       NCOLE=0
       NCOLF=0
      ELSE
       NCOLB=NCELL(7,ICB)
       NCOLE=NCELL(7,ICE)
       NCOLF=NCELL(7,ICF)
      ENDIF
      IF(ICE.LE.0)THEN
       NCOLE=0
      ELSE
       NCOLE=NCELL(7,ICE)
      ENDIF
      IF(ICF.LE.0)THEN
       NCOLF=0
      ELSE
       NCOLF=NCELL(7,ICF)
      ENDIF
       NCOLC=NCELL(7,ICC)
       NCOLG=NCELL(7,ICG)
       NCOLH=NCELL(7,ICH)
       NCOLD=NCELL(7,ICD)
       NCOLI=NCELL(7,ICI)
       NCOLJ=NCELL(7,ICJ)

      IF(NCOLA.NE.NCOLB)THEN
       IF(NCOLA.NE.NCOLC)THEN
```

```
        IF(NCOLA.NE.NCOLD)THEN
        IF(NCOLA.NE.NCOLE)THEN
        IF(NCOLA.NE.NCOLF)THEN
        IF(NCOLA.NE.NCOLG)THEN
        IF(NCOLA.NE.NCOLH)THEN
        IF(NCOLA.NE.NCOLI)THEN
        IF(NCOLA.NE.NCOLJ)THEN
         NGREEN=NGREEN+1
         NCOLOR(2,NGREEN)=I
         NCELL(7,I)=2
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        ENDIF
       ENDIF
      ENDIF

      ENDIF
    5 CONTINUE
      DO 6 I=1,NEL
      IF(NCELL(7,I).EQ.0)THEN
C   Faces of cell I
      NF1=NCELL(1,I)
      NF2=NCELL(2,I)
      NF3=NCELL(3,I)
C   Adjacent cells to cell I
      ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
      ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
      ICD=NFACE(1,NF3) + NFACE(2,NF3) - I
      ICA=I
C-----Find Cell #'s of E, F, G, H, I, J
C------Faces of Cell B
      IF(ICB.GT.0)THEN
      NFS1=NCELL(1,ICB)
      NFS2=NCELL(2,ICB)
      NFS3=NCELL(3,ICB)
C------Cells surrounding cell B (E, F)
C------Determine cell numbers in the order A-E-F
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICB
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICB
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICB
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICE=ICS(LAD)
      ICF=ICS(LDB)

      ELSE
       ICE=ICB
       ICF=ICB
      ENDIF
C------Faces of Cell C
      NFS1=NCELL(1,ICC)
      NFS2=NCELL(2,ICC)
      NFS3=NCELL(3,ICC)
C------Cells surrounding cell C (G, H)
C------Determine cell numbers in the order A-G-H
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICC
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICC
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICC
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICG=ICS(LAD)
      ICH=ICS(LDB)

C------Faces of Cell D
      NFS1=NCELL(1,ICD)
      NFS2=NCELL(2,ICD)
      NFS3=NCELL(3,ICD)
C------Cells surrounding cell D (I, J)
C------Determine cell numbers in the order A-I-J
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICD
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICD
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICD
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICI=ICS(LAD)
      ICJ=ICS(LDB)

      NCOLA=3

      IF(ICB.LE.0)THEN
       NCOLB=0
       NCOLE=0
       NCOLF=0
      ELSE
       NCOLB=NCELL(7,ICB)
       NCOLE=NCELL(7,ICE)
       NCOLF=NCELL(7,ICF)
      ENDIF
      IF(ICE.LE.0)THEN
       NCOLE=0
      ELSE
       NCOLE=NCELL(7,ICE)
      ENDIF
      IF(ICF.LE.0)THEN
       NCOLF=0
      ELSE
       NCOLF=NCELL(7,ICF)
      ENDIF
       NCOLC=NCELL(7,ICC)
       NCOLG=NCELL(7,ICG)
       NCOLH=NCELL(7,ICH)
       NCOLD=NCELL(7,ICD)
       NCOLI=NCELL(7,ICI)
       NCOLJ=NCELL(7,ICJ)
      IF(NCOLA.NE.NCOLB)THEN
       IF(NCOLA.NE.NCOLC)THEN
        IF(NCOLA.NE.NCOLD)THEN
        IF(NCOLA.NE.NCOLE)THEN
        IF(NCOLA.NE.NCOLF)THEN
        IF(NCOLA.NE.NCOLG)THEN
        IF(NCOLA.NE.NCOLH)THEN
        IF(NCOLA.NE.NCOLI)THEN
        IF(NCOLA.NE.NCOLJ)THEN
         NRED=NRED+1
         NCOLOR(3,NRED)=I
         NCELL(7,I)=3
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        ENDIF
       ENDIF
      ENDIF
      ENDIF
    6 CONTINUE
      DO 7 I=1,NEL
      IF(NCELL(7,I).EQ.0)THEN
C   Faces of cell I
      NF1=NCELL(1,I)
      NF2=NCELL(2,I)
      NF3=NCELL(3,I)
C   Adjacent cells to cell I
      ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
      ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
      ICD=NFACE(1,NF3) + NFACE(2,NF3) - I
      ICA=I
C-----Find Cell #'s of E, F, G, H, I, J
C------Faces of Cell B
      IF(ICB.GT.0)THEN
      NFS1=NCELL(1,ICB)
      NFS2=NCELL(2,ICB)
      NFS3=NCELL(3,ICB)
C------Cells surrounding cell B (E, F)
C------Determine cell numbers in the order A-E-F
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICB
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICB
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICB
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
```

```
         NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
         NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
         LAB=NFA*1 + NFB*2 + NFC*3
         LAD=NPERM(LAB,2)
         LDB=NPERM(LAB,3)
        ICE=ICS(LAD)
        ICF=ICS(LDB)

        ELSE
         ICE=ICB
         ICF=ICB
        ENDIF
C------Faces of Cell C
        NFS1=NCELL(1,ICC)
        NFS2=NCELL(2,ICC)
        NFS3=NCELL(3,ICC)
C------Cells surrounding cell C (G, H)
C------Determine cell numbers in the order A-G-H
        ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICC
        ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICC
        ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICC
         NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
         NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
         NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
         LAB=NFA*1 + NFB*2 + NFC*3
         LAD=NPERM(LAB,2)
         LDB=NPERM(LAB,3)
        ICG=ICS(LAD)
        ICH=ICS(LDB)

C------Faces of Cell D
        NFS1=NCELL(1,ICD)
        NFS2=NCELL(2,ICD)
        NFS3=NCELL(3,ICD)
C------Cells surrounding cell D (I, J)
C------Determine cell numbers in the order A-I-J
        ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICD
        ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICD
        ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICD
         NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
         NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
         NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
         LAB=NFA*1 + NFB*2 + NFC*3
         LAD=NPERM(LAB,2)
         LDB=NPERM(LAB,3)
        ICI=ICS(LAD)
        ICJ=ICS(LDB)

        NCOLA=4

        IF(ICB.LE.0)THEN
         NCOLB=0
         NCOLE=0
         NCOLF=0
        ELSE
         NCOLB=NCELL(7,ICB)
         NCOLE=NCELL(7,ICE)
         NCOLF=NCELL(7,ICF)
        ENDIF
        IF(ICE.LE.0)THEN
         NCOLE=0
        ELSE
         NCOLE=NCELL(7,ICE)
        ENDIF
        IF(ICF.LE.0)THEN
         NCOLF=0
        ELSE
         NCOLF=NCELL(7,ICF)
        ENDIF
         NCOLC=NCELL(7,ICC)
         NCOLG=NCELL(7,ICG)
         NCOLH=NCELL(7,ICH)
         NCOLD=NCELL(7,ICD)
         NCOLI=NCELL(7,ICI)
         NCOLJ=NCELL(7,ICJ)

        IF(NCOLA.NE.NCOLB)THEN
         IF(NCOLA.NE.NCOLC)THEN
          IF(NCOLA.NE.NCOLD)THEN
          IF(NCOLA.NE.NCOLE)THEN
          IF(NCOLA.NE.NCOLF)THEN
          IF(NCOLA.NE.NCOLG)THEN
          IF(NCOLA.NE.NCOLH)THEN
          IF(NCOLA.NE.NCOLI)THEN
          IF(NCOLA.NE.NCOLJ)THEN
           NYELLOW=NYELLOW+1
           NCOLOR(4,NYELLOW)=I
           NCELL(7,I)=4
          ENDIF
          ENDIF
          ENDIF
          ENDIF
          ENDIF
          ENDIF
          ENDIF
         ENDIF
        ENDIF

        ENDIF
      7 CONTINUE
C     Yellow
        DO 11 I=1,NEL
        IF(NCELL(7,I).EQ.0)THEN
C     Faces of cell I
        NF1=NCELL(1,I)
        NF2=NCELL(2,I)
        NF3=NCELL(3,I)
C     Adjacent cells to cell I
        ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
        ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
        ICD=NFACE(1,NF3) + NFACE(2,NF3) - I

        NCOLA=4

        IF(ICB.LE.0)THEN
```

```
          NCOLB=0
         ELSE
          NCOLB=NCELL(7,ICB)
         ENDIF
          NCOLC=NCELL(7,ICC)
          NCOLD=NCELL(7,ICD)
C     Check surrounding cells for color
         IF(NCOLA.NE.NCOLB)THEN
          IF(NCOLA.NE.NCOLC)THEN
           IF(NCOLA.NE.NCOLD)THEN
           NYELLOW=NYELLOW+1
           NCOLOR(4,NYELLOW)=I
           NCELL(7,I)=4
           ENDIF
          ENDIF
         ENDIF

         ENDIF

      11 CONTINUE

C     Red
         DO 10 I=1,NEL
         IF(NCELL(7,I).EQ.0)THEN

C     Faces of cell I
         NF1=NCELL(1,I)
         NF2=NCELL(2,I)
         NF3=NCELL(3,I)
C     Adjacent cells to cell I
         ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
         ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
         ICD=NFACE(1,NF3) + NFACE(2,NF3) - I

         NCOLA=3

         IF(ICB.LE.0)THEN
          NCOLB=0
         ELSE
          NCOLB=NCELL(7,ICB)
         ENDIF
          NCOLC=NCELL(7,ICC)
          NCOLD=NCELL(7,ICD)
C     Check surrounding cells for color
         IF(NCOLA.NE.NCOLB)THEN
          IF(NCOLA.NE.NCOLC)THEN
           IF(NCOLA.NE.NCOLD)THEN
           NRED=NRED+1
           NCOLOR(3,NRED)=I
           NCELL(7,I)=3
           ENDIF
          ENDIF
         ENDIF

         ENDIF

      10 CONTINUE

C     Green
         DO 9 I=1,NEL
         IF(NCELL(7,I).EQ.0)THEN
```

```
C   Faces of cell I
      NF1=NCELL(1,I)
      NF2=NCELL(2,I)
      NF3=NCELL(3,I)
C   Adjacent cells to cell I
      ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
      ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
      ICD=NFACE(1,NF3) + NFACE(2,NF3) - I

      NCOLA=2

      IF(ICB.LE.0)THEN
       NCOLB=0
      ELSE
       NCOLB=NCELL(7,ICB)
      ENDIF
       NCOLC=NCELL(7,ICC)
       NCOLD=NCELL(7,ICD)
C   Check surrounding cells for color
      IF(NCOLA.NE.NCOLB)THEN
       IF(NCOLA.NE.NCOLC)THEN
        IF(NCOLA.NE.NCOLD)THEN
        NGREEN=NGREEN+1
        NCOLOR(2,NGREEN)=I
        NCELL(7,I)=2
        ENDIF
       ENDIF
      ENDIF

      ENDIF

    9 CONTINUE

C   Blue
      DO 8 I=1,NEL
      IF(NCELL(7,I).EQ.0)THEN

C   Faces of cell I
      NF1=NCELL(1,I)
      NF2=NCELL(2,I)
      NF3=NCELL(3,I)
C   Adjacent cells to cell I
      ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
      ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
      ICD=NFACE(1,NF3) + NFACE(2,NF3) - I

      NCOLA=1

      IF(ICB.LE.0)THEN
       NCOLB=0
      ELSE
       NCOLB=NCELL(7,ICB)
      ENDIF
       NCOLC=NCELL(7,ICC)
       NCOLD=NCELL(7,ICD)
C   Check surrounding cells for color
      IF(NCOLA.NE.NCOLB)THEN
       IF(NCOLA.NE.NCOLC)THEN
        IF(NCOLA.NE.NCOLD)THEN
```

```
          NBLUE=NBLUE+1
          NCOLOR(1,NBLUE)=I
          NCELL(7,I)=1
          ENDIF
         ENDIF
        ENDIF

        ENDIF
    8 CONTINUE
C   Check cells
        NCHECK=0
        DO 12 I=1,NEL
C   Faces of cell I
        NF1=NCELL(1,I)
        NF2=NCELL(2,I)
        NF3=NCELL(3,I)
C   Adjacent cells to cell I
        ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
        ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
        ICD=NFACE(1,NF3) + NFACE(2,NF3) - I

        NCOLA=NCELL(7,I)

        IF(ICB.LE.0)THEN
         NCOLB=0
        ELSE
         NCOLB=NCELL(7,ICB)
        ENDIF
         NCOLC=NCELL(7,ICC)
         NCOLD=NCELL(7,ICD)
C   Check surrounding cells for color
        IF(NCOLA.EQ.NCOLB)NCHECK=NCHECK+1
        IF(NCOLA.EQ.NCOLD)NCHECK=NCHECK+1
        IF(NCOLA.EQ.NCOLC)NCHECK=NCHECK+1

   12 CONTINUE

        IF(NCHECK.EQ.0)PRINT*,'Cell colorings are OK'
        IF(NCHECK.GT.0)PRINT*,'There are',ncheck,
       .' cells that are adjacent'

        K=1
        WRITE(55,140)NBLUE
        DO 100 I=1,NBLUE
        WRITE(55,150)NCOLOR(K,I)
  100 CONTINUE
        K=2
        WRITE(55,140)NGREEN
        DO 110 I=1,NGREEN
        WRITE(55,150)NCOLOR(K,I)
  110 CONTINUE
        K=3
        WRITE(55,140)NRED
        DO 120 I=1,NRED
        WRITE(55,150)NCOLOR(K,I)
  120 CONTINUE
```

```
      K=4
      WRITE(55,140)NYELLOW
      DO 130 I=1,NYELLOW
      WRITE(55,150)NCOLOR(K,I)
  130 CONTINUE

  140 FORMAT(I10)
  150 FORMAT(I20)

      RETURN
      END
```

APPENDIX D. VISCOUS FLOW COMPUTER CODE

This computer program solves the two-dimensional Navier-Stokes equations on a triangular unstructured grid. The flow input parameters are described in the subroutine INPUT. The triangular grid geometry and connectivity are read in as input in the subroutine GRIDIN. This code contains vectorization commands specific to the Cray computer.

```
*DECK UNSTF
      PROGRAM UNSTF
*CALL COMMZ
      OPEN(UNIT=1,STATUS='OLD',FILE='/wrk/aejorgen/plot.unsfs')
      OPEN(UNIT=10,STATUS='OLD',FILE='/wrk/aejorgen/input.viscous')
      OPEN(UNIT=15,STATUS='OLD',FILE='/wrk/aejorgen/output.viscous')
      OPEN(UNIT=20,STATUS='OLD',FILE='/wrk/aejorgen/facell.data')
      OPEN(UNIT=23,STATUS='OLD',FILE='/wrk/aejorgen/node.data')
      OPEN(UNIT=26,STATUS='OLD',FILE='/wrk/aejorgen/color.map')
      OPEN(UNIT=30,STATUS='OLD',FILE='/wrk/aejorgen/unst.data')
      OPEN(UNIT=35,STATUS='OLD',FORM='UNFORMATTED',
     . FILE='/wrk/aejorgen/rest.data')
      OPEN(UNIT=40,STATUS='OLD',FILE='/wrk/aejorgen/press.plot')
      OPEN(UNIT=50,STATUS='OLD',FILE='/wrk/aejorgen/error.file')
      OPEN(UNIT=60,STATUS='OLD',FILE='/wrk/aejorgen/res.plot')
      CALL INPUT
      CALL INITV
      CALL CONNECT
      IF(NSOLVE.EQ.2)CALL BMCO
      NCOUNT=0
      NRES=0
      IF(NRST.GT.0)THEN
      CALL REREAD
      ELSE
c    initial conditions for fully developed flow(primitive)
      CALL INITFD
      CALL NONDIM
      CALL VOLUME
      CALL TIMST
      ENDIF
      CALL OUTPUT
      DO 1 NTS=1,NTTS
      NCOUNT=NCOUNT+1
      PRINT*,'TOTAL ITERATION COUNT =',NCOUNT
      LINITR=0
    2 ERR=0.0
      RES1=0.0
      RES2=0.0
      RES3=0.0
      RES4=0.0
      LINITR=LINITR+1
      PRINT*,'Linearization iteration =',LINITR
      CALL INVIS
      CALL DAMPING
      CALL VISC
      CALL ENSCALE
c     call energy
      NR=0
      CALL SOLVE
c     print*,'Ax-b=',RSQ
C  Check Linearization error
      DO 5 I=1,NCT
       EU=ABS(XI(I,1))
       EV=ABS(XI(I,2))
       EP=ABS(XI(I,3))
       ET=ABS(XI(I,4))
       ERR=AMAX1(ERR,EU,EV,EP,ET)
       RESXI(I,NXM)=ABS(B(I,NXM))
       RESXI(I,NYM)=ABS(B(I,NYM))
       RESXI(I,NEN)=ABS(B(I,NEN))
       RESXI(I,NCO)=ABS(B(I,NCO))
       RES1=AMAX1(RES1,RESXI(I,NXM))
       RES2=AMAX1(RES2,RESXI(I,NYM))
       RES3=AMAX1(RES3,RESXI(I,NEN))
       RES4=AMAX1(RES4,RESXI(I,NCO))
    5 CONTINUE
      PRINT*,'LINEARIZATION ERROR =',ERR
      PRINT*,'X-MOMENTUM RESIDUAL =',RES1
      PRINT*,'Y-MOMENTUM RESIDUAL =',RES2
      PRINT*,'ENERGY RESIDUAL     =',RES3
      PRINT*,'CONTINUITY RESIDUAL =',RES4
      IF(NCOUNT.EQ.2.OR.MOD(NCOUNT,10).EQ.0)THEN
       NRES=NRES+1
       RES5(1,NRES)=NCOUNT
       RES5(2,NRES)=ALOG10(RES4)
      ENDIF
C  Update Solution Vector for Linearization
      DO 3 I=1,NCT
       UP(I)=UP(I)+XI(I,1)
       VP(I)=VP(I)+XI(I,2)
       PP(I)=PP(I)+XI(I,3)
       TP(I)=TP(I)+XI(I,4)
    3 CONTINUE
      IF(ERR.GT.1.D-5.AND.LINITR.LT.NLIN)GO TO 2
C  Update Solution Vector for Next Time Step
      DO 4 I=1,NCT
       U(I)=UP(I)
       V(I)=VP(I)
       P(I)=PP(I)
       T(I)=TP(I)
    4 CONTINUE
    1 CONTINUE
      IF(NRST.LT.2)CALL REWRITE
      CALL OUTPUT
      CALL PLOUT1
      CALL PLOUT2
      CLOSE(1)
      CLOSE(10)
      CLOSE(15)
      CLOSE(20)
      CLOSE(23)
      CLOSE(26)
      CLOSE(30)
      CLOSE(35)
      CLOSE(50)
      CLOSE(60)
      STOP
      END



*DECK INPUT
      SUBROUTINE INPUT
      COMMON/WORD/METHOD
      CHARACTER*3 METHOD
*CALL COMMZ
C
C--INPUT FLOW CODE PARAMETERS AND GRID
C---Code Parameters:
C        NTTS   = Number of Total Time Steps
C        NLIN   = Number of Total Linearization Iterations
C        NSI    = Number of Solver Iterations
C        NRST   = Restart file 0,1,2=0read-1write; 1read-1write;
C                                 1read-0write
C        NIBC   = Inlet Boundary Condition
C        NEBC   = Exit Boundary Condition
C        NSOLVE = Type of solver for Ax=b
C        NDAMP  = Type of artificial damping
C        E1     = Damping coefficient for Laplacian
C        E3     = Damping coefficient for biharmonic
C        NXM    = Variable solved for in the X-Momentum equation 1=U
C        NYM    = Variable solved for in the Y-Momentum equation 2=V
C        NEN    = Variable solved for in the Energy equation 3=P
C        NCO    = Variable solved for in the Continuity equation 4=T
C        METHOD = Character variable that describes the solver
C        NPRET  = Type of preconditioning
C        IGRID  = Dimensional or nondimensional x, y coordinates
C                 1-dim 0-non
C---Time Step Parameters:
C        CFL    = Courant, Friedrichs and Lewy number
C        NDTT   = Minimum or local time step: 0,1
C        PSEUDO = Do not use or use pseudo time step: 0.0, 1.0
C        DTAU   = Nondimensional pseudo time step
C---Fluid Parameters:
C        CPO  = Specific Heat capacity at constant pressure
C        RO   = Gas Constant
C        XMUO = Coefficient of viscosity
C        XLREF= Reference length in computing Reynolds number
```

```
C       PR    = Prandtl Number
C---Through Flow Parameters:
C       P0    = Inlet total pressure
C       T0    = Inlet total temperature
C       PSRAT = exit static pressure ratio (exit static:inlet total)
C       UT    = total velocity
C       UTANG = angle of total velocity
C---Solid Wall Parameters:
C       TW = Temperature of Wall
C
C--NAMELIST INPUT
      NAMELIST /NL1/ NTTS,NLIN,NSI,NRST,NIBC,NEBC,NSOLVE,
     . NDAMP,E1,E3,NXM,NYM,NEN,NCO,METHOD,NPRET,KBV,IGRID
      NAMELIST /NL2/ CFL,NDTT,PSEUDO,DTAU
      NAMELIST /NL3/ CP0,R0,XMU0,XLREF,PR
      NAMELIST /NL4/ P0,T0,PSRAT,UT,UTANG
      NAMELIST /NL5/ TW
      READ(10,NL1)
      READ(10,NL2)
      READ(10,NL3)
      READ(10,NL4)
      READ(10,NL5)
C--ECHO INPUT
      WRITE(15,100)
      WRITE(15,102)
      WRITE(15,105)NTTS
      WRITE(15,107)NLIN
      WRITE(15,108)NSI
      WRITE(15,109)NRST
      WRITE(15,110)CFL
      WRITE(15,115)NIBC
      WRITE(15,120)NEBC
      WRITE(15,121)NSOLVE
      WRITE(15,122)NDAMP
      WRITE(15,131)E1
      WRITE(15,132)E3
      WRITE(15,133)NXM,NYM,NEN,NCO
      WRITE(15,124)
      CV0=CP0-R0
      G=CP0/CV0
      WRITE(15,125)CP0
      WRITE(15,126)CV0
      WRITE(15,128)G
      WRITE(15,130)R0
      WRITE(15,135)XMU0
      WRITE(15,137)XLREF
      WRITE(15,140)PR
      WRITE(15,142)
      WRITE(15,145)P0
      WRITE(15,150)T0
      WRITE(15,155)PSRAT
      WRITE(15,160)UT
      WRITE(15,165)UTANG
      WRITE(15,167)
      WRITE(15,170)TW
      CALL GRIDIN
C--COMPUTE QUANTITIES FOR NONDIMENSIONALIZATION
      RHOO=PO/(RO*TO)
      AO=SQRT(G*RO*TO)
      UO=UT
      RENO=RHOO*UO*XLREF/XMUO
      XMACHO=UT/AO
      WRITE(15,175)
      WRITE(15,185)RHOO
      WRITE(15,190)UO
      WRITE(15,195)RENO
      WRITE(15,200)XMACHO
C     Nondimensionalize geometric quantities
C     at each node
      XD=1.0
      IF(IGRID.EQ.1)XD=XLREF
      WRITE(15,210)
      DO 2 I=1,NNT
      X(I)=X(I)/XD
      Y(I)=Y(I)/XD
      WRITE(15,215)I,X(I),Y(I)
    2 CONTINUE
  100 FORMAT('1',2X,'INPUT TO UNSTRUCTURED FLOW CODE')
  102 FORMAT(//,5X,'CODE PARAMETERS',/)
  105 FORMAT(10X,'Number of Total Time Steps(NTTS)     ',
     .'         =',I11)
  107 FORMAT(10X,'Number of Linearization Iterations(N',
     .'LIN)    =',I11)
  108 FORMAT(10X,'Number of Solver Iterations(NS',
     .'I)      =',I11)
  109 FORMAT(10X,'Read restart file 1=yes 0=no (NRST) ',
     .'         =',I11)
  110 FORMAT(10X,'Courant, Friedrichs, and Lewy number',
     .'(CFL)   =',F11.3)
  115 FORMAT(10X,'Inlet Boundary Condition(NIBC)      ',
     .'         =',I11)
  120 FORMAT(10X,'Exit Boundary Condition(NEBC)       ',
     .'         =',I11)
  121 FORMAT(10X,'Type of solver for Ax=b(NSOLVE)     ',
     .'         =',I11)
  122 FORMAT(10X,'Type of artificial damping(NDAMP)   ',
     .'         =',I11)
  131 FORMAT(10X,'Damping coefficient for Laplacian(E1',
     .')        =',F11.3)
  132 FORMAT(10X,'Damping coefficient for biharmonic(E',
     .'3)       =',F11.3)
  133 FORMAT(10X,'Order of equations solved for U, V, ',
     .'P, T     =',4I5)
  124 FORMAT(//,5X,'FLUID PARAMETERS',/)
  125 FORMAT(10X,'Specific Heat Capacity at constant P',
     .'(CPO)  =',F11.3)
  126 FORMAT(10X,'Specific Heat Capacity at constant V',
     .'(CVO)  =',F11.3)
  128 FORMAT(10X,'Ratio of Specific Heats(G)           ',
     .'       =',F11.3)
  130 FORMAT(10X,'Gas Constant(RO)                     ',
     .'       =',F11.3)
  135 FORMAT(10X,'Coefficient of Viscosity(XMUO)       ',
     .'       =',1PE11.4)
  137 FORMAT(10X,'Reference length for Reynolds number',
     .'(XLREF)=',F11.3)
  140 FORMAT(10X,'Prandtl Number(PR)                   ',
     .'       =',F11.3)
  142 FORMAT(//,5X,'THROUGH FLOW PARAMETERS',/)
  145 FORMAT(10X,'Inlet Total Pressure(PO)             ',
     .'       =',1PE11.4)
  150 FORMAT(10X,'Inlet Total Temperature(TO)          ',
     .'       =',F11.3)
  155 FORMAT(10X,'Exit Static Pressure Ratio(PSRAT)    ',
     .'       =',F11.3)
  160 FORMAT(10X,'Inlet Total Velocity(UT)             ',
     .'       =',F11.3)
  165 FORMAT(10X,'Inlet Total Velocity Angle(UTANG)    ',
     .'       =',F11.3)
  167 FORMAT(//,5X,'SOLID WALL PARAMETERS',/)
  170 FORMAT(10X,'Wall Temperature(TW)                 ',
     .'       =',F11.3)
  175 FORMAT(//,5X,'NONDIMENSIONAL QUANTITIES',/)
  180 FORMAT(10X,'Reference Length(L)                  ',
     .'       =',F11.3)
  185 FORMAT(10X,'Reference Density(RHOO)              ',
     .'       =',F11.3)
  190 FORMAT(10X,'Reference Velocity(UO)               ',
     .'       =',F11.3)
  195 FORMAT(10X,'Reynolds Number(RENO)                ',
     .'       =',1PE11.4)
  200 FORMAT(10X,'Mach Number(XMACHO)                  ',
     .'       =',F11.5)
  210 FORMAT(//,5X,'NODE',11X,'X',15X,'Y',/)
  215 FORMAT(2X,I6,5X,F11.5,5X,F11.5)
      RETURN
      END


*DECK GRIDIN
      SUBROUTINE GRIDIN
*CALL COMMZ
C--Read in grid data
C            Cell number - nc
C-----READ IN FACE DATA
```

```
      READ(20,*)
      READ(20,*)
      READ(20,*)NFT
      READ(20,*)
      DO 1 I=1,NFT
      READ(20,*)NDUM,NFACE(1,I),NFACE(2,I)
    1 CONTINUE
C-----READ IN CELL DATA
      READ(20,*)
      READ(20,*)
      READ(20,*)NCT
      READ(20,*)
      DO 2 I=1,NCT
      READ(20,*)NDUM,(NCELL(K,I),K=1,6)
    2 CONTINUE
C-----READ IN NODE DATA
      READ(23,*)
      READ(23,*)
      READ(23,*)NNT
      READ(23,*)
      DO 3 I=1,NNT
      READ(23,*)NDUM,X(I),Y(I)
    3 CONTINUE
C-----INLET CELLS(Must be determined to specify inlet boundary
C     conditions)
      NILT=0
      DO 4 I=1,NFT
      IF(NFACE(1,I).EQ.-1)THEN
       NILT=NILT+1
       NCELLIL(NILT)=NFACE(2,I)
      ELSE
      IF(NFACE(2,I).EQ.-1)THEN
       NILT=NILT+1
       NCELLIL(NILT)=NFACE(1,I)
      ENDIF
      ENDIF
    4 CONTINUE
C----Input 4 color cell information
c      DO 5 K=1,4
c       READ(26,*)NRGBY(K)
c       NCLOOP=NRGBY(K)
c       DO 5 N=1,NCLOOP
c        READ(26,*)NCOLOR(K,N)
c    5 CONTINUE

      RETURN
      END




*DECK INITV
      SUBROUTINE INITV
*CALL COMMZ
C      Input ordering array for ordering extra viscous triangles
      NPERM(1,1)=1
      NPERM(1,2)=2
      NPERM(1,3)=3
      NPERM(2,1)=2
      NPERM(2,2)=3
      NPERM(2,3)=1
      NPERM(3,1)=3
      NPERM(3,2)=1
      NPERM(3,3)=2
      RETURN
      END



*DECK CONNECT
      SUBROUTINE CONNECT
*CALL COMMZ
      DO 1 I=1,NCT
      ICA=I
C---FACES OF CELL I
      NF1=NCELL(1,I)
      NF2=NCELL(2,I)
      NF3=NCELL(3,I)
C------NUMBER OF CELL ADJACENT TO CELL I ACROSS FACE A-B
      ICB=NFACE(1,NF1) + NFACE(2,NF1) - I
C------NUMBER OF CELL ADJACENT TO CELL I ACROSS FACE B-C
      ICC=NFACE(1,NF2) + NFACE(2,NF2) - I
C------NUMBER OF CELL ADJACENT TO CELL I ACROSS FACE C-A
      ICD=NFACE(1,NF3) + NFACE(2,NF3) - I
C-----Find Cell #'s of E, F, G, H, I, J
C------Faces of Cell B
      NFS1=NCELL(1,ICB)
      NFS2=NCELL(2,ICB)
      NFS3=NCELL(3,ICB)
C------Cells surrounding cell B (E, F)
C------Determine cell numbers in the order A-E-F
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICB
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICB
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICB
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICE=ICS(LAD)
      ICF=ICS(LDB)
C------Faces of Cell C
      NFS1=NCELL(1,ICC)
      NFS2=NCELL(2,ICC)
      NFS3=NCELL(3,ICC)
C------Cells surrounding cell C (G, H)
C------Determine cell numbers in the order A-G-H
```

```
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICC
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICC
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICC
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICG=ICS(LAD)
      ICH=ICS(LDB)

C------Faces of Cell D
      NFS1=NCELL(1,ICD)
      NFS2=NCELL(2,ICD)
      NFS3=NCELL(3,ICD)
C------Cells surrounding cell D (I, J)
C------Determine cell numbers in the order A-I-J
      ICS(1)=NFACE(1,NFS1)+NFACE(2,NFS1)-ICD
      ICS(2)=NFACE(1,NFS2)+NFACE(2,NFS2)-ICD
      ICS(3)=NFACE(1,NFS3)+NFACE(2,NFS3)-ICD
       NFA=INT(1./COSH(FLOAT(ICS(1)-ICA)))
       NFB=INT(1./COSH(FLOAT(ICS(2)-ICA)))
       NFC=INT(1./COSH(FLOAT(ICS(3)-ICA)))
       LAB=NFA*1 + NFB*2 + NFC*3
       LAD=NPERM(LAB,2)
       LDB=NPERM(LAB,3)
      ICI=ICS(LAD)
      ICJ=ICS(LDB)

C------Save column numbers for SOLVER
      NUMEL(I,1)=ICA
      NUMEL(I,2)=ICB
      NUMEL(I,3)=ICC
      NUMEL(I,4)=ICD

      IF(ICB.GT.0)THEN
       NUMEL(I,5)=ICE
       NUMEL(I,6)=ICF
      ELSE
       ICE=ICB
       ICF=ICB
      ENDIF

      NUMEL(I,7)=ICG
      NUMEL(I,8)=ICH
      NUMEL(I,9)=ICI
      NUMEL(I,10)=ICJ

    1 CONTINUE

      RETURN
      END


*DECK INITFD
```

```
      SUBROUTINE INITFD
*CALL COMMZ
C     Initial Conditions computed for fully developed flow
C     U, V, P, T
C     Inlet conditions
      PIE  =3.141592654
      TORAD=1.745329252E-2
      UTANG=UTANG*TORAD
      GM=G-1.0
      TS=TO-.5*UT**2/CPO
      CS=SQRT(G*RO*TS)
      XMACHT=UT/CS
      PS=PO*(1.0+GM/2*XMACHT**2)**(-G/GM)
C   Channel flow or uniform periodic flow
c       h=1.0
c       h2=h/2.
c       DO 1 n=1,NCT
c       n1=ncell(4,n)
c       n2=ncell(5,n)
c       n3=ncell(6,n)
c       ycav=(y(n1)+y(n2)+y(n3))/3.
C   Fully developed flow
c       u(n)=ut*(1.-(ycav-h2)**2/h2**2)
c       u(n)=ut*(1.-(ycav)**2/h2**2)
C   Uniform flow
c       u(n)=ut*cos(utang)
c       V(N)=ut*sin(utang)
c       P(N)=PS
c       T(N)=TS
c     1 CONTINUE
C   Sudden expansion -- fully developed
      ycl=1.5
      DO 2 N=1,NCT
      n1=ncell(4,n)
      n2=ncell(5,n)
      n3=ncell(6,n)
      xcav=(x(n1)+x(n2)+x(n3))/3.
      ycav=(y(n1)+y(n2)+y(n3))/3.
      if(ycav.le.ycl)then
       if(xcav.le.1.0)then
c         U(N)=UT*SIN(pie*(ycav-1.0))
        U(N)=UT*(1.-(ycav-1.5)**2/.25)
       else
c         U(N)=UT/9.*SIN(pie*ycav/3.)
        U(N)=UT/9.*(1.-(ycav-1.5)**2/2.25)
       endif
      else
       if(xcav.le.1.0)then
c         U(N)=UT*SIN(pie*sqrt((ycav-2.)**2))
        U(N)=UT*(1.-(ycav-1.5)**2/.25)
       else
c         U(N)=UT/9.*SIN(pie*sqrt((ycav-3.)**2)/3.)
```

```
          U(N)=UT/9.*(1.-(ycav-1.5)**2/2.25)
         endif
        endif
        V(N)=0.0
        P(N)=PS
        T(N)=TS
      2 CONTINUE
C     Backward facing step -- fully developed
c        h1=.5
c        h2=1.9423/2.
c        h12=.5
c        h22=.02885
c        DO 2 N=1,NCT
c        n1=ncell(4,n)
c        n2=ncell(5,n)
c        n3=ncell(6,n)
c        xcav=(x(n1)+x(n2)+x(n3))/3.
c        ycav=(y(n1)+y(n2)+y(n3))/3.
c        if(xcav.le.1.0)then
c         U(N)=UT*(1.-(ycav-h12)**2/h1**2)
c        else
c         U(N)=UT/1.9423*(1.-(ycav-h22)**2/h2**2)
c        endif
c        V(N)=0.0
c        P(N)=PS
c        T(N)=TS
c      2 CONTINUE
C     Celtic valve fully developed
c        ycl=0.0
c        DO 3 N=1,NCT
c        n1=ncell(4,n)
c        n2=ncell(5,n)
c        n3=ncell(6,n)
c        xcav=(x(n1)+x(n2)+x(n3))/3.
c        ycav=(y(n1)+y(n2)+y(n3))/3.
c        if(xcav.le.-1.414)then
c          u(n)=ut*(1.-(ycav)**2/0.5**2)
c        else
c         U(N)=UT
c        endif
c        V(N)=0.0
c        P(N)=PS
c        T(N)=TS
c      3 CONTINUE
C     Corner3 flow fully developed
c        xcl1=1.5
c        ycl1=5.5
c        xcl2=3.5
c        ycl2=3.5
c        DO 4 N=1,NCT
c        n1=ncell(4,n)
c        n2=ncell(5,n)
c        n3=ncell(6,n)
c        xcav=(x(n1)+x(n2)+x(n3))/3.
```

```
c       ycav=(y(n1)+y(n2)+y(n3))/3.
c       if(ycav.gt.xcav+4.)then
c        if(ycav.le.ycl1)then
c         U(N)=UT*SIN(pie*(ycav-ycl1+0.5))
c         V(N)=0.0
c        else
c         U(N)=UT*SIN(pie*sqrt((ycav-ycl1-0.5)**2))
c         V(N)=0.0
c        endif
c       else
c       if(ycav.gt.xcav+2.)then
c        if(xcav.le.xcl1)then
c         U(N)=0.0
c         V(N)=-UT*SIN(pie*(xcav-xcl1+0.5))
c        else
c         U(N)=0.0
c         V(N)=-UT*SIN(pie*sqrt((xcav-xcl1-0.5)**2))
c        endif
c       else
c       if(ycav.gt.xcav)then
c        if(ycav.le.ycl1)then
c         U(N)=UT*SIN(pie*(ycav-ycl2+0.5))
c         V(N)=0.0
c        else
c         U(N)=UT*SIN(pie*sqrt((ycav-ycl2-0.5)**2))
c         V(N)=0.0
c        endif
c       else
c        if(xcav.le.xcl1)then
c         U(N)=0.0
c         V(N)=-UT*SIN(pie*(xcav-xcl2+0.5))
c        else
c         U(N)=0.0
c         V(N)=-UT*SIN(pie*sqrt((xcav-xcl2-0.5)**2))
c        endif
c       endif
c       endif
c       endif
c       P(N)=PS
c       T(N)=TS
c     4 CONTINUE
C    Compute inlet conditions along boundary A-B for subsonic or
C    supersonic boundary conditions
      DO 10 I=1,NILT
      N=NCELLIL(I)
      UI(N)=U(N)
      VI(N)=V(N)
      PI(N)=PS
      TI(N)=TS
   10 CONTINUE

c     PEXIT=PSRAT*PO
C      Freestream conditions(subsonic flow)
      PEXIT=PS
```

```
C   Sutherland's Law Constants(dimensional)
C    Air at moderate temperatures(Metric units)
      SC1=1.458E-6
      SC2=110.4
C   Initialize A matrix and b vector
      DO 25 J=0,10
      DO 25 I=0,NCT
      A(I,J,1,1)=0.0
      A(I,J,1,2)=0.0
      A(I,J,1,3)=0.0
      A(I,J,1,4)=0.0
      A(I,J,2,1)=0.0
      A(I,J,2,2)=0.0
      A(I,J,2,3)=0.0
      A(I,J,2,4)=0.0
      A(I,J,3,1)=0.0
      A(I,J,3,2)=0.0
      A(I,J,3,3)=0.0
      A(I,J,3,4)=0.0
      A(I,J,4,1)=0.0
      A(I,J,4,2)=0.0
      A(I,J,4,3)=0.0
      A(I,J,4,4)=0.0
   25 CONTINUE
      DO 30 I=0,NCT
      B(I,1)=0.0
      B(I,2)=0.0
      B(I,3)=0.0
      B(I,4)=0.0
   30 CONTINUE
      RETURN
      END



*DECK NONDIM
      SUBROUTINE NONDIM
*CALL COMMZ
C     Nondimensionalize flow quantities
C     in each cell
      WRITE(15,100)
      CP=CP0*T0/U0**2
      R=R0*T0/U0**2
      DO 1 I=1,NCT
      U(I)=U(I)/U0
      V(I)=V(I)/U0
      P(I)=P(I)/RHO0/U0**2
      T(I)=T(I)/T0
      WRITE(15,105)I,U(I),V(I),P(I),T(I)
    1 CONTINUE
C     Nondimensionalize inlet conditions for subsonic and
```

```
C     supersonic flow BC's at side A-B
      DO 2 I=1,NILT
      N=NCELLIL(I)
      UI(N)=UI(N)/U0
      VI(N)=VI(N)/U0
      PI(N)=PI(N)/RHO0/U0**2
      TI(N)=TI(N)/T0
    2 CONTINUE
      P0I=P0/RHO0/U0**2
C     Nondimensionalize wall temperature for solid wall BC

      TW=TW/T0

C     Nondimensionalize exit pressure for subsonic flow BC

      PEXIT=PEXIT/RHO0/U0**2

C   Initialize nondimensional provisional quantities
      DO 3 I=1,NCT
      UP(I)=U(I)
      VP(I)=V(I)
      PP(I)=P(I)
      TP(I)=T(I)
    3 CONTINUE
C   Sutherland's Law Constants nondimensionalized

      SC1=SC1*SQRT(T0)/XMU0
      SC2=SC2/T0

  100 FORMAT(///,1X,'NONDIMENSIONALIZED QUANTITIES',//,
     . 5X,'CELL',11X,'U',15X,'V',15X,'P',15X,'T',/)
  105 FORMAT(2X,I6,5X,F11.5,5X,F11.5,5X,F11.5,5X,F11.5)
      RETURN
      END



*DECK VOLUME
      SUBROUTINE VOLUME
*CALL COMMZ

C  Compute Volume(VOL(I)) of cell

      WRITE(15,100)
      DO 1 I=1,NCT

C---NODES OF CELL I
      N1=NCELL(4,I)
      N2=NCELL(5,I)
      N3=NCELL(6,I)
C------NODES OF FACE A-B ARE N1 AND N2
      DXAB=X(N2)-X(N1)
      DYAB=Y(N2)-Y(N1)
C------NODES OF FACE B-C ARE N2 AND N3
      DXBC=X(N3)-X(N2)
      DYBC=Y(N3)-Y(N2)
C------NODES OF FACE C-A ARE N3 AND N1
```

```
      DXCA=X(N1)-X(N3)
      DYCA=Y(N1)-Y(N3)
C-------CALCULATE AREA OF CELL I
      AB=SQRT(DXAB**2+DYAB**2)
      BC=SQRT(DXBC**2+DYBC**2)
      CA=SQRT(DXCA**2+DYCA**2)
      SABC=.5*(AB+BC+CA)
      VOL(I)=SQRT(SABC*(SABC-AB)*(SABC-BC)*(SABC-CA))
      WRITE(15,110)I,VOL(I)
    1 CONTINUE
  100 FORMAT(//,5X,'VOLUME OF CELLS',//,4X,'CELL',
     .11X,'VOLUME',/)
  110 FORMAT(I6,10X,1PE11.5)
      RETURN
      END



*DECK TIMST
      SUBROUTINE TIMST
*CALL COMMZ
C     Time step based on edge velocities (average of adjacent cell
C     quantities) and edge geometry

      DO 1 I=1,NCT

C----Determine adjacent cell
      ICA=I
      ICB=NUMEL(I,2)
      IF(ICB.LE.0)THEN
       ICB=ICA
      ENDIF
      ICC=NUMEL(I,3)
      ICD=NUMEL(I,4)
C---NODES OF CELL I
      N1=NCELL(4,I)
      N2=NCELL(5,I)
      N3=NCELL(6,I)

C     Compute average U, V, P, T.
      UAB=.5*(U(ICA)+U(ICB))
      VAB=.5*(V(ICA)+V(ICB))
      PAB=.5*(P(ICA)+P(ICB))
      TAB=.5*(T(ICA)+T(ICB))
      UBC=.5*(U(ICA)+U(ICC))
      VBC=.5*(V(ICA)+V(ICC))
      PBC=.5*(P(ICA)+P(ICC))
      TBC=.5*(T(ICA)+T(ICC))

      UCA=.5*(U(ICA)+U(ICD))
      VCA=.5*(V(ICA)+V(ICD))
      PCA=.5*(P(ICA)+P(ICD))
      TCA=.5*(T(ICA)+T(ICD))

C     Compute DELTA(x), DELTA(y) on each edge
C------NODES OF FACE A-B ARE N1 AND N2
      DXAB=X(N2)-X(N1)
      DYAB=Y(N2)-Y(N1)
C------NODES OF FACE B-C ARE N2 AND N3
      DXBC=X(N3)-X(N2)
      DYBC=Y(N3)-Y(N2)
C------NODES OF FACE C-A ARE N3 AND N1
      DXCA=X(N1)-X(N3)
      DYCA=Y(N1)-Y(N3)

C     Compute (speed of sound)**2 for each edge
      CCAB=G*R*TAB
      CCBC=G*R*TBC
      CCCA=G*R*TCA
C     Compute velocity and speed of sound across each edge AB,
C     BC, CA; i.e. the contravariant component of velocity
C     and the speed of sound.
      QSAB=DYAB*UAB-DXAB*VAB
      QSBC=DYBC*UBC-DXBC*VBC
      QSCA=DYCA*UCA-DXCA*VCA

      CSAB=CCAB*(DXAB**2+DYAB**2)
      CSBC=CCBC*(DXBC**2+DYBC**2)
      CSCA=CCCA*(DXCA**2+DYCA**2)

      AAB=ABS(QSAB)+SQRT(CSAB)
      ABC=ABS(QSBC)+SQRT(CSBC)
      ACA=ABS(QSCA)+SQRT(CSCA)
C     Sum reciprocal of time step of each face and temporarily
C     stored in dt(i)
      DT(I)=AAB+ABC+ACA
C     Actual time step  CFL condition is multiplied when integrating
      DT(I)=VOL(I)/DT(I)
    1 CONTINUE

C     Compute minimum time step
      DTMIN=1.E6
      IF(NDTT.EQ.0)THEN
      DO 2 I=1,NCT
      DTMIN=AMIN1(DTMIN,DT(I))
    2 CONTINUE
```

```
      ENDIF

      WRITE(15,100)
      DO 3 I=1,NCT
      WRITE(15,105)I,DT(I)
    3 CONTINUE

      WRITE(15,110)DTMIN

  100 FORMAT(//,5X,'TIME STEP',/,5X,'CELL',13X,'DT',/)
  105 FORMAT(5X,I6,10X,1PE11.5)
  110 FORMAT(///,5X,'MINIMUM TIME STEP = ',1PE11.5)

      RETURN
      END



*DECK INVIS
      SUBROUTINE INVIS
*CALL COMMZ

      DO 1 I=1,NCT

      DELT=CFL*AMIN1(DT(I),DTMIN)
      DELT=1.E6
      DTAU=CFL*AMIN1(DT(I),DTMIN)

      ICA=I
      ICB=NUMEL(I,2)
      ICC=NUMEL(I,3)
      ICD=NUMEL(I,4)

C---NODES OF CELL I
      N1=NCELL(4,ICA)
      N2=NCELL(5,ICA)
      N3=NCELL(6,ICA)
C------NODES OF FACE A-B ARE N1 AND N2
      DXAB=X(N2)-X(N1)
      DYAB=Y(N2)-Y(N1)
C------NODES OF FACE B-C ARE N2 AND N3
      DXBC=X(N3)-X(N2)
      DYBC=Y(N3)-Y(N2)
C------NODES OF FACE C-A ARE N3 AND N1
      DXCA=X(N1)-X(N3)
      DYCA=Y(N1)-Y(N3)
C-------AREA OF CELL I
      S=VOL(ICA)
C-------VARIABLES OF CELL A, B, C, AND D
C    CELL A
      UA=U(ICA)
      VA=V(ICA)
      PA=P(ICA)
      TA=T(ICA)
C    PROVISIONAL VALUES
      UPA=UP(ICA)
      VPA=VP(ICA)
      PPA=PP(ICA)
```

```
        TPA=TP(ICA)
         xyb=1.
         xywa=0.0
         xyia=0.0
         xyea=0.0
C       CELL B
        if(icb.gt.0)then
        UPB=UP(ICB)
        VPB=VP(ICB)
        PPB=PP(ICB)
        TPB=TP(ICB)
        else
        if(icb.eq.0)then
        xyb=0.0
        xywa=1.0
C     VISCOUS WALL BOUNDARY CONDITION ON FACE AB
        UPB=-UPA
        VPB=-VPA
        PPB=PPA
        TPB=TPA
        else
C     INLET
        if(icb.eq.-1)then
        xyb=0.0
        xyia=1.0
        if(nibc.eq.0)then
        UPB=2.*UI(ICA)-UPA
        VPB=2.*VI(ICA)-VPA
        PPB=PPA
        TPB=2.*TI(ICA)-TPA
        DPC=1.
        else
        if(nibc.eq.1)then
C     Riemann invariant formulation
        GM=G-1.
        GP=G+1.
        VAB=VI(ICA)
        CSA=SQRT(G*R*TA)
        RMIN=UA-2.*CSA/GM
        UAB=(GM*RMIN+SQRT(4.*GP*CP-2.*(GP*VAB**2+GM*RMIN**2)))/GP
        UVSQ=UAB**2+VAB**2
        TAB=1.-UVSQ/(2.*CP)
        CSABSQ=G*R*TAB
        XMABSQ=UVSQ/CSABSQ
        GGM=G/GM
        PAB=POI*(1.+.5*GM*XMABSQ)**(-GGM)
        UPB=2.*UAB-UPA
        VPB=2.*VAB-VPA
        PPB=2.*PAB-PPA
        TPB=2.*TAB-TPA
        DPC=1.
        else
        if(nibc.eq.2)then
C     SUPERSONIC INLET BOUNDARY CONDITION ON FACE AB
C     Set quantities in cell B such that side A-B of cell A
C     holds the set boundary conditions.
        UPB=2.*UI(ICA)-UPA
```

```
      VPB=2.*VI(ICA)-VPA
      PPB=2.*PI(ICA)-PPA
      TPB=2.*TI(ICA)-TPA
      DPC=-1.
      endif
      endif
      endif
      else
C   EXIT
      if(icb.eq.-2)then
      xyb=0.0
      xyea=1.0
      if(nebc.eq.1)then
C   SUBSONIC EXIT BOUNDARY CONDITION ON FACE AB
      UPB=UP(ICA)
      VPB=VP(ICA)
      PPB=2.*PEXIT-PPA
      TPB=TP(ICA)
      DPC=1.
      else
      if(nebc.eq.2)then
C   SUPERSONIC EXIT BOUNDARY CONDITION ON FACE AB
      UPB=UP(ICA)
      VPB=VP(ICA)
      PPB=PP(ICA)
      TPB=TP(ICA)
      DPC=-1.
      endif
      endif
      endif
      endif
      endif
      endif
C    CELL C
      UPC=UP(ICC)
      VPC=VP(ICC)
      PPC=PP(ICC)
      TPC=TP(ICC)
C    CELL D
      UPD=UP(ICD)
      VPD=VP(ICD)
      PPD=PP(ICD)
      TPD=TP(ICD)
C--CONTINUITY EQUATION

C   DELTA(U)
C           A

      A(I,1,NCO,1)=0.0
     . -xywa*.5*PPB/TPB*DYAB
     . -xyia*.5*PPB/TPB*DYAB
     . +xyea*.5*PPB/TPB*DYAB

C   DELTA(V)
C           A

      A(I,1,NCO,2)=0.0
     . +xywa*.5*PPB/TPB*DXAB
     . +xyia*.5*PPB/TPB*DXAB
     . -xyea*.5*PPB/TPB*DXAB
```

```
C   DELTA(P)
C          A
     A(I,1,NCO,3)=S/TPA/DELT
    . +xywa*.5*    (UPB*DYAB-VPB*DXAB)/TPB
    . +xyia*.5*DPC*(UPB*DYAB-VPB*DXAB)/TPB
    . -xyea*.5*DPC*(UPB*DYAB-VPB*DXAB)/TPB
C    Pseudo time term
     A(I,1,NCO,3)=A(I,1,NCO,3)
    .            +PSEUDO/DTAU*R*S/TPA
C   DELTA(T)
C          A
     A(I,1,NCO,4)=-S*PPA/TPA**2/DELT
    . -xywa*.5*PPB/TPB**2*(UPB*DYAB-VPB*DXAB)
    . +xyia*.5*PPB/TPB**2*(UPB*DYAB-VPB*DXAB)
    . -xyea*.5*PPB/TPB**2*(UPB*DYAB-VPB*DXAB)
C    Pseudo time term
     A(I,1,NCO,4)=A(I,1,NCO,4)
    .            -PSEUDO/DTAU*S*PPA/TPA**2
C   DELTA(U)
C          B
     A(I,2,NCO,1)=xyb*.5*PPB/TPB*DYAB
C   DELTA(V)
C          B
     A(I,2,NCO,2)=-xyb*.5*PPB/TPB*DXAB
C   DELTA(P)
C          B
     A(I,2,NCO,3)=xyb*.5*(UPB*DYAB-VPB*DXAB)/TPB
C   DELTA(T)
C          B
     A(I,2,NCO,4)=-xyb*.5*PPB/TPB**2*(UPB*DYAB-VPB*DXAB)
C   DELTA(U)
C          C
     A(I,3,NCO,1)=.5*PPC/TPC*DYBC
C   DELTA(V)
C          C
     A(I,3,NCO,2)=-.5*PPC/TPC*DXBC
C   DELTA(P)
C          C
     A(I,3,NCO,3)=.5*(UPC*DYBC-VPC*DXBC)/TPC
C   DELTA(T)
```

```
C           C
      A(I,3,NCO,4)=-.5*PPC/TPC**2*(UPC*DYBC-VPC*DXBC)
     .
C    DELTA(U)
C           D
      A(I,4,NCO,1)=.5*PPD/TPD*DYCA
C    DELTA(V)
C           D
      A(I,4,NCO,2)=-.5*PPD/TPD*DXCA
C    DELTA(P)
C           D
      A(I,4,NCO,3)=.5*(UPD*DYCA-VPD*DXCA)/TPD
C    DELTA(T)
C           D
      A(I,4,NCO,4)=-.5*PPD/TPD**2*(UPD*DYCA-VPD*DXCA)
C    RHS
C
      B(I,NCO)=-(S*(PPA/TPA-PA/TA)/DELT
     . +.5*(PPB*UPB/TPB*DYAB+PPC*UPC/TPC*DYBC+PPD*UPD/TPD*DYCA
     .      -PPB*VPB/TPB*DXAB-PPC*VPC/TPC*DXBC-PPD*VPD/TPD*DXCA))
C--X MOMENTUM EQUATION
C
C     DELTA(U)
C            A
      A(I,1,NXM,1)=S*PPA/TPA/DELT
     . -xywa*.5*PPB/TPB*(2.*UPB*DYAB-VPB*DXAB)
     . -xyia*.5*PPB/TPB*(2.*UPB*DYAB-VPB*DXAB)
     . +xyea*.5*PPB/TPB*(2.*UPB*DYAB-VPB*DXAB)
C     Pseudo time term
      A(I,1,NXM,1)=A(I,1,NXM,1)
     .            +PSEUDO/DTAU*S*PPA/TPA
C     DELTA(V)
C            A
      A(I,1,NXM,2)=0.0
     . +xywa*.5*PPB*UPB/TPB*DXAB
     . +xyia*.5*PPB*UPB/TPB*DXAB
     . -xyea*.5*PPB*UPB/TPB*DXAB
C     DELTA(P)
C            A
      A(I,1,NXM,3)=S*UPA/TPA/DELT
     . +xywa*.5*    (UPB/TPB*(UPB*DYAB-VPB*DXAB)+R*DYAB)
     . +xyia*.5*DPC*(UPB/TPB*(UPB*DYAB-VPB*DXAB)+R*DYAB)
     . -xyea*.5*DPC*(UPB/TPB*(UPB*DYAB-VPB*DXAB)+R*DYAB)
```

```
C     Pseudo time term
       A(I,1,NXM,3)=A(I,1,NXM,3)
      .             +PSEUDO/DTAU*R*S*UPA/TPA
C     DELTA(T)
C             A
       A(I,1,NXM,4)=-S*PPA*UPA/TPA**2/DELT
      . -xywa*.5*PPB*UPB/TPB**2*(UPB*DYAB-VPB*DXAB)
      . +xyia*.5*PPB*UPB/TPB**2*(UPB*DYAB-VPB*DXAB)
      . -xyea*.5*PPB*UPB/TPB**2*(UPB*DYAB-VPB*DXAB)
C     Pseudo time term
       A(I,1,NXM,4)=A(I,1,NXM,4)
      .             -PSEUDO/DTAU*S*PPA*UPA/TPA**2
C     DELTA(U)
C             B
       A(I,2,NXM,1)=xyb*.5*PPB/TPB*(2.*UPB*DYAB-VPB*DXAB)
C     DELTA(V)
C             B
       A(I,2,NXM,2)=-xyb*.5*PPB*UPB/TPB*DXAB
C     DELTA(P)
C             B
       A(I,2,NXM,3)=xyb*.5*(UPB/TPB*(UPB*DYAB-VPB*DXAB)+R*DYAB)
C     DELTA(T)
C             B
       A(I,2,NXM,4)=-xyb*.5*PPB*UPB/TPB**2*(UPB*DYAB-VPB*DXAB)
      .
C     DELTA(U)
C             C
       A(I,3,NXM,1)=.5*PPC/TPC*(2.*UPC*DYBC-VPC*DXBC)
      .
C     DELTA(V)
C             C
       A(I,3,NXM,2)=-.5*PPC*UPC/TPC*DXBC
C     DELTA(P)
C             C
       A(I,3,NXM,3)=.5*(UPC/TPC*(UPC*DYBC-VPC*DXBC)+R*DYBC)
C     DELTA(T)
C             C
       A(I,3,NXM,4)=-.5*PPC*UPC/TPC**2*(UPC*DYBC-VPC*DXBC)
      .
C     DELTA(U)
C             D
```

```
       A(I,4,NXM,1)=.5*PPD/TPD*(2.*UPD*DYCA-VPD*DXCA)
     .
C     DELTA(V)
C             D
       A(I,4,NXM,2)=-.5*PPD*UPD/TPD*DXCA
C     DELTA(P)
C             D
       A(I,4,NXM,3)=.5*(UPD/TPD*(UPD*DYCA-VPD*DXCA)+R*DYCA)
C     DELTA(T)
C             D
       A(I,4,NXM,4)=-.5*PPD*UPD/TPD**2*(UPD*DYCA-VPD*DXCA)
     .
C     RHS
C
       B(I,NXM)=-(S*(PPA*UPA/TPA-PA*UA/TA)/DELT
     .   +.5*(PPB*UPB**2/TPB*DYAB
     .        +PPC*UPC**2/TPC*DYBC
     .        +PPD*UPD**2/TPD*DYCA
     .        +R*(PPB*DYAB+PPC*DYBC+PPD*DYCA)
     .        -PPB*UPB*VPB/TPB*DXAB
     .        -PPC*UPC*VPC/TPC*DXBC
     .        -PPD*UPD*VPD/TPD*DXCA))
C--Y MOMENTUM EQUATION
C
C     DELTA(U)
C             A
       A(I,1,NYM,1)=0.0
     . -xywa*.5*PPB*VPB/TPB*DYAB
     . -xyia*.5*PPB*VPB/TPB*DYAB
     . +xyea*.5*PPB*VPB/TPB*DYAB
C     DELTA(V)
C             A
       A(I,1,NYM,2)=S*PPA/TPA/DELT
     . -xywa*.5*PPB/TPB*(UPB*DYAB-2.*VPB*DXAB)
     . -xyia*.5*PPB/TPB*(UPB*DYAB-2.*VPB*DXAB)
     . +xyea*.5*PPB/TPB*(UPB*DYAB-2.*VPB*DXAB)
     .
C     Pseudo time term
       A(I,1,NYM,2)=A(I,1,NYM,2)
     .             +PSEUDO/DTAU*S*PPA/TPA
C     DELTA(P)
C             A
       A(I,1,NYM,3)=S*VPA/TPA/DELT
     . +xywa*.5*    (VPB/TPB*(UPB*DYAB-VPB*DXAB)-R*DXAB)
     . +xyia*.5*DPC*(VPB/TPB*(UPB*DYAB-VPB*DXAB)-R*DXAB)
     . -xyea*.5*DPC*(VPB/TPB*(UPB*DYAB-VPB*DXAB)-R*DXAB)
C    Pseudo time term
      A(I,1,NYM,3)=A(I,1,NYM,3)
     .             +PSEUDO/DTAU*R*S*VPA/TPA
C    DELTA(T)
C           A
      A(I,1,NYM,4)=-S*PPA*VPA/TPA**2/DELT
     . -xywa*.5*PPB*VPB/TPB**2*(UPB*DYAB-VPB*DXAB)
     . +xyia*.5*PPB*VPB/TPB**2*(UPB*DYAB-VPB*DXAB)
     . -xyea*.5*PPB*VPB/TPB**2*(UPB*DYAB-VPB*DXAB)
C    Pseudo time term
      A(I,1,NYM,4)=A(I,1,NYM,4)
     .             -PSEUDO/DTAU*S*PPA*VPA/TPA**2
C    DELTA(U)
C           B
      A(I,2,NYM,1)=xyb*.5*PPB*VPB/TPB*DYAB
C    DELTA(V)
C           B
      A(I,2,NYM,2)=xyb*.5*PPB/TPB*(UPB*DYAB-2.*VPB*DXAB)
     .
C    DELTA(P)
C           B
      A(I,2,NYM,3)=xyb*.5*(VPB/TPB*(UPB*DYAB-VPB*DXAB)-R*DXAB)
C    DELTA(T)
C           B
      A(I,2,NYM,4)=-xyb*.5*PPB*VPB/TPB**2*(UPB*DYAB-VPB*DXAB)

C    DELTA(U)
C           C
      A(I,3,NYM,1)=.5*PPC*VPC/TPC*DYBC
C    DELTA(V)
C           C
      A(I,3,NYM,2)=.5*PPC/TPC*(UPC*DYBC-2.*VPC*DXBC)
C    DELTA(P)
C           C
      A(I,3,NYM,3)=.5*(VPC/TPC*(UPC*DYBC-VPC*DXBC)-R*DXBC)
     .
     .
C    DELTA(T)
C           C
      A(I,3,NYM,4)=-.5*PPC*VPC/TPC**2*(UPC*DYBC-VPC*DXBC)
```

```
C     DELTA(U)
C             D
       A(I,4,NYM,1)=.5*PPD*VPD/TPD*DYCA
C     DELTA(V)
C             D
       A(I,4,NYM,2)=.5*PPD/TPD*(UPD*DYCA-2.*VPD*DXCA)
C     DELTA(P)
C             D
       A(I,4,NYM,3)=.5*(VPD/TPD*(UPD*DYCA-VPD*DXCA)-R*DXCA)
C     DELTA(T)
C             D
       A(I,4,NYM,4)=-.5*PPD*VPD/TPD**2*(UPD*DYCA-VPD*DXCA)
C     RHS
C
       B(I,NYM)=-(S*(PPA*VPA/TPA-PA*VA/TA)/DELT
      . +.5*(PPB*UPB*VPB/TPB*DYAB
      .     +PPC*UPC*VPC/TPC*DYBC
      .     +PPD*UPD*VPD/TPD*DYCA
      .     -PPB*VPB**2/TPB*DXAB
      .     -PPC*VPC**2/TPC*DXBC
      .     -PPD*VPD**2/TPD*DXCA
      .     -R*(PPB*DXAB+PPC*DXBC+PPD*DXCA)))
C--ENERGY EQUATION
C
C     DELTA(U)
C             A
       A(I,1,NEN,1)=S*PPA*UPA/TPA/DELT
      . -xywa*.5*
      .  PPB*((CP+.5/TPB*(3.*UPB**2+VPB**2))*DYAB
      .  -UPB*VPB/TPB*DXAB)
      . -xyia*.5*
      .  PPB*((CP+.5/TPB*(VPB**2+3.*UPB**2))*DYAB
      .  -UPB*VPB/TPB*DXAB)
      . +xyea*.5*
      .  PPB*((CP+.5/TPB*(VPB**2+3.*UPB**2))*DYAB
      .  -UPB*VPB/TPB*DXAB)
C     Pseudo time term
       A(I,1,NEN,1)=A(I,1,NEN,1)
      .            +PSEUDO/DTAU*S*PPA*UPA/TPA
C     DELTA(V)
C             A
       A(I,1,NEN,2)=S*PPA*VPA/TPA/DELT
      . -xywa*.5*
```

```
     .  PPB*(UPB*VPB/TPB*DYAB
     .  -(CP+.5/TPB*(UPB**2+3.*VPB**2))*DXAB)
     . -xyia*.5*
     .  PPB*(UPB*VPB/TPB*DYAB
     .  -(CP+.5/TPB*(UPB**2+3.*VPB**2))*DXAB)
     . +xyea*.5*
     .  PPB*(UPB*VPB/TPB*DYAB
     .  -(CP+.5/TPB*(UPB**2+3.*VPB**2))*DXAB)
C    Pseudo time term
      A(I,1,NEN,2)=A(I,1,NEN,2)
     .            +PSEUDO/DTAU*S*PPA*VPA/TPA
C    DELTA(P)
C           A
      A(I,1,NEN,3)=S*(CP-R+.5/TPA*(UPA**2+VPA**2))/DELT
     . +xywa*.5*
     .      (CP+.5/TPB*(UPB**2+VPB**2))*(UPB*DYAB-VPB*DXAB)
     . +xyia*.5*
     .  DPC*(CP+.5/TPB*(UPB**2+VPB**2))*(UPB*DYAB-VPB*DXAB)
     . -xyea*.5*
     .  DPC*(CP+.5/TPB*(UPB**2+VPB**2))*(UPB*DYAB-VPB*DXAB)
C    Pseudo time term
      A(I,1,NEN,3)=A(I,1,NEN,3)
     .        +PSEUDO/DTAU*R*S*(CP-R+.5/TPA*(UPA**2+VPA**2))
C    DELTA(T)
C           A
      A(I,1,NEN,4)=-.5*S*PPA/TPA**2*(UPA**2+VPA**2)/DELT
     . -xywa*.25*PPB/TPB**2*(UPB**2+VPB**2)*(UPB*DYAB-VPB*DXAB)
     . +xyia*.25*PPB/TPB**2*(UPB**2+VPB**2)*(UPB*DYAB-VPB*DXAB)
     . -xyea*.25*PPB/TPB**2*(UPB**2+VPB**2)*(UPB*DYAB-VPB*DXAB)
C    Pseudo time term
      A(I,1,NEN,4)=A(I,1,NEN,4)
     .        -PSEUDO/DTAU*.5*S*PPA/TPA**2*(UPA**2+VPA**2)
C    DELTA(U)
C          B
      A(I,2,NEN,1)=xyb*.5*PPB*
     . ((CP+.5/TPB*(3.*UPB**2+VPB**2))*DYAB-UPB*VPB/TPB*DXAB)
C    DELTA(V)
C           B
      A(I,2,NEN,2)=xyb*.5*PPB*
     . (UPB*VPB/TPB*DYAB-(CP+.5/TPB*(UPB**2+3.*VPB**2))*DXAB)
C    DELTA(P)
C           B
      A(I,2,NEN,3)=xyb*.5*
     . (CP+.5/TPB*(UPB**2+VPB**2))*(UPB*DYAB-VPB*DXAB)
```

```
C     DELTA(T)
C             B
        A(I,2,NEN,4)=-xyb*.25*PPB/TPB**2*
      . (UPB**2+VPB**2)*(UPB*DYAB-VPB*DXAB)
C     DELTA(U)
C             C
        A(I,3,NEN,1)=.5*PPC*
      . ((CP+.5/TPC*(3.*UPC**2+VPC**2))*DYBC-UPC*VPC/TPC*DXBC)
C     DELTA(V)
C             C
        A(I,3,NEN,2)=.5*PPC*
      . (UPC*VPC/TPC*DYBC-(CP+.5/TPC*(UPC**2+3.*VPC**2))*DXBC)
C     DELTA(P)
C             C
        A(I,3,NEN,3)=.5*
      . (CP+.5/TPC*(UPC**2+VPC**2))*(UPC*DYBC-VPC*DXBC)
C     DELTA(T)
C             C
        A(I,3,NEN,4)=-.25*PPC/TPC**2*
      . (UPC**2+VPC**2)*(UPC*DYBC-VPC*DXBC)
C     DELTA(U)
C             D
        A(I,4,NEN,1)=.5*PPD*
      . ((CP+.5/TPD*(3.*UPD**2+VPD**2))*DYCA-UPD*VPD/TPD*DXCA)
C     DELTA(V)
C             D
        A(I,4,NEN,2)=.5*PPD*
      . (UPD*VPD/TPD*DYCA-(CP+.5/TPD*(UPD**2+3.*VPD**2))*DXCA)
C     DELTA(P)
C             D
        A(I,4,NEN,3)=.5*
      . (CP+.5/TPD*(UPD**2+VPD**2))*(UPD*DYCA-VPD*DXCA)
C     DELTA(T)
C             D
        A(I,4,NEN,4)=-.25*PPD/TPD**2*
      . (UPD**2+VPD**2)*(UPD*DYCA-VPD*DXCA)
C     RHS
C
        B(I,NEN)=-(S*(((CP-R)*PPA+.5*PPA/TPA*(UPA**2+VPA**2))
      .   -((CP-R)*PA+.5*PA/TA*(UA**2+VA**2)))/DELT
      . +.5*(PPB*(CP+.5/TPB*(UPB**2+VPB**2))*UPB*DYAB
      .      +PPC*(CP+.5/TPC*(UPC**2+VPC**2))*UPC*DYBC
```

```
     .     +PPD*(CP+.5/TPD*(UPD**2+VPD**2))*UPD*DYCA
     .     -PPB*(CP+.5/TPB*(UPB**2+VPB**2))*VPB*DXAB
     .     -PPC*(CP+.5/TPC*(UPC**2+VPC**2))*VPC*DXBC
     .     -PPD*(CP+.5/TPD*(UPD**2+VPD**2))*VPD*DXCA))

    1 CONTINUE
      RETURN
      END



*DECK VISC
      SUBROUTINE VISC
*CALL COMMZ


      DO 1 I=1,NCT

      ICA=I
      ICB=NUMEL(I,2)
      ICC=NUMEL(I,3)
      ICD=NUMEL(I,4)
      ICE=NUMEL(I,5)
      ICF=NUMEL(I,6)
      ICG=NUMEL(I,7)
      ICH=NUMEL(I,8)
      ICI=NUMEL(I,9)
      ICJ=NUMEL(I,10)
C---NODES OF CELL I
      N1=NCELL(4,ICA)
      N2=NCELL(5,ICA)
      N3=NCELL(6,ICA)
C-----Find Nodes D, E, F
      N4=NCELL(4,ICB)+NCELL(5,ICB)+NCELL(6,ICB)-N1-N2
      N5=NCELL(4,ICC)+NCELL(5,ICC)+NCELL(6,ICC)-N2-N3
      N6=NCELL(4,ICD)+NCELL(5,ICD)+NCELL(6,ICD)-N3-N1

C------Compute metrics on edges
      if(icb.gt.0)then
       X4=X(n4)
       Y4=Y(n4)
      else
C        Symmetry of cell B and A gives node d
      a1=x(n1)
      a2=y(n1)
      b1=x(n2)
      b2=y(n2)
      c1=x(n3)
      c2=y(n3)
      t4=((a1-b1)*(a1-c1)+(a2-b2)*(a2-c2))/
     .  ((b1-a1)**2+(b2-a2)**2)
      X4=c1+2.*(a1-c1+t4*(b1-a1))
      Y4=c2+2.*(a2-c2+t4*(b2-a2))
      endif
C temporary fix for periodic boundary with pitch of 3.0
c      pitch=3.0
c      npf=3971
c      n1n2l=201
c      n1n2u=253
c      if(ncell(1,ica).ge.npf)then
c       n4=ncell(6,icb)
c       X4=X(n4)
c       if(n1+n2.le.n1n2l)y4=y(n4)-pitch
c       if(n1+n2.ge.n1n2u)y4=y(n4)+pitch
c      endif
      DXAB=X(N2)-X(N1)
      DYAB=Y(N2)-Y(N1)
      DXBC=X(N3)-X(N2)
      DYBC=Y(N3)-Y(N2)
      DXCA=X(N1)-X(N3)
      DYCA=Y(N1)-Y(N3)
      DXAD=X4-X(N1)
      DYAD=Y4-Y(N1)
      DXDB=X(N2)-X4
      DYDB=Y(N2)-Y4
      DXBE=X(N5)-X(N2)
      DYBE=Y(N5)-Y(N2)
      DXEC=X(N3)-X(N5)
      DYEC=Y(N3)-Y(N5)
      DXCF=X(N6)-X(N3)
      DYCF=Y(N6)-Y(N3)
      DXFA=X(N1)-X(N6)
      DYFA=Y(N1)-Y(N6)
C-------AREA OF Quad. CELL I
      SA=VOL(ICA)
      if(icb.gt.0)then
       SB=VOL(ICB)
      else
       SB=VOL(ICA)
      endif
      SC=VOL(ICC)
      SD=VOL(ICD)
      SAB=SA+SB
      SAC=SA+SC
      SAD=SA+SD
C-------VARIABLES OF CELL A, B, C, D, E, F, G, H, I, AND J
C    CELL A
      UA=U(ICA)
      VA=V(ICA)
      TA=T(ICA)
C    PROVISIONAL VALUES
      UPA=UP(ICA)
      VPA=VP(ICA)
      TPA=TP(ICA)

       xyb=1.
       xye=1.
       xyf=1.
      xyg=1.
      xyh=1.
      xyi=1.
      xyj=1.
      xywab=0.
      xywad=0.
      xywdb=0.
      xywbe=0.
      xywec=0.
      xywcf=0.
      xywfa=0.
      xyiab=0.
      xyiad=0.
      xyidb=0.
      xyibe=0.
      xyiec=0.
      xyicf=0.
      xyifa=0.
      xyeab=0.
      xyead=0.
      xyedb=0.
      xyebe=0.
      xyeec=0.
      xyecf=0.
      xyefa=0.
C     CELL B
     if(icb.gt.0)then
      UPB=UP(ICB)
      VPB=VP(ICB)
      TPB=TP(ICB)
     else
     if(icb.eq.0)then
      xyb=0.0
      xye=0.0
      xyf=0.0
      xywab=1.
      UPB=-UPA
      VPB=-VPA
      TPB=TPA
      UPE=-UP(ICD)
      VPE=-VP(ICD)
      TPE=TP(ICD)
      UPF=-UP(ICC)
      VPF=-VP(ICC)
      TPF=TP(ICC)
     else
     if(icb.eq.-1)then
      xyb=0.0
      xye=0.0
      xyf=0.0
      xyiab=1.
      UPB=2.*UI(ICA)-UPA
      VPB=2.*VI(ICA)-VPA
```

```
       TPB=2.*TI(ICA)-TPA
       UPE=UPB
       VPE=VPB
       TPE=TPB
       UPF=UPB
       VPF=VPB
       TPF=TPB
      else
      if(icb.eq.-2)then
       xyb=0.0
       xye=0.0
       xyf=0.0
       xyeab=1.
       UPB=UPA
       VPB=VPA
       TPB=TPA
       UPE=UP(ICD)
       VPE=VP(ICD)
       TPE=TP(ICD)
       UPF=UP(ICC)
       VPF=VP(ICC)
       TPF=TP(ICC)
      endif
      endif
      endif
      endif
C     CELL C
      UPC=UP(ICC)
      VPC=VP(ICC)
      TPC=TP(ICC)
C     CELL D
      UPD=UP(ICD)
      VPD=VP(ICD)
      TPD=TP(ICD)
C     CELL E
      if(icb.gt.0)then
       if(ice.gt.0)then
        UPE=UP(ICE)
        VPE=VP(ICE)
        TPE=TP(ICE)
       else
       if(ice.eq.0)then
        xye=0.0
        xywad=1.0
        UPE=-UPB
        VPE=-VPB
        TPE=TPB
       else
       if(ice.eq.-1)then
        xye=0.0
        xyiad=1.0
        UPE=2.*UI(ICB)-UPB
        VPE=2.*VI(ICB)-VPB
        TPE=2.*TI(ICB)-TPB
       else
       if(ice.eq.-2)then
        xye=0.0
        xyead=1.0
        UPE=UPB
        VPE=VPB
        TPE=TPB
       endif
       endif
       endif
       endif
      endif
C     CELL F
      if(icb.gt.0)then
       if(icf.gt.0)then
        UPF=UP(ICF)
        VPF=VP(ICF)
        TPF=TP(ICF)
       else
       if(icf.eq.0)then
        xyf=0.0
        xywdb=1.0
        UPF=-UPB
        VPF=-VPB
        TPF=TPB
       else
       if(icf.eq.-1)then
        xyf=0.0
        xyidb=1.0
        UPF=2.*UI(ICB)-UPB
        VPF=2.*VI(ICB)-VPB
        TPF=2.*TI(ICB)-TPB
       else
       if(icf.eq.-2)then
        xyf=0.0
        xyedb=1.0
        UPF=UPB
        VPF=VPB
        TPF=TPB
       endif
       endif
       endif
       endif
      endif
C     CELL G
      if(icg.gt.0)then
       UPG=UP(ICG)
       VPG=VP(ICG)
       TPG=TP(ICG)
      else
      if(icg.eq.0)then
       xyg=0.0
       xywbe=1.0
       UPG=-UPC
       VPG=-VPC
       TPG=TPC
      else
      if(icg.eq.-1)then
       xyg=0.0
       xyibe=1.0
       UPG=2.*UI(ICC)-UPC
```

```
        VPG=2.*VI(ICC)-VPC
        TPG=2.*TI(ICC)-TPC
       else
       if(icg.eq.-2)then
        xyg=0.0
        xyebe=1.0
        UPG=UPC
        VPG=VPC
        TPG=TPC
       endif
       endif
       endif
       endif
C     CELL H
       if(ich.gt.0)then
        UPH=UP(ICH)
        VPH=VP(ICH)
        TPH=TP(ICH)
       else
       if(ich.eq.0)then
        xyh=0.0
        xywec=1.0
        UPH=-UPC
        VPH=-VPC
        TPH=TPC
       else
       if(ich.eq.-1)then
        xyh=0.0
        xyiec=1.0
        UPH=2.*UI(ICC)-UPC
        VPH=2.*VI(ICC)-VPC
        TPH=2.*TI(ICC)-TPC
       else
       if(ich.eq.-2)then
        xyh=0.0
        xyeec=1.0
        UPH=UPC
        VPH=VPC
        TPH=TPC
       endif
       endif
       endif
       endif
C     CELL I
       if(ici.gt.0)then
        UPI=UP(ICI)
        VPI=VP(ICI)
        TPI=TP(ICI)
       else
       if(ici.eq.0)then
        xyi=0.0
        xywcf=1.0
        UPI=-UPD
        VPI=-VPD
        TPI=TPD
       else
       if(ici.eq.-1)then
        xyi=0.0
        xyicf=1.0
        UPI=2.*UI(ICD)-UPD
        VPI=2.*VI(ICD)-VPD
        TPI=2.*TI(ICD)-TPD
       else
       if(ici.eq.-2)then
        xyi=0.0
        xyecf=1.0
        UPI=UPD
        VPI=VPD
        TPI=TPD
       endif
       endif
       endif
       endif
C     CELL J
       if(icj.gt.0)then
       UPJ=UP(ICJ)
       VPJ=VP(ICJ)
       TPJ=TP(ICJ)
       else
       if(icj.eq.0)then
        xyj=0.0
        xywfa=1.0
        UPJ=-UPD
        VPJ=-VPD
        TPJ=TPD
       else
       if(icj.eq.-1)then
        xyj=0.0
        xyifa=1.0
        UPJ=2.*UI(ICD)-UPD
        VPJ=2.*VI(ICD)-VPD
        TPJ=2.*TI(ICD)-TPD
       else
       if(icj.eq.-2)then
        xyj=0.0
        xyefa=1.0
        UPJ=UPD
        VPJ=VPD
        TPJ=TPD
       endif
       endif
       endif
       endif
C     Coefficients of equations
C     Use nondimensional Sutherland's Law to compute viscosity

       XMU=SC1*SQRT(TPA**3)/(TPA+SC2)

       CCOEF =.5*XMU*R/RENO
       CCOEFF=.5*XMU*R/RENO*CP/PR

C--X MOMENTUM EQUATION

C      DELTA(U)
C              A
```

```
      A(ICA,1,NXM,1)=A(ICA,1,NXM,1)-CCOEF*
     .    (4./3.*(DYAB/SAB*(DYBC+DYCA)
     .           +DYBC/SAC*(DYAB+DYCA)
     .           +DYCA/SAD*(DYAB+DYBC))
     .           +DXAB/SAB*(DXBC+DXCA)
     .           +DXBC/SAC*(DXAB+DXCA)
     .           +DXCA/SAD*(DXAB+DXBC))
     . +xywab*CCOEF*
     .  (4./3.*(DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB)
     .         +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB)
     . +xyiab*CCOEF*
     .  (4./3.*(DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB)
     .         +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB
     .  +4./3.*DYAB/SAB*DYAD + DXAB/SAB*DXAD
     .  +4./3.*DYAB/SAB*DYDB + DXAB/SAB*DXDB)
     . -xyeab*CCOEF*
     .  (4./3.*(DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB)
     .         +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB)
C     DELTA(V)
C             A
      A(ICA,1,NXM,2)=A(ICA,1,NXM,2)+CCOEF*
     .   (-2./3.*(DYAB/SAB*(DXBC+DXCA)
     .           +DYBC/SAC*(DXAB+DXCA)
     .           +DYCA/SAD*(DXAB+DXBC))
     .           +DXAB/SAB*(DYBC+DYCA)
     .           +DXBC/SAC*(DYAB+DYCA)
     .           +DXCA/SAD*(DYAB+DYBC))
     . -xywab*CCOEF*
     .  (-2./3.*(DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB)
     .          +DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB)
     . -xyiab*CCOEF*
     .  (-2./3.*(DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB)
     .          +DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB
     .   -2./3.*DYAB/SAB*DXAD + DXAB/SAB*DYAD
     .   -2./3.*DYAB/SAB*DXDB + DXAB/SAB*DYDB)
     . +xyeab*CCOEF*
     .  (-2./3.*(DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB)
     .          +DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB)
C     DELTA(U)
C             B
      A(ICA,2,NXM,1)=A(ICA,2,NXM,1)-xyb*CCOEF*
     . (4./3.*(DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB)
     .        +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB)
     . +xywad*CCOEF*(4./3.*DYAB/SAB*DYAD + DXAB/SAB*DXAD)
     . +xywdb*CCOEF*(4./3.*DYAB/SAB*DYDB + DXAB/SAB*DXDB)
     . +xyiad*CCOEF*(4./3.*DYAB/SAB*DYAD + DXAB/SAB*DXAD)
     . +xyidb*CCOEF*(4./3.*DYAB/SAB*DYDB + DXAB/SAB*DXDB)
     . -xyead*CCOEF*(4./3.*DYAB/SAB*DYAD + DXAB/SAB*DXAD)
     . -xyedb*CCOEF*(4./3.*DYAB/SAB*DYDB + DXAB/SAB*DXDB)
C     DELTA(V)
C             B
```

```
      A(ICA,2,NXM,2)=A(ICA,2,NXM,2)+xyb*CCOEF*
     . (-2./3.*(DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB)
     .          +DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB)
     . -xywad*CCOEF*(-2./3.*DYAB/SAB*DXAD + DXAB/SAB*DYAD)
     . -xywdb*CCOEF*(-2./3.*DYAB/SAB*DXDB + DXAB/SAB*DYDB)
     . -xyiad*CCOEF*(-2./3.*DYAB/SAB*DXAD + DXAB/SAB*DYAD)
     . -xyidb*CCOEF*(-2./3.*DYAB/SAB*DXDB + DXAB/SAB*DYDB)
     . +xyead*CCOEF*(-2./3.*DYAB/SAB*DXAD + DXAB/SAB*DYAD)
     . +xyedb*CCOEF*(-2./3.*DYAB/SAB*DXDB + DXAB/SAB*DYDB)
C     DELTA(U)
C             C
      A(ICA,3,NXM,1)=A(ICA,3,NXM,1)-CCOEF*
     . (4./3.*(DYAB/SAB*DYBC+DYBC/SAC*(DYBE+DYEC)+DYCA/SAD*DYBC)
     .         +DXAB/SAB*DXBC+DXBC/SAC*(DXBE+DXEC)+DXCA/SAD*DXBC)
     . +xywab*CCOEF*(4./3.*DYAB/SAB*DYDB + DXAB/SAB*DXDB)
     . +xywbe*CCOEF*(4./3.*DYBC/SAC*DYBE + DXBC/SAC*DXBE)
     . +xywec*CCOEF*(4./3.*DYBC/SAC*DYEC + DXBC/SAC*DXEC)
     . +xyibe*CCOEF*(4./3.*DYBC/SAC*DYBE + DXBC/SAC*DXBE)
     . +xyiec*CCOEF*(4./3.*DYBC/SAC*DYEC + DXBC/SAC*DXEC)
     . -xyeab*CCOEF*(4./3.*DYAB/SAB*DYDB + DXAB/SAB*DXDB)
     . -xyebe*CCOEF*(4./3.*DYBC/SAC*DYBE + DXBC/SAC*DXBE)
     . -xyeec*CCOEF*(4./3.*DYBC/SAC*DYEC + DXBC/SAC*DXEC)
C     DELTA(V)
C             C
      A(ICA,3,NXM,2)=A(ICA,3,NXM,2)+CCOEF*
     . (-2./3.*(DYAB/SAB*DXBC+DYBC/SAC*(DXBE+DXEC)+DYCA/SAD*DXBC)
     .          +DXAB/SAB*DYBC+DXBC/SAC*(DYBE+DYEC)+DXCA/SAD*DYBC)
     . -xywab*CCOEF*(-2./3.*DYAB/SAB*DXDB + DXAB/SAB*DYDB)
     . -xywbe*CCOEF*(-2./3.*DYBC/SAC*DXBE + DXBC/SAC*DYBE)
     . -xywec*CCOEF*(-2./3.*DYBC/SAC*DXEC + DXBC/SAC*DYEC)
     . -xyibe*CCOEF*(-2./3.*DYBC/SAC*DXBE + DXBC/SAC*DYBE)
     . -xyiec*CCOEF*(-2./3.*DYBC/SAC*DXEC + DXBC/SAC*DYEC)
     . +xyeab*CCOEF*(-2./3.*DYAB/SAB*DXDB + DXAB/SAB*DYDB)
     . +xyebe*CCOEF*(-2./3.*DYBC/SAC*DXBE + DXBC/SAC*DYBE)
     . +xyeec*CCOEF*(-2./3.*DYBC/SAC*DXEC + DXBC/SAC*DYEC)
C     DELTA(U)
C             D
      A(ICA,4,NXM,1)=A(ICA,4,NXM,1)-CCOEF*
     . (4./3.*(DYAB/SAB*DYCA+DYBC/SAC*DYCA+DYCA/SAD*(DYCF+DYFA))
     .         +DXAB/SAB*DXCA+DXBC/SAC*DXCA+DXCA/SAD*(DXCF+DXFA))
     . +xywab*CCOEF*(4./3.*DYAB/SAB*DYAD + DXAB/SAB*DXAD)
     . +xywcf*CCOEF*(4./3.*DYCA/SAD*DYCF + DXCA/SAD*DXCF)
     . +xywfa*CCOEF*(4./3.*DYCA/SAD*DYFA + DXCA/SAD*DXFA)
     . +xyicf*CCOEF*(4./3.*DYCA/SAD*DYCF + DXCA/SAD*DXCF)
     . +xyifa*CCOEF*(4./3.*DYCA/SAD*DYFA + DXCA/SAD*DXFA)
     . -xyeab*CCOEF*(4./3.*DYAB/SAB*DYAD + DXAB/SAB*DXAD)
     . -xyecf*CCOEF*(4./3.*DYCA/SAD*DYCF + DXCA/SAD*DXCF)
     . -xyefa*CCOEF*(4./3.*DYCA/SAD*DYFA + DXCA/SAD*DXFA)
C      DELTA(V)
C              D
       A(ICA,4,NXM,2)=A(ICA,4,NXM,2)+CCOEF*
     . (-2./3.*(DYAB/SAB*DXCA+DYBC/SAC*DXCA+DYCA/SAD*(DXCF+DXFA))
     .          +DXAB/SAB*DYCA+DXBC/SAC*DYCA+DXCA/SAD*(DYCF+DYFA))
     . -xywab*CCOEF*(-2./3.*DYAB/SAB*DXAD + DXAB/SAB*DYAD)
     . -xywcf*CCOEF*(-2./3.*DYCA/SAD*DXCF + DXCA/SAD*DYCF)
     . -xywfa*CCOEF*(-2./3.*DYCA/SAD*DXFA + DXCA/SAD*DYFA)
     . -xyicf*CCOEF*(-2./3.*DYCA/SAD*DXCF + DXCA/SAD*DYCF)
     . -xyifa*CCOEF*(-2./3.*DYCA/SAD*DXFA + DXCA/SAD*DYFA)
     . +xyeab*CCOEF*(-2./3.*DYAB/SAB*DXAD + DXAB/SAB*DYAD)
     . +xyecf*CCOEF*(-2./3.*DYCA/SAD*DXCF + DXCA/SAD*DYCF)
     . +xyefa*CCOEF*(-2./3.*DYCA/SAD*DXFA + DXCA/SAD*DYFA)
C      DELTA(U)
C              E
       A(ICA,5,NXM,1)=-xye*CCOEF*
     . (4./3.*DYAB/SAB*DYAD + DXAB/SAB*DXAD)
C      DELTA(V)
C              E
       A(ICA,5,NXM,2)=xye*CCOEF*
     .(-2./3.*DYAB/SAB*DXAD + DXAB/SAB*DYAD)
C      DELTA(U)
C              F
       A(ICA,6,NXM,1)=-xyf*CCOEF*
     . (4./3.*DYAB/SAB*DYDB + DXAB/SAB*DXDB)
C      DELTA(V)
C              F
       A(ICA,6,NXM,2)=xyf*CCOEF*
     .(-2./3.*DYAB/SAB*DXDB + DXAB/SAB*DYDB)
C      DELTA(U)
C              G
       A(ICA,7,NXM,1)=-xyg*CCOEF*
     . (4./3.*DYBC/SAC*DYBE + DXBC/SAC*DXBE)
C      DELTA(V)
C              G
       A(ICA,7,NXM,2)=xyg*CCOEF*
     .(-2./3.*DYBC/SAC*DXBE + DXBC/SAC*DYBE)
C      DELTA(U)
C              H
       A(ICA,8,NXM,1)=-xyh*CCOEF*
     . (4./3.*DYBC/SAC*DYEC + DXBC/SAC*DXEC)
```

```
C     DELTA(V)
C             H
      A(ICA,8,NXM,2)=xyh*CCOEF*
     .(-2./3.*DYBC/SAC*DXEC + DXBC/SAC*DYEC)
C     DELTA(U)
C             I
      A(ICA,9,NXM,1)=-xyi*CCOEF*
     . (4./3.*DYCA/SAD*DYCF + DXCA/SAD*DXCF)
C     DELTA(V)
C             I
      A(ICA,9,NXM,2)=xyi*CCOEF*
     .(-2./3.*DYCA/SAD*DXCF + DXCA/SAD*DYCF)
C     DELTA(U)
C             J
      A(ICA,10,NXM,1)=-xyj*CCOEF*
     . (4./3.*DYCA/SAD*DYFA + DXCA/SAD*DXFA)
C     DELTA(V)
C             J
      A(ICA,10,NXM,2)=xyj*CCOEF*
     .(-2./3.*DYCA/SAD*DXFA + DXCA/SAD*DYFA)
C     RHS(split into 2 parts)
C

      B(ICA,NXM)=B(ICA,NXM) - CCOEF*
     . (-4./3.*(DYAB/SAB*((UPB+UPE)*DYAD+(UPB+UPF)*DYDB
     .                   +(UPA+UPC)*DYBC+(UPA+UPD)*DYCA)
     .         +DYBC/SAC*((UPA+UPB)*DYAB+(UPC+UPG)*DYBE
     .                   +(UPC+UPH)*DYEC+(UPA+UPD)*DYCA)
     .         +DYCA/SAD*((UPA+UPB)*DYAB+(UPA+UPC)*DYBC
     .                   +(UPD+UPI)*DYCF+(UPD+UPJ)*DYFA))
     .  -2./3.*(DYAB/SAB*((VPB+VPE)*DXAD+(VPB+VPF)*DXDB
     .                   +(VPA+VPC)*DXBC+(VPA+VPD)*DXCA)
     .         +DYBC/SAC*((VPA+VPB)*DXAB+(VPC+VPG)*DXBE
     .                   +(VPC+VPH)*DXEC+(VPA+VPD)*DXCA)
     .         +DYCA/SAD*((VPA+VPB)*DXAB+(VPA+VPC)*DXBC
     .                   +(VPD+VPI)*DXCF+(VPD+VPJ)*DXFA)))
      B(ICA,NXM)=B(ICA,NXM) - CCOEF*
     .        (-DXAB/SAB*((UPB+UPE)*DXAD+(UPB+UPF)*DXDB
     .                   +(UPA+UPC)*DXBC+(UPA+UPD)*DXCA)
     .         -DXBC/SAC*((UPA+UPB)*DXAB+(UPC+UPG)*DXBE
     .                   +(UPC+UPH)*DXEC+(UPA+UPD)*DXCA)
     .         -DXCA/SAD*((UPA+UPB)*DXAB+(UPA+UPC)*DXBC
     .                   +(UPD+UPI)*DXCF+(UPD+UPJ)*DXFA)
     .         +DXAB/SAB*((VPB+VPE)*DYAD+(VPB+VPF)*DYDB
     .                   +(VPA+VPC)*DYBC+(VPA+VPD)*DYCA)
     .         +DXBC/SAC*((VPA+VPB)*DYAB+(VPC+VPG)*DYBE
     .                +(VPC+VPH)*DYEC+(VPA+VPD)*DYCA)
     .          +DXCA/SAD*((VPA+VPB)*DYAB+(VPA+VPC)*DYBC
     .                +(VPD+VPI)*DYCF+(VPD+VPJ)*DYFA))


C--Y MOMENTUM EQUATION
C
C     DELTA(U)
C             A

      A(ICA,1,NYM,1)=A(ICA,1,NYM,1)+CCOEF*
     .           (DYAB/SAB*(DXBC+DXCA)
     .           +DYBC/SAC*(DXAB+DXCA)
     .           +DYCA/SAD*(DXAB+DXBC)
     .    -2./3.*(DXAB/SAB*(DYBC+DYCA)
     .           +DXBC/SAC*(DYAB+DYCA)
     .           +DXCA/SAD*(DYAB+DYBC)))
     . -xywab*CCOEF*
     .         (DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB
     .  -2./3.*(DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB))
     . -xyiab*CCOEF*
     .         (DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB
     .  -2./3.*(DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB)
     .        +DYAB/SAB*DXAD - 2./3.*DXAB/SAB*DYAD
     .        +DYAB/SAB*DXDB - 2./3.*DXAB/SAB*DYDB)
     . +xyeab*CCOEF*
     .         (DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB
     .  -2./3.*(DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB))
C     DELTA(V)
C             A

      A(ICA,1,NYM,2)=A(ICA,1,NYM,2)-CCOEF*
     .           (DYAB/SAB*(DYBC+DYCA)
     .           +DYBC/SAC*(DYAB+DYCA)
     .           +DYCA/SAD*(DYAB+DYBC)
     .    +4./3.*(DXAB/SAB*(DXBC+DXCA)
     .           +DXBC/SAC*(DXAB+DXCA)
     .           +DXCA/SAD*(DXAB+DXBC)))
     . +xywab*CCOEF*
     .         (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     .  +4./3.*(DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB))
     . +xyiab*CCOEF*
     .         (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     .  +4./3.*(DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB)
     .        +DYAB/SAB*DYAD + 4./3.*DXAB/SAB*DXAD
     .        +DYAB/SAB*DYDB + 4./3.*DXAB/SAB*DXDB)
     . -xyeab*CCOEF*
     .         (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     .  +4./3.*(DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB))
C     DELTA(U)
C             B

      A(ICA,2,NYM,1)=A(ICA,2,NYM,1)+xyb*CCOEF*
     .        (DYAB/SAB*(DXAD+DXDB)+DYBC/SAC*DXAB+DYCA/SAD*DXAB
     . -2./3.*(DXAB/SAB*(DYAD+DYDB)+DXBC/SAC*DYAB+DXCA/SAD*DYAB))
     . -xywad*CCOEF*(DYAB/SAB*DXAD - 2./3.*DXAB/SAB*DYAD)
     . -xywdb*CCOEF*(DYAB/SAB*DXDB - 2./3.*DXAB/SAB*DYDB)
     . -xyiad*CCOEF*(DYAB/SAB*DXAD - 2./3.*DXAB/SAB*DYAD)
     . -xyidb*CCOEF*(DYAB/SAB*DXDB - 2./3.*DXAB/SAB*DYDB)
     . +xyead*CCOEF*(DYAB/SAB*DXAD - 2./3.*DXAB/SAB*DYAD)
     . +xyedb*CCOEF*(DYAB/SAB*DXDB - 2./3.*DXAB/SAB*DYDB)
C     DELTA(V)
C             B
      A(ICA,2,NYM,2)=A(ICA,2,NYM,2)-xyb*CCOEF*
     .         (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     . +4./3.*(DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB))
     . +xywad*CCOEF*(DYAB/SAB*DYAD + 4./3.*DXAB/SAB*DXAD)
     . +xywdb*CCOEF*(DYAB/SAB*DYDB + 4./3.*DXAB/SAB*DXDB)
     . +xyiad*CCOEF*(DYAB/SAB*DYAD + 4./3.*DXAB/SAB*DXAD)
     . +xyidb*CCOEF*(DYAB/SAB*DYDB + 4./3.*DXAB/SAB*DXDB)
     . -xyead*CCOEF*(DYAB/SAB*DYAD + 4./3.*DXAB/SAB*DXAD)
     . -xyedb*CCOEF*(DYAB/SAB*DYDB + 4./3.*DXAB/SAB*DXDB)
C     DELTA(U)
C             C
      A(ICA,3,NYM,1)=A(ICA,3,NYM,1)+CCOEF*
     .         (DYAB/SAB*DXBC+DYBC/SAC*(DXBE+DXEC)+DYCA/SAD*DXBC
     . -2./3.*(DXAB/SAB*DYBC+DXBC/SAC*(DYBE+DYEC)+DXCA/SAD*DYBC))
     . -xywab*CCOEF*(DYAB/SAB*DXDB - 2./3.*DXAB/SAB*DYDB)
     . -xywbe*CCOEF*(DYBC/SAC*DXBE - 2./3.*DXBC/SAC*DYBE)
     . -xywec*CCOEF*(DYBC/SAC*DXEC - 2./3.*DXBC/SAC*DYEC)
     . -xyibe*CCOEF*(DYBC/SAC*DXBE - 2./3.*DXBC/SAC*DYBE)
     . -xyiec*CCOEF*(DYBC/SAC*DXEC - 2./3.*DXBC/SAC*DYEC)
     . +xyeab*CCOEF*(DYAB/SAB*DXDB - 2./3.*DXAB/SAB*DYDB)
     . +xyebe*CCOEF*(DYBC/SAC*DXBE - 2./3.*DXBC/SAC*DYBE)
     . +xyeec*CCOEF*(DYBC/SAC*DXEC - 2./3.*DXBC/SAC*DYEC)
C     DELTA(V)
C             C
      A(ICA,3,NYM,2)=A(ICA,3,NYM,2)-CCOEF*
     .         (DYAB/SAB*DYBC+DYBC/SAC*(DYBE+DYEC)+DYCA/SAD*DYBC
     . +4./3.*(DXAB/SAB*DXBC+DXBC/SAC*(DXBE+DXEC)+DXCA/SAD*DXBC))
     . +xywab*CCOEF*(DYAB/SAB*DYDB + 4./3.*DXAB/SAB*DXDB)
     . +xywbe*CCOEF*(DYBC/SAC*DYBE + 4./3.*DXBC/SAC*DXBE)
     . +xywec*CCOEF*(DYBC/SAC*DYEC + 4./3.*DXBC/SAC*DXEC)
     . +xyibe*CCOEF*(DYBC/SAC*DYBE + 4./3.*DXBC/SAC*DXBE)
     . +xyiec*CCOEF*(DYBC/SAC*DYEC + 4./3.*DXBC/SAC*DXEC)
     . -xyeab*CCOEF*(DYAB/SAB*DYDB + 4./3.*DXAB/SAB*DXDB)
     . -xyebe*CCOEF*(DYBC/SAC*DYBE + 4./3.*DXBC/SAC*DXBE)
     . -xyeec*CCOEF*(DYBC/SAC*DYEC + 4./3.*DXBC/SAC*DXEC)
C     DELTA(U)
C             D
```

```
      A(ICA,4,NYM,1)=A(ICA,4,NYM,1)+CCOEF*
     .        (DYAB/SAB*DXCA+DYBC/SAC*DXCA+DYCA/SAD*(DXCF+DXFA)
     . -2./3.*(DXAB/SAB*DYCA+DXBC/SAC*DYCA+DXCA/SAD*(DYCF+DYFA)))
     . -xywab*CCOEF*(DYAB/SAB*DXAD - 2./3.*DXAB/SAB*DYAD)
     . -xywcf*CCOEF*(DYCA/SAD*DXCF - 2./3.*DXCA/SAD*DYCF)
     . -xywfa*CCOEF*(DYCA/SAD*DXFA - 2./3.*DXCA/SAD*DYFA)
     . -xyicf*CCOEF*(DYCA/SAD*DXCF - 2./3.*DXCA/SAD*DYCF)
     . -xyifa*CCOEF*(DYCA/SAD*DXFA - 2./3.*DXCA/SAD*DYFA)
     . +xyeab*CCOEF*(DYAB/SAB*DXAD - 2./3.*DXAB/SAB*DYAD)
     . +xyecf*CCOEF*(DYCA/SAD*DXCF - 2./3.*DXCA/SAD*DYCF)
     . +xyefa*CCOEF*(DYCA/SAD*DXFA - 2./3.*DXCA/SAD*DYFA)
C     DELTA(V)
C             D
      A(ICA,4,NYM,2)=A(ICA,4,NYM,2)-CCOEF*
     .        (DYAB/SAB*DYCA+DYBC/SAC*DYCA+DYCA/SAD*(DYCF+DYFA)
     . +4./3.*(DXAB/SAB*DXCA+DXBC/SAC*DXCA+DXCA/SAD*(DXCF+DXFA)))
     . +xywab*CCOEF*(DYAB/SAB*DYAD + 4./3.*DXAB/SAB*DXAD)
     . +xywcf*CCOEF*(DYCA/SAD*DYCF + 4./3.*DXCA/SAD*DXCF)
     . +xywfa*CCOEF*(DYCA/SAD*DYFA + 4./3.*DXCA/SAD*DXFA)
     . +xyicf*CCOEF*(DYCA/SAD*DYCF + 4./3.*DXCA/SAD*DXCF)
     . +xyifa*CCOEF*(DYCA/SAD*DYFA + 4./3.*DXCA/SAD*DXFA)
     . -xyeab*CCOEF*(DYAB/SAB*DYAD + 4./3.*DXAB/SAB*DXAD)
     . -xyecf*CCOEF*(DYCA/SAD*DYCF + 4./3.*DXCA/SAD*DXCF)
     . -xyefa*CCOEF*(DYCA/SAD*DYFA + 4./3.*DXCA/SAD*DXFA)
C     DELTA(U)
C             E
      A(ICA,5,NYM,1)=xye*CCOEF*
     .(DYAB/SAB*DXAD - 2./3.*DXAB/SAB*DYAD)
C     DELTA(V)
C             E
      A(ICA,5,NYM,2)=-xye*CCOEF*
     .(DYAB/SAB*DYAD + 4./3.*DXAB/SAB*DXAD)
C     DELTA(U)
C             F
      A(ICA,6,NYM,1)=xyf*CCOEF*
     .(DYAB/SAB*DXDB - 2./3.*DXAB/SAB*DYDB)
C     DELTA(V)
C             F
      A(ICA,6,NYM,2)=-xyf*CCOEF*
     .(DYAB/SAB*DYDB + 4./3.*DXAB/SAB*DXDB)
C     DELTA(U)
C             G
      A(ICA,7,NYM,1)=xyg*CCOEF*
     .(DYBC/SAC*DXBE - 2./3.*DXBC/SAC*DYBE)
```

```
C     DELTA(V)
C             G
      A(ICA,7,NYM,2)=-xyg*CCOEF*
     .(DYBC/SAC*DYBE + 4./3.*DXBC/SAC*DXBE)
C     DELTA(U)
C             H
      A(ICA,8,NYM,1)=xyh*CCOEF*
     .(DYBC/SAC*DXEC - 2./3.*DXBC/SAC*DYEC)
C     DELTA(V)
C             H
      A(ICA,8,NYM,2)=-xyh*CCOEF*
     .(DYBC/SAC*DYEC + 4./3.*DXBC/SAC*DXEC)
C     DELTA(U)
C             I
      A(ICA,9,NYM,1)=xyi*CCOEF*
     .(DYCA/SAD*DXCF - 2./3.*DXCA/SAD*DYCF)
C     DELTA(V)
C             I
      A(ICA,9,NYM,2)=-xyi*CCOEF*
     .(DYCA/SAD*DYCF + 4./3.*DXCA/SAD*DXCF)
C     DELTA(U)
C             J
      A(ICA,10,NYM,1)=xyj*CCOEF*
     .(DYCA/SAD*DXFA - 2./3.*DXCA/SAD*DYFA)
C     DELTA(V)
C             J
      A(ICA,10,NYM,2)=-xyj*CCOEF*
     .(DYCA/SAD*DYFA + 4./3.*DXCA/SAD*DXFA)


C     RHS(split into 2 parts)
C
      B(ICA,NYM)=B(ICA,NYM) - CCOEF*
     .        (-DYAB/SAB*((VPB+VPE)*DYAD+(VPB+VPF)*DYDB
     .                   +(VPA+VPC)*DYBC+(VPA+VPD)*DYCA)
     .         -DYBC/SAC*((VPA+VPB)*DYAB+(VPC+VPG)*DYBE
     .                   +(VPC+VPH)*DYEC+(VPA+VPD)*DYCA)
     .         -DYCA/SAD*((VPA+VPB)*DYAB+(VPA+VPC)*DYBC
     .                   +(VPD+VPI)*DYCF+(VPD+VPJ)*DYFA)
     .         +DYAB/SAB*((UPB+UPE)*DXAD+(UPB+UPF)*DXDB
     .                   +(UPA+UPC)*DXBC+(UPA+UPD)*DXCA)
     .         +DYBC/SAC*((UPA+UPB)*DXAB+(UPC+UPG)*DXBE
     .                   +(UPC+UPH)*DXEC+(UPA+UPD)*DXCA)
     .         +DYCA/SAD*((UPA+UPB)*DXAB+(UPA+UPC)*DXBC
     .                   +(UPD+UPI)*DXCF+(UPD+UPJ)*DXFA))
```

```
      B(ICA,NYM)=B(ICA,NYM) - CCOEF*
     . (-4./3.*(DXAB/SAB*((VPB+VPE)*DXAD+(VPB+VPF)*DXDB
     .                  +(VPA+VPC)*DXBC+(VPA+VPD)*DXCA)
     .         +DXBC/SAC*((VPA+VPB)*DXAB+(VPC+VPG)*DXBE
     .                  +(VPC+VPH)*DXEC+(VPA+VPD)*DXCA)
     .         +DXCA/SAD*((VPA+VPB)*DXAB+(VPA+VPC)*DXBC
     .                  +(VPD+VPI)*DXCF+(VPD+VPJ)*DXFA))
     .  -2./3.*(DXAB/SAB*((UPB+UPE)*DYAD+(UPB+UPF)*DYDB
     .                  +(UPA+UPC)*DYBC+(UPA+UPD)*DYCA)
     .         +DXBC/SAC*((UPA+UPB)*DYAB+(UPC+UPG)*DYBE
     .                  +(UPC+UPH)*DYEC+(UPA+UPD)*DYCA)
     .         +DXCA/SAD*((UPA+UPB)*DYAB+(UPA+UPC)*DYBC
     .                  +(UPD+UPI)*DYCF+(UPD+UPJ)*DYFA)))

C--ENERGY EQUATION
C
C     DELTA(T)
C            A

      A(ICA,1,NEN,4)=A(ICA,1,NEN,4)-CCOEFF*
     .           (DYAB/SAB*(DYBC+DYCA)
     .           +DYBC/SAC*(DYAB+DYCA)
     .           +DYCA/SAD*(DYAB+DYBC)
     .           +DXAB/SAB*(DXBC+DXCA)
     .           +DXBC/SAC*(DXAB+DXCA)
     .           +DXCA/SAD*(DXAB+DXBC))
     .  -xywab*CCOEFF*
     .      (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     .      +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB)
     .  +xyiab*CCOEFF*
     .      (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     .      +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB
     .      +DYAB/SAB*DYAD+DXAB/SAB*DXAD
     .      +DYAB/SAB*DYDB+DXAB/SAB*DXDB)
     .  -xyeab*CCOEFF*
     .      (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     .      +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB)

C     DELTA(T)
C            B

      A(ICA,2,NEN,4)=A(ICA,2,NEN,4)-xyb*CCOEFF*
     .      (DYAB/SAB*(DYAD+DYDB)+DYBC/SAC*DYAB+DYCA/SAD*DYAB
     .      +DXAB/SAB*(DXAD+DXDB)+DXBC/SAC*DXAB+DXCA/SAD*DXAB)
     .  -xywad*CCOEFF*(DYAB/SAB*DYAD+DXAB/SAB*DXAD)
     .  -xywdb*CCOEFF*(DYAB/SAB*DYDB+DXAB/SAB*DXDB)
     .  +xyiad*CCOEFF*(DYAB/SAB*DYAD+DXAB/SAB*DXAD)
     .  +xyidb*CCOEFF*(DYAB/SAB*DYDB+DXAB/SAB*DXDB)
     .  -xyead*CCOEFF*(DYAB/SAB*DYAD+DXAB/SAB*DXAD)
     .  -xyedb*CCOEFF*(DYAB/SAB*DYDB+DXAB/SAB*DXDB)

C     DELTA(T)
C            C
```

```
      A(ICA,3,NEN,4)=A(ICA,3,NEN,4)-CCOEFF*
     .      (DYAB/SAB*DYBC+DYBC/SAC*(DYBE+DYEC)+DYCA/SAD*DYBC
     .      +DXAB/SAB*DXBC+DXBC/SAC*(DXBE+DXEC)+DXCA/SAD*DXBC)
     .  -xywab*CCOEFF*(DYAB/SAB*DYDB+DXAB/SAB*DXDB)
     .  -xywbe*CCOEFF*(DYBC/SAC*DYBE+DXBC/SAC*DXBE)
     .  -xywec*CCOEFF*(DYBC/SAC*DYEC+DXBC/SAC*DXEC)
     .  +xyibe*CCOEFF*(DYBC/SAC*DYBE+DXBC/SAC*DXBE)
     .  +xyiec*CCOEFF*(DYBC/SAC*DYEC+DXBC/SAC*DXEC)
     .  -xyeab*CCOEFF*(DYAB/SAB*DYDB+DXAB/SAB*DXDB)
     .  -xyebe*CCOEFF*(DYBC/SAC*DYBE+DXBC/SAC*DXBE)
     .  -xyeec*CCOEFF*(DYBC/SAC*DYEC+DXBC/SAC*DXEC)
C     DELTA(T)
C             D
      A(ICA,4,NEN,4)=A(ICA,4,NEN,4)-CCOEFF*
     .      (DYAB/SAB*DYCA+DYBC/SAC*DYCA+DYCA/SAD*(DYCF+DYFA)
     .      +DXAB/SAB*DXCA+DXBC/SAC*DXCA+DXCA/SAD*(DXCF+DXFA))
     .  -xywab*CCOEFF*(DYAB/SAB*DYAD +DXAB/SAB*DXAD)
     .  -xywcf*CCOEFF*(DYCA/SAD*DYCF+DXCA/SAD*DXCF)
     .  -xywfa*CCOEFF*(DYCA/SAD*DYFA+DXCA/SAD*DXFA)
     .  +xyicf*CCOEFF*(DYCA/SAD*DYCF+DXCA/SAD*DXCF)
     .  +xyifa*CCOEFF*(DYCA/SAD*DYFA+DXCA/SAD*DXFA)
     .  -xyeab*CCOEFF*(DYAB/SAB*DYAD +DXAB/SAB*DXAD)
     .  -xyecf*CCOEFF*(DYCA/SAD*DYCF+DXCA/SAD*DXCF)
     .  -xyefa*CCOEFF*(DYCA/SAD*DYFA+DXCA/SAD*DXFA)
C     DELTA(T)
C             E
      A(ICA,5,NEN,4)=-xye*CCOEFF*
     .           (DYAB/SAB*DYAD
     .           +DXAB/SAB*DXAD)
C     DELTA(T)
C             F
      A(ICA,6,NEN,4)=-xyf*CCOEFF*
     .           (DYAB/SAB*DYDB
     .           +DXAB/SAB*DXDB)
C     DELTA(T)
C             G
      A(ICA,7,NEN,4)=-xyg*CCOEFF*
     .           (DYBC/SAC*DYBE
     .           +DXBC/SAC*DXBE)
C     DELTA(T)
C             H
      A(ICA,8,NEN,4)=-xyh*CCOEFF*
     .           (DYBC/SAC*DYEC
     .           +DXBC/SAC*DXEC)
C     DELTA(T)
```

```
C           I
      A(ICA,9,NEN,4)=-xyi*CCOEFF*
     .           (DYCA/SAD*DYCF
     .           +DXCA/SAD*DXCF)
C     DELTA(T)
C             J
      A(ICA,10,NEN,4)=-xyj*CCOEFF*
     .           (DYCA/SAD*DYFA
     .           +DXCA/SAD*DXFA)

C     RHS(split into 3 parts)
C
      B(ICA,NEN)=B(ICA,NEN) + CCOEFF*
     .         (DYAB/SAB*((TPB+TPE)*DYAD+(TPB+TPF)*DYDB
     .                  +(TPA+TPC)*DYBC+(TPA+TPD)*DYCA)
     .         +DYBC/SAC*((TPA+TPB)*DYAB+(TPC+TPG)*DYBE
     .                  +(TPC+TPH)*DYEC+(TPA+TPD)*DYCA)
     .         +DYCA/SAD*((TPA+TPB)*DYAB+(TPA+TPC)*DYBC
     .                  +(TPD+TPI)*DYCF+(TPD+TPJ)*DYFA)
     .         +DXAB/SAB*((TPB+TPE)*DXAD+(TPB+TPF)*DXDB
     .                  +(TPA+TPC)*DXBC+(TPA+TPD)*DXCA)
     .         +DXBC/SAC*((TPA+TPB)*DXAB+(TPC+TPG)*DXBE
     .                  +(TPC+TPH)*DXEC+(TPA+TPD)*DXCA)
     .         +DXCA/SAD*((TPA+TPB)*DXAB+(TPA+TPC)*DXBC
     .                  +(TPD+TPI)*DXCF+(TPD+TPJ)*DXFA))
C       Lag dissipation terms
      DUXAB=.5/SAB*((UPB+UPE)*DYAD+(UPB+UPF)*DYDB
     .          +(UPA+UPC)*DYBC+(UPA+UPD)*DYCA)
      DUYAB=-.5/SAB*((UPB+UPE)*DXAD+(UPB+UPF)*DXDB
     .          +(UPA+UPC)*DXBC+(UPA+UPD)*DXCA)
      DVXAB=.5/SAB*((VPB+VPE)*DYAD+(VPB+VPF)*DYDB
     .          +(VPA+VPC)*DYBC+(VPA+VPD)*DYCA)
      DVYAB=-.5/SAB*((VPB+VPE)*DXAD+(VPB+VPF)*DXDB
     .          +(VPA+VPC)*DXBC+(VPA+VPD)*DXCA)
      DUXBC=.5/SAC*((UPA+UPB)*DYAB+(UPC+UPG)*DYBE
     .          +(UPC+UPH)*DYEC+(UPA+UPD)*DYCA)
      DUYBC=-.5/SAC*((UPA+UPB)*DXAB+(UPC+UPG)*DXBE
     .          +(UPC+UPH)*DXEC+(UPA+UPD)*DXCA)
      DVXBC=.5/SAC*((VPA+VPB)*DYAB+(VPC+VPG)*DYBE
     .          +(VPC+VPH)*DYEC+(VPA+VPD)*DYCA)
      DVYBC=-.5/SAC*((VPA+VPB)*DXAB+(VPC+VPG)*DXBE
     .          +(VPC+VPH)*DXEC+(VPA+VPD)*DXCA)
      DUXCA=.5/SAD*((UPA+UPB)*DYAB+(UPA+UPC)*DYBC
     .          +(UPD+UPI)*DYCF+(UPD+UPJ)*DYFA)
      DUYCA=-.5/SAD*((UPA+UPB)*DXAB+(UPA+UPC)*DXBC
     .          +(UPD+UPI)*DXCF+(UPD+UPJ)*DXFA)
      DVXCA=.5/SAD*((VPA+VPB)*DYAB+(VPA+VPC)*DYBC
     .          +(VPD+VPI)*DYCF+(VPD+VPJ)*DYFA)
      DVYCA=-.5/SAD*((VPA+VPB)*DXAB+(VPA+VPC)*DXBC
     .          +(VPD+VPI)*DXCF+(VPD+VPJ)*DXFA)
```

```
      B(ICA,NEN)=B(ICA,NEN) - CCOEF*
     . (-(2./3.*(UPA+UPB)*(2.*DUXAB-DVYAB)
     . +(VPA+VPB)*(DVXAB+DUYAB))*DYAB
     .  -(2./3.*(UPA+UPC)*(2.*DUXBC-DVYBC)
     . +(VPA+VPC)*(DVXBC+DUYBC))*DYBC
     .  -(2./3.*(UPA+UPD)*(2.*DUXCA-DVYCA)
     . +(VPA+VPD)*(DVXCA+DUYCA))*DYCA
     .  +((UPA+UPB)*(DVXAB+DUYAB)
     . +2./3.*(VPA+VPB)*(2.*DVYAB-DUXAB))*DXAB
     .  +((UPA+UPC)*(DVXBC+DUYBC)
     . +2./3.*(VPA+VPC)*(2.*DVYBC-DUXBC))*DXBC
     .  +((UPA+UPD)*(DVXCA+DUYCA)
     . +2./3.*(VPA+VPD)*(2.*DVYCA-DUXCA))*DXCA)

    1 CONTINUE

      RETURN
      END



*DECK ENSCALE
      SUBROUTINE ENSCALE
*CALL COMMZ
C     Rescales the energy equation by Cp
      RCP=1./CP
      DO 1 L=1,4
      DO 1 J=1,NCPL
       DO 1 I=1,NCT
        A(I,J,NEN,L)=A(I,J,NEN,L)*RCP
    1 CONTINUE
      DO 2 I=1,NCT
       B(I,NEN)=B(I,NEN)*RCP
    2 CONTINUE
      RETURN
      END



*DECK SOLVE
      SUBROUTINE SOLVE
*CALL COMMZ
      IF(NSOLVE.EQ.0)CALL GAUSSV
      IF(NSOLVE.EQ.1)CALL BLOCKGS
      IF(NSOLVE.EQ.2)CALL BRTCC
      IF(NSOLVE.EQ.3)CALL BCGS
      RETURN
      END
```

```
*DECK GAUSSV
      SUBROUTINE GAUSSV
C  Computes the solution to Ax=b using point Gauss-Seidel iteration.
C  First all blocks except the diagonal block are moved to the RHS.
C  The off-diagonal terms of the diagonal block are also moved to
C  the RHS.  Then the solution is given by the diagonal terms of the
C  diagonal block.
*CALL COMMZ

C**********************
C   Initial guess     *
C**********************

      DO 25 I=1,NCT
       XI(I,1)=0.0D0
       XI(I,2)=0.0D0
       XI(I,3)=0.0D0
       XI(I,4)=0.0D0
   25 CONTINUE

C******************************
C   Gauss-Seidel iteration    *
C******************************

      NITER=0
      DO 19 ITER=1,NSI
       NITER=NITER+1
       DXMAX=0.D0
        DO 10 I=1,NCT
         I1=NUMEL(I,1)
         I2=NUMEL(I,2)
         I3=NUMEL(I,3)
         I4=NUMEL(I,4)
         I5=NUMEL(I,5)
         I6=NUMEL(I,6)
         I7=NUMEL(I,7)
         I8=NUMEL(I,8)
         I9=NUMEL(I,9)
         I10=NUMEL(I,10)
          X1OLD=XI(I1,1)
          X2OLD=XI(I1,2)
          X3OLD=XI(I1,3)
          X4OLD=XI(I1,4)
          XI(I1,1)=(B(I1,1)
     .                    -A(I1,1,1,2)*XI(I1,2)-A(I1,1,1,3)*XI(I1,3)
     .-A(I1,1,1,4)*XI(I1,4)-A(I1,2,1,1)*XI(I2,1)-A(I1,2,1,2)*XI(I2,2)
     .-A(I1,2,1,3)*XI(I2,3)-A(I1,2,1,4)*XI(I2,4)-A(I1,3,1,1)*XI(I3,1)
     .-A(I1,3,1,2)*XI(I3,2)-A(I1,3,1,3)*XI(I3,3)-A(I1,3,1,4)*XI(I3,4)
     .-A(I1,4,1,1)*XI(I4,1)-A(I1,4,1,2)*XI(I4,2)-A(I1,4,1,3)*XI(I4,3)
     .-A(I1,4,1,4)*XI(I4,4)-A(I1,5,1,1)*XI(I5,1)-A(I1,5,1,2)*XI(I5,2)
     .-A(I1,5,1,3)*XI(I5,3)-A(I1,5,1,4)*XI(I5,4)-A(I1,6,1,1)*XI(I6,1)
     .-A(I1,6,1,2)*XI(I6,2)-A(I1,6,1,3)*XI(I6,3)-A(I1,6,1,4)*XI(I6,4)
     .-A(I1,7,1,1)*XI(I7,1)-A(I1,7,1,2)*XI(I7,2)-A(I1,7,1,3)*XI(I7,3)
     .-A(I1,7,1,4)*XI(I7,4)-A(I1,8,1,1)*XI(I8,1)-A(I1,8,1,2)*XI(I8,2)
.-A(I1,8,1,3)*XI(I8,3)-A(I1,8,1,4)*XI(I8,4)-A(I1,9,1,1)*XI(I9,1)
.-A(I1,9,1,2)*XI(I9,2)-A(I1,9,1,3)*XI(I9,3)-A(I1,9,1,4)*XI(I9,4)
.-A(I1,10,1,1)*XI(I10,1)-A(I1,10,1,2)*XI(I10,2)
.-A(I1,10,1,3)*XI(I10,3)-A(I1,10,1,4)*XI(I10,4))/A(I1,1,1,1)
     XI(I1,2)=(B(I1,2)
.-A(I1,1,2,1)*XI(I1,1)                    -A(I1,1,2,3)*XI(I1,3)
.-A(I1,1,2,4)*XI(I1,4)-A(I1,2,2,1)*XI(I2,1)-A(I1,2,2,2)*XI(I2,2)
.-A(I1,2,2,3)*XI(I2,3)-A(I1,2,2,4)*XI(I2,4)-A(I1,3,2,1)*XI(I3,1)
.-A(I1,3,2,2)*XI(I3,2)-A(I1,3,2,3)*XI(I3,3)-A(I1,3,2,4)*XI(I3,4)
.-A(I1,4,2,1)*XI(I4,1)-A(I1,4,2,2)*XI(I4,2)-A(I1,4,2,3)*XI(I4,3)
.-A(I1,4,2,4)*XI(I4,4)-A(I1,5,2,1)*XI(I5,1)-A(I1,5,2,2)*XI(I5,2)
.-A(I1,5,2,3)*XI(I5,3)-A(I1,5,2,4)*XI(I5,4)-A(I1,6,2,1)*XI(I6,1)
.-A(I1,6,2,2)*XI(I6,2)-A(I1,6,2,3)*XI(I6,3)-A(I1,6,2,4)*XI(I6,4)
.-A(I1,7,2,1)*XI(I7,1)-A(I1,7,2,2)*XI(I7,2)-A(I1,7,2,3)*XI(I7,3)
.-A(I1,7,2,4)*XI(I7,4)-A(I1,8,2,1)*XI(I8,1)-A(I1,8,2,2)*XI(I8,2)
.-A(I1,8,2,3)*XI(I8,3)-A(I1,8,2,4)*XI(I8,4)-A(I1,9,2,1)*XI(I9,1)
.-A(I1,9,2,2)*XI(I9,2)-A(I1,9,2,3)*XI(I9,3)-A(I1,9,2,4)*XI(I9,4)
.-A(I1,10,2,1)*XI(I10,1)-A(I1,10,2,2)*XI(I10,2)
.-A(I1,10,2,3)*XI(I10,3)-A(I1,10,2,4)*XI(I10,4))/A(I1,1,2,2)
     XI(I1,3)=(B(I1,3)
.-A(I1,1,3,1)*XI(I1,1)-A(I1,1,3,2)*XI(I1,2)
.-A(I1,1,3,4)*XI(I1,4)-A(I1,2,3,1)*XI(I2,1)-A(I1,2,3,2)*XI(I2,2)
.-A(I1,2,3,3)*XI(I2,3)-A(I1,2,3,4)*XI(I2,4)-A(I1,3,3,1)*XI(I3,1)
.-A(I1,3,3,2)*XI(I3,2)-A(I1,3,3,3)*XI(I3,3)-A(I1,3,3,4)*XI(I3,4)
.-A(I1,4,3,1)*XI(I4,1)-A(I1,4,3,2)*XI(I4,2)-A(I1,4,3,3)*XI(I4,3)
.-A(I1,4,3,4)*XI(I4,4)-A(I1,5,3,1)*XI(I5,1)-A(I1,5,3,2)*XI(I5,2)
.-A(I1,5,3,3)*XI(I5,3)-A(I1,5,3,4)*XI(I5,4)-A(I1,6,3,1)*XI(I6,1)
.-A(I1,6,3,2)*XI(I6,2)-A(I1,6,3,3)*XI(I6,3)-A(I1,6,3,4)*XI(I6,4)
.-A(I1,7,3,1)*XI(I7,1)-A(I1,7,3,2)*XI(I7,2)-A(I1,7,3,3)*XI(I7,3)
.-A(I1,7,3,4)*XI(I7,4)-A(I1,8,3,1)*XI(I8,1)-A(I1,8,3,2)*XI(I8,2)
.-A(I1,8,3,3)*XI(I8,3)-A(I1,8,3,4)*XI(I8,4)-A(I1,9,3,1)*XI(I9,1)
.-A(I1,9,3,2)*XI(I9,2)-A(I1,9,3,3)*XI(I9,3)-A(I1,9,3,4)*XI(I9,4)
.-A(I1,10,3,1)*XI(I10,1)-A(I1,10,3,2)*XI(I10,2)
.-A(I1,10,3,3)*XI(I10,3)-A(I1,10,3,4)*XI(I10,4))/A(I1,1,3,3)
     XI(I1,4)=(B(I1,4)
.-A(I1,1,4,1)*XI(I1,1)-A(I1,1,4,2)*XI(I1,2)-A(I1,1,4,3)*XI(I1,3)
.                      -A(I1,2,4,1)*XI(I2,1)-A(I1,2,4,2)*XI(I2,2)
.-A(I1,2,4,3)*XI(I2,3)-A(I1,2,4,4)*XI(I2,4)-A(I1,3,4,1)*XI(I3,1)
.-A(I1,3,4,2)*XI(I3,2)-A(I1,3,4,3)*XI(I3,3)-A(I1,3,4,4)*XI(I3,4)
.-A(I1,4,4,1)*XI(I4,1)-A(I1,4,4,2)*XI(I4,2)-A(I1,4,4,3)*XI(I4,3)
.-A(I1,4,4,4)*XI(I4,4)-A(I1,5,4,1)*XI(I5,1)-A(I1,5,4,2)*XI(I5,2)
.-A(I1,5,4,3)*XI(I5,3)-A(I1,5,4,4)*XI(I5,4)-A(I1,6,4,1)*XI(I6,1)
.-A(I1,6,4,2)*XI(I6,2)-A(I1,6,4,3)*XI(I6,3)-A(I1,6,4,4)*XI(I6,4)
.-A(I1,7,4,1)*XI(I7,1)-A(I1,7,4,2)*XI(I7,2)-A(I1,7,4,3)*XI(I7,3)
.-A(I1,7,4,4)*XI(I7,4)-A(I1,8,4,1)*XI(I8,1)-A(I1,8,4,2)*XI(I8,2)
.-A(I1,8,4,3)*XI(I8,3)-A(I1,8,4,4)*XI(I8,4)-A(I1,9,4,1)*XI(I9,1)
.-A(I1,9,4,2)*XI(I9,2)-A(I1,9,4,3)*XI(I9,3)-A(I1,9,4,4)*XI(I9,4)
.-A(I1,10,4,1)*XI(I10,1)-A(I1,10,4,2)*XI(I10,2)
.-A(I1,10,4,3)*XI(I10,3)-A(I1,10,4,4)*XI(I10,4))/A(I1,1,4,4)
    DX1=ABS(XI(I1,1)-X1OLD)
```

```
        DX2=ABS(XI(I1,2)-X2OLD)
        DX3=ABS(XI(I1,3)-X3OLD)
        DX4=ABS(XI(I1,4)-X4OLD)
       DXMAX=AMAX1(DXMAX,DX1,DX2,DX3,DX4)
   10 CONTINUE
      IF(DXMAX.LE.1.E-6)GO TO 20
   19 CONTINUE
   20 PRINT*,'Number of Gauss-Seidel iterations =',NITER
      RETURN
      END



*DECK BLOCKGS
      SUBROUTINE BLOCKGS
C  Computes the solution to Ax=b using block Gauss-Seidel iteration.
C  First all blocks except the diagonal block are moved to the RHS.
C  Then the remaining diagonal block is reduced by LU decomposition.
*CALL COMMZ
C*********************
C   Initial guess    *
C*********************
      DO 1 I=1,NCT
       XI(I,1)=0.0D0
       XI(I,2)=0.0D0
       XI(I,3)=0.0D0
       XI(I,4)=0.0D0
    1 CONTINUE
C************************************
C   Block Gauss-Seidel iteration    *
C************************************
      NITER=0
      DO 2 ITER=1,NSI
       NITER=NITER+1
       DXMAX=0.D0
        DO 3 I=1,NCT
         I1=NUMEL(I,1)
         I2=NUMEL(I,2)
         I3=NUMEL(I,3)
         I4=NUMEL(I,4)
         I5=NUMEL(I,5)
         I6=NUMEL(I,6)
         I7=NUMEL(I,7)
         I8=NUMEL(I,8)
         I9=NUMEL(I,9)
         I10=NUMEL(I,10)
          X1OLD=XI(I1,1)
          X2OLD=XI(I1,2)
          X3OLD=XI(I1,3)
          X4OLD=XI(I1,4)
```

```
C*********************************************
C   Compute new RHS (Move all blocks to rhs     *
C   except the diagonal block).                 *
C*********************************************
      XI(I1,1)=B(I1,1)
     .-A(I1,2,1,1)*XI(I2,1)-A(I1,2,1,2)*XI(I2,2)
     .-A(I1,2,1,3)*XI(I2,3)-A(I1,2,1,4)*XI(I2,4)-A(I1,3,1,1)*XI(I3,1)
     .-A(I1,3,1,2)*XI(I3,2)-A(I1,3,1,3)*XI(I3,3)-A(I1,3,1,4)*XI(I3,4)
     .-A(I1,4,1,1)*XI(I4,1)-A(I1,4,1,2)*XI(I4,2)-A(I1,4,1,3)*XI(I4,3)
     .-A(I1,4,1,4)*XI(I4,4)-A(I1,5,1,1)*XI(I5,1)-A(I1,5,1,2)*XI(I5,2)
     .-A(I1,5,1,3)*XI(I5,3)-A(I1,5,1,4)*XI(I5,4)-A(I1,6,1,1)*XI(I6,1)
     .-A(I1,6,1,2)*XI(I6,2)-A(I1,6,1,3)*XI(I6,3)-A(I1,6,1,4)*XI(I6,4)
     .-A(I1,7,1,1)*XI(I7,1)-A(I1,7,1,2)*XI(I7,2)-A(I1,7,1,3)*XI(I7,3)
     .-A(I1,7,1,4)*XI(I7,4)-A(I1,8,1,1)*XI(I8,1)-A(I1,8,1,2)*XI(I8,2)
     .-A(I1,8,1,3)*XI(I8,3)-A(I1,8,1,4)*XI(I8,4)-A(I1,9,1,1)*XI(I9,1)
     .-A(I1,9,1,2)*XI(I9,2)-A(I1,9,1,3)*XI(I9,3)-A(I1,9,1,4)*XI(I9,4)
     .-A(I1,10,1,1)*XI(I10,1)-A(I1,10,1,2)*XI(I10,2)
     .-A(I1,10,1,3)*XI(I10,3)-A(I1,10,1,4)*XI(I10,4)

      XI(I1,2)=B(I1,2)
     .-A(I1,2,2,1)*XI(I2,1)-A(I1,2,2,2)*XI(I2,2)
     .-A(I1,2,2,3)*XI(I2,3)-A(I1,2,2,4)*XI(I2,4)-A(I1,3,2,1)*XI(I3,1)
     .-A(I1,3,2,2)*XI(I3,2)-A(I1,3,2,3)*XI(I3,3)-A(I1,3,2,4)*XI(I3,4)
     .-A(I1,4,2,1)*XI(I4,1)-A(I1,4,2,2)*XI(I4,2)-A(I1,4,2,3)*XI(I4,3)
     .-A(I1,4,2,4)*XI(I4,4)-A(I1,5,2,1)*XI(I5,1)-A(I1,5,2,2)*XI(I5,2)
     .-A(I1,5,2,3)*XI(I5,3)-A(I1,5,2,4)*XI(I5,4)-A(I1,6,2,1)*XI(I6,1)
     .-A(I1,6,2,2)*XI(I6,2)-A(I1,6,2,3)*XI(I6,3)-A(I1,6,2,4)*XI(I6,4)
     .-A(I1,7,2,1)*XI(I7,1)-A(I1,7,2,2)*XI(I7,2)-A(I1,7,2,3)*XI(I7,3)
     .-A(I1,7,2,4)*XI(I7,4)-A(I1,8,2,1)*XI(I8,1)-A(I1,8,2,2)*XI(I8,2)
     .-A(I1,8,2,3)*XI(I8,3)-A(I1,8,2,4)*XI(I8,4)-A(I1,9,2,1)*XI(I9,1)
     .-A(I1,9,2,2)*XI(I9,2)-A(I1,9,2,3)*XI(I9,3)-A(I1,9,2,4)*XI(I9,4)
     .-A(I1,10,2,1)*XI(I10,1)-A(I1,10,2,2)*XI(I10,2)
     .-A(I1,10,2,3)*XI(I10,3)-A(I1,10,2,4)*XI(I10,4)

      XI(I1,3)=B(I1,3)
     .-A(I1,2,3,1)*XI(I2,1)-A(I1,2,3,2)*XI(I2,2)
     .-A(I1,2,3,3)*XI(I2,3)-A(I1,2,3,4)*XI(I2,4)-A(I1,3,3,1)*XI(I3,1)
     .-A(I1,3,3,2)*XI(I3,2)-A(I1,3,3,3)*XI(I3,3)-A(I1,3,3,4)*XI(I3,4)
     .-A(I1,4,3,1)*XI(I4,1)-A(I1,4,3,2)*XI(I4,2)-A(I1,4,3,3)*XI(I4,3)
     .-A(I1,4,3,4)*XI(I4,4)-A(I1,5,3,1)*XI(I5,1)-A(I1,5,3,2)*XI(I5,2)
     .-A(I1,5,3,3)*XI(I5,3)-A(I1,5,3,4)*XI(I5,4)-A(I1,6,3,1)*XI(I6,1)
     .-A(I1,6,3,2)*XI(I6,2)-A(I1,6,3,3)*XI(I6,3)-A(I1,6,3,4)*XI(I6,4)
     .-A(I1,7,3,1)*XI(I7,1)-A(I1,7,3,2)*XI(I7,2)-A(I1,7,3,3)*XI(I7,3)
     .-A(I1,7,3,4)*XI(I7,4)-A(I1,8,3,1)*XI(I8,1)-A(I1,8,3,2)*XI(I8,2)
     .-A(I1,8,3,3)*XI(I8,3)-A(I1,8,3,4)*XI(I8,4)-A(I1,9,3,1)*XI(I9,1)
     .-A(I1,9,3,2)*XI(I9,2)-A(I1,9,3,3)*XI(I9,3)-A(I1,9,3,4)*XI(I9,4)
     .-A(I1,10,3,1)*XI(I10,1)-A(I1,10,3,2)*XI(I10,2)
     .-A(I1,10,3,3)*XI(I10,3)-A(I1,10,3,4)*XI(I10,4)

      XI(I1,4)=B(I1,4)
     .-A(I1,2,4,1)*XI(I2,1)-A(I1,2,4,2)*XI(I2,2)
     .-A(I1,2,4,3)*XI(I2,3)-A(I1,2,4,4)*XI(I2,4)-A(I1,3,4,1)*XI(I3,1)
     .-A(I1,3,4,2)*XI(I3,2)-A(I1,3,4,3)*XI(I3,3)-A(I1,3,4,4)*XI(I3,4)
     .-A(I1,4,4,1)*XI(I4,1)-A(I1,4,4,2)*XI(I4,2)-A(I1,4,4,3)*XI(I4,3)
     .-A(I1,4,4,4)*XI(I4,4)-A(I1,5,4,1)*XI(I5,1)-A(I1,5,4,2)*XI(I5,2)
     .-A(I1,5,4,3)*XI(I5,3)-A(I1,5,4,4)*XI(I5,4)-A(I1,6,4,1)*XI(I6,1)
     .-A(I1,6,4,2)*XI(I6,2)-A(I1,6,4,3)*XI(I6,3)-A(I1,6,4,4)*XI(I6,4)
     .-A(I1,7,4,1)*XI(I7,1)-A(I1,7,4,2)*XI(I7,2)-A(I1,7,4,3)*XI(I7,3)
     .-A(I1,7,4,4)*XI(I7,4)-A(I1,8,4,1)*XI(I8,1)-A(I1,8,4,2)*XI(I8,2)
     .-A(I1,8,4,3)*XI(I8,3)-A(I1,8,4,4)*XI(I8,4)-A(I1,9,4,1)*XI(I9,1)
     .-A(I1,9,4,2)*XI(I9,2)-A(I1,9,4,3)*XI(I9,3)-A(I1,9,4,4)*XI(I9,4)
     .-A(I1,10,4,1)*XI(I10,1)-A(I1,10,4,2)*XI(I10,2)
     .-A(I1,10,4,3)*XI(I10,3)-A(I1,10,4,4)*XI(I10,4)
C****************************************
C* Perform LU decomposition and solve.  *
C****************************************

      AL11=A(I1,1,1,1)
      AL21=A(I1,1,2,1)
      AL31=A(I1,1,3,1)
      AL41=A(I1,1,4,1)

      AU11=1.
      AU12=A(I1,1,1,2)/AL11
      AU13=A(I1,1,1,3)/AL11
      AU14=A(I1,1,1,4)/AL11

      AL22=A(I1,1,2,2)-AL21*AU12
      AL32=A(I1,1,3,2)-AL31*AU12
      AL42=A(I1,1,4,2)-AL41*AU12

      AU22=1.
      AU23=(A(I1,1,2,3)-AL21*AU13)/AL22
      AU24=(A(I1,1,2,4)-AL21*AU14)/AL22

      AL33=A(I1,1,3,3)-AL31*AU13-AL32*AU23
      AL43=A(I1,1,4,3)-AL41*AU13-AL42*AU23

      AU33=1.
      AU34=(A(I1,1,3,4)-AL31*AU14-AL32*AU24)/AL33

      AL44=A(I1,1,4,4)-AL41*AU14-AL42*AU24-AL43*AU34

      AU44=1.
C**************************
C* Forward substitution   *
C**************************

      XI(I1,1)=XI(I1,1)/AL11
      XI(I1,2)=(XI(I1,2)-AL21*XI(I1,1))/AL22
      XI(I1,3)=(XI(I1,3)-AL31*XI(I1,1)-AL32*XI(I1,2))/AL33
      XI(I1,4)=(XI(I1,4)-AL41*XI(I1,1)-AL42*XI(I1,2)
     .                  -AL43*XI(I1,3))/AL44

C***************************
C* Backward substitution   *
C***************************

      XI(I1,4)=XI(I1,4)
      XI(I1,3)=XI(I1,3)-AU34*XI(I1,4)
      XI(I1,2)=XI(I1,2)-AU23*XI(I1,3)-AU24*XI(I1,4)
      XI(I1,1)=XI(I1,1)-AU12*XI(I1,2)-AU13*XI(I1,3)-AU14*XI(I1,4)

         DX1=ABS(XI(I1,1)-X1OLD)
         DX2=ABS(XI(I1,2)-X2OLD)
         DX3=ABS(XI(I1,3)-X3OLD)
         DX4=ABS(XI(I1,4)-X4OLD)
        DXMAX=AMAX1(DXMAX,DX1,DX2,DX3,DX4)
    3 CONTINUE
      IF(DXMAX.LE.1.E-6)GO TO 4
    2 CONTINUE
    4 PRINT*,'Number of Gauss-Seidel iterations =',NITER
      RETURN
      END




*DECK BCGS
      SUBROUTINE BCGS
C  Computes the solution to Ax=b using block Gauss-Seidel iteration.
C  First all blocks except the diagonal block are moved to the RHS.
C  Then the remaining diagonal block is reduced by LU decomposition.
C  Cell coloring is used to vectorize the solver over its level 1
C  neighbors.
*CALL COMMZ
C***********************
C   Initial guess     *
C***********************

      DO 1 I=1,NCT
       XI(I,1)=0.0D0
       XI(I,2)=0.0D0
       XI(I,3)=0.0D0
       XI(I,4)=0.0D0
    1 CONTINUE

C************************************
C   Block Gauss-Seidel iteration    *
C************************************

      NITER=0
      DO 2 ITER=1,NSI
       NITER=NITER+1
       DXMAX=0.D0
C   Cell coloring loop; four colors.
       DO 3 K=1,4
       NCLOOP=NRGBY(K)
C   Vectorize the next loop for each color
CDIR@ IVDEP
        DO 3 N=1,NCLOOP
         I=NCOLOR(K,N)
         I1=NUMEL(I,1)
         I2=NUMEL(I,2)
         I3=NUMEL(I,3)
```

```
          I4=NUMEL(I,4)
          I5=NUMEL(I,5)
          I6=NUMEL(I,6)
          I7=NUMEL(I,7)
          I8=NUMEL(I,8)
          I9=NUMEL(I,9)
          I10=NUMEL(I,10)
           X1OLD=XI(I1,1)
           X2OLD=XI(I1,2)
           X3OLD=XI(I1,3)
           X4OLD=XI(I1,4)
C***********************************************
C   Compute new RHS (Move all blocks to rhs    *
C   except the diagonal block).                *
C***********************************************
           XI(I1,1)=B(I1,1)
     .-A(I1,2,1,1)*XI(I2,1)-A(I1,2,1,2)*XI(I2,2)
     .-A(I1,2,1,3)*XI(I2,3)-A(I1,2,1,4)*XI(I2,4)-A(I1,3,1,1)*XI(I3,1)
     .-A(I1,3,1,2)*XI(I3,2)-A(I1,3,1,3)*XI(I3,3)-A(I1,3,1,4)*XI(I3,4)
     .-A(I1,4,1,1)*XI(I4,1)-A(I1,4,1,2)*XI(I4,2)-A(I1,4,1,3)*XI(I4,3)
     .-A(I1,4,1,4)*XI(I4,4)-A(I1,5,1,1)*XI(I5,1)-A(I1,5,1,2)*XI(I5,2)
     .-A(I1,5,1,3)*XI(I5,3)-A(I1,5,1,4)*XI(I5,4)-A(I1,6,1,1)*XI(I6,1)
     .-A(I1,6,1,2)*XI(I6,2)-A(I1,6,1,3)*XI(I6,3)-A(I1,6,1,4)*XI(I6,4)
     .-A(I1,7,1,1)*XI(I7,1)-A(I1,7,1,2)*XI(I7,2)-A(I1,7,1,3)*XI(I7,3)
     .-A(I1,7,1,4)*XI(I7,4)-A(I1,8,1,1)*XI(I8,1)-A(I1,8,1,2)*XI(I8,2)
     .-A(I1,8,1,3)*XI(I8,3)-A(I1,8,1,4)*XI(I8,4)-A(I1,9,1,1)*XI(I9,1)
     .-A(I1,9,1,2)*XI(I9,2)-A(I1,9,1,3)*XI(I9,3)-A(I1,9,1,4)*XI(I9,4)
     .-A(I1,10,1,1)*XI(I10,1)-A(I1,10,1,2)*XI(I10,2)
     .-A(I1,10,1,3)*XI(I10,3)-A(I1,10,1,4)*XI(I10,4)

           XI(I1,2)=B(I1,2)
     .-A(I1,2,2,1)*XI(I2,1)-A(I1,2,2,2)*XI(I2,2)
     .-A(I1,2,2,3)*XI(I2,3)-A(I1,2,2,4)*XI(I2,4)-A(I1,3,2,1)*XI(I3,1)
     .-A(I1,3,2,2)*XI(I3,2)-A(I1,3,2,3)*XI(I3,3)-A(I1,3,2,4)*XI(I3,4)
     .-A(I1,4,2,1)*XI(I4,1)-A(I1,4,2,2)*XI(I4,2)-A(I1,4,2,3)*XI(I4,3)
     .-A(I1,4,2,4)*XI(I4,4)-A(I1,5,2,1)*XI(I5,1)-A(I1,5,2,2)*XI(I5,2)
     .-A(I1,5,2,3)*XI(I5,3)-A(I1,5,2,4)*XI(I5,4)-A(I1,6,2,1)*XI(I6,1)
     .-A(I1,6,2,2)*XI(I6,2)-A(I1,6,2,3)*XI(I6,3)-A(I1,6,2,4)*XI(I6,4)
     .-A(I1,7,2,1)*XI(I7,1)-A(I1,7,2,2)*XI(I7,2)-A(I1,7,2,3)*XI(I7,3)
     .-A(I1,7,2,4)*XI(I7,4)-A(I1,8,2,1)*XI(I8,1)-A(I1,8,2,2)*XI(I8,2)
     .-A(I1,8,2,3)*XI(I8,3)-A(I1,8,2,4)*XI(I8,4)-A(I1,9,2,1)*XI(I9,1)
     .-A(I1,9,2,2)*XI(I9,2)-A(I1,9,2,3)*XI(I9,3)-A(I1,9,2,4)*XI(I9,4)
     .-A(I1,10,2,1)*XI(I10,1)-A(I1,10,2,2)*XI(I10,2)
     .-A(I1,10,2,3)*XI(I10,3)-A(I1,10,2,4)*XI(I10,4)

           XI(I1,3)=B(I1,3)
     .-A(I1,2,3,1)*XI(I2,1)-A(I1,2,3,2)*XI(I2,2)
     .-A(I1,2,3,3)*XI(I2,3)-A(I1,2,3,4)*XI(I2,4)-A(I1,3,3,1)*XI(I3,1)
     .-A(I1,3,3,2)*XI(I3,2)-A(I1,3,3,3)*XI(I3,3)-A(I1,3,3,4)*XI(I3,4)
     .-A(I1,4,3,1)*XI(I4,1)-A(I1,4,3,2)*XI(I4,2)-A(I1,4,3,3)*XI(I4,3)
     .-A(I1,4,3,4)*XI(I4,4)-A(I1,5,3,1)*XI(I5,1)-A(I1,5,3,2)*XI(I5,2)
     .-A(I1,5,3,3)*XI(I5,3)-A(I1,5,3,4)*XI(I5,4)-A(I1,6,3,1)*XI(I6,1)
     .-A(I1,6,3,2)*XI(I6,2)-A(I1,6,3,3)*XI(I6,3)-A(I1,6,3,4)*XI(I6,4)
     .-A(I1,7,3,1)*XI(I7,1)-A(I1,7,3,2)*XI(I7,2)-A(I1,7,3,3)*XI(I7,3)
     .-A(I1,7,3,4)*XI(I7,4)-A(I1,8,3,1)*XI(I8,1)-A(I1,8,3,2)*XI(I8,2)
     .-A(I1,8,3,3)*XI(I8,3)-A(I1,8,3,4)*XI(I8,4)-A(I1,9,3,1)*XI(I9,1)
     .-A(I1,9,3,2)*XI(I9,2)-A(I1,9,3,3)*XI(I9,3)-A(I1,9,3,4)*XI(I9,4)
     .-A(I1,10,3,1)*XI(I10,1)-A(I1,10,3,2)*XI(I10,2)
     .-A(I1,10,3,3)*XI(I10,3)-A(I1,10,3,4)*XI(I10,4)

         XI(I1,4)=B(I1,4)
     .-A(I1,2,4,1)*XI(I2,1)-A(I1,2,4,2)*XI(I2,2)
     .-A(I1,2,4,3)*XI(I2,3)-A(I1,2,4,4)*XI(I2,4)-A(I1,3,4,1)*XI(I3,1)
     .-A(I1,3,4,2)*XI(I3,2)-A(I1,3,4,3)*XI(I3,3)-A(I1,3,4,4)*XI(I3,4)
     .-A(I1,4,4,1)*XI(I4,1)-A(I1,4,4,2)*XI(I4,2)-A(I1,4,4,3)*XI(I4,3)
     .-A(I1,4,4,4)*XI(I4,4)-A(I1,5,4,1)*XI(I5,1)-A(I1,5,4,2)*XI(I5,2)
     .-A(I1,5,4,3)*XI(I5,3)-A(I1,5,4,4)*XI(I5,4)-A(I1,6,4,1)*XI(I6,1)
     .-A(I1,6,4,2)*XI(I6,2)-A(I1,6,4,3)*XI(I6,3)-A(I1,6,4,4)*XI(I6,4)
     .-A(I1,7,4,1)*XI(I7,1)-A(I1,7,4,2)*XI(I7,2)-A(I1,7,4,3)*XI(I7,3)
     .-A(I1,7,4,4)*XI(I7,4)-A(I1,8,4,1)*XI(I8,1)-A(I1,8,4,2)*XI(I8,2)
     .-A(I1,8,4,3)*XI(I8,3)-A(I1,8,4,4)*XI(I8,4)-A(I1,9,4,1)*XI(I9,1)
     .-A(I1,9,4,2)*XI(I9,2)-A(I1,9,4,3)*XI(I9,3)-A(I1,9,4,4)*XI(I9,4)
     .-A(I1,10,4,1)*XI(I10,1)-A(I1,10,4,2)*XI(I10,2)
     .-A(I1,10,4,3)*XI(I10,3)-A(I1,10,4,4)*XI(I10,4)
C****************************************
C* Perform LU decomposition and solve.  *
C****************************************

      AL11=A(I1,1,1,1)
      AL21=A(I1,1,2,1)
      AL31=A(I1,1,3,1)
      AL41=A(I1,1,4,1)

      AU11=1.
      AU12=A(I1,1,1,2)/AL11
      AU13=A(I1,1,1,3)/AL11
      AU14=A(I1,1,1,4)/AL11

      AL22=A(I1,1,2,2)-AL21*AU12
      AL32=A(I1,1,3,2)-AL31*AU12
      AL42=A(I1,1,4,2)-AL41*AU12

      AU22=1.
      AU23=(A(I1,1,2,3)-AL21*AU13)/AL22
      AU24=(A(I1,1,2,4)-AL21*AU14)/AL22

      AL33=A(I1,1,3,3)-AL31*AU13-AL32*AU23
      AL43=A(I1,1,4,3)-AL41*AU13-AL42*AU23

      AU33=1.
      AU34=(A(I1,1,3,4)-AL31*AU14-AL32*AU24)/AL33

      AL44=A(I1,1,4,4)-AL41*AU14-AL42*AU24-AL43*AU34

      AU44=1.

C*************************
C* Forward substitution  *
C*************************
```

```
      XI(I1,1)=XI(I1,1)/AL11
      XI(I1,2)=(XI(I1,2)-AL21*XI(I1,1))/AL22
      XI(I1,3)=(XI(I1,3)-AL31*XI(I1,1)-AL32*XI(I1,2))/AL33
      XI(I1,4)=(XI(I1,4)-AL41*XI(I1,1)-AL42*XI(I1,2)
     .                 -AL43*XI(I1,3))/AL44

C***************************
C* Backward substitution   *
C***************************

      XI(I1,4)=XI(I1,4)
      XI(I1,3)=XI(I1,3)-AU34*XI(I1,4)
      XI(I1,2)=XI(I1,2)-AU23*XI(I1,3)-AU24*XI(I1,4)
      XI(I1,1)=XI(I1,1)-AU12*XI(I1,2)-AU13*XI(I1,3)-AU14*XI(I1,4)

         DX1=ABS(XI(I1,1)-X1OLD)
         DX2=ABS(XI(I1,2)-X2OLD)
         DX3=ABS(XI(I1,3)-X3OLD)
         DX4=ABS(XI(I1,4)-X4OLD)
        DXMAX=AMAX1(DXMAX,DX1,DX2,DX3,DX4)
    3 CONTINUE

    2 CONTINUE
      PRINT*,'Number of Gauss-Seidel iterations =',NITER

      RETURN
      END



*DECK BMCO
      SUBROUTINE BMCO
*CALL COMMZ
c    Block Matrix Column Ordering
c      sorts the block matrix in column order

      ICC=0
      DO 1 ICG=1,NCT
       IP=0
       DO 2 I=1,NCT
        DO 2 J=1,NCPL
         IF(NUMEL(I,J).EQ.ICG) THEN
          ICC=ICC+1
          IP=IP+1
          IB(ICG,IP)=I
          JB(ICG,IP)=J
         ENDIF
    2  CONTINUE
       NBC(ICG)=IP
    1 CONTINUE

      NEQNS=NCT*NBLOCK

      RETURN
      END
```

```
*DECK BRTCC
      SUBROUTINE BRTCC
*CALL COMMZ
c    Block row to compressed column
c     converts a block row matrix to compressed column format

c    Compute column pointer: # of 1st entry of each column

      IPP=1
      IC=1
      COLPTR(1)=1
      DO 1 ICG=1,NCT
       IP=NBC(ICG)
        DO 1 L=1,NBLOCK
         IC=IC+1
         IPP=IPP+IP*NBLOCK
         COLPTR(IC)=IPP
    1 CONTINUE

c    Compute row index: row # of each entry of each column
c                       and the value of that entry

      IR=0
      DO 2 ICG=1,NCT
       IP=NBC(ICG)
       DO 2 L=1,NBLOCK
        DO 2 II=1,IP
         I=IB(ICG,II)
         J=JB(ICG,II)
         DO 2 K=1,NBLOCK
          IR=IR+1
          ROWIND(IR)=(I-1)*NBLOCK+K
          AC(IR)=A(I,J,K,L)
    2 CONTINUE

      N=0
      DO 3 I=1,NCT
       DO 3 K=1,NBLOCK
        N=N+1
        RHS(N)=B(I,K)
    3 CONTINUE

      CALL SPARSE

      RETURN
      END


*DECK SPARSE
      SUBROUTINE SPARSE
      COMMON/WORD/METHOD
      CHARACTER*3 METHOD
*CALL COMMZ
      INTEGER IWORK(LIWORK), IPARAM(40)
      REAL RPARAM(30), WORK(LWORK)
c....Let the initial guess for x be random numbers between 0 and 1
      DO 20 I = 1, NEQNS
        XR(I) = RANF()
   20 CONTINUE
c.....Set default parameter values
      CALL DFAULTS ( IPARAM, RPARAM )
c.....Solve a nonsymmetric matrix IPARAM(1)=0
c.....Select left preconditioning
      IPARAM(1) = 0
      IPARAM(3) = NSI
      IPARAM(7) = 0
      IPARAM(9) = 1
      IPARAM(10) = NPRET
      IPARAM(16) = KBV
      IPARAM(17) = KBV
      RPARAM(1) = 1.E-2
c.....Call SITRSOL to solve the problem
      ipath = 2
      CALL SITRSOL ( method, ipath, neqns, neqns, xr, rhs, colptr,
     .               rowind, ac, liwork, iwork, lwork, work,
     .               iparam, rparam, ierr )
      KC=0
      DO 35 I=1,NCT
       DO 35 K=1,4
       KC=KC+1
       XI(I,K)=XR(KC)
   35 CONTINUE
      PRINT*,'Number of Sparse solver iterations=',IPARAM(4)
      RETURN
      END


*DECK DAMPING
      SUBROUTINE DAMPING
*CALL COMMZ
      IF(NDAMP.EQ.0)RETURN
      IF(NDAMP.EQ.1)CALL DAMP24
      IF(NDAMP.EQ.2)CALL DAMP4
      IF(NDAMP.EQ.3)CALL DAMP4P
      RETURN
      END


*DECK DAMP24
```

```
      SUBROUTINE DAMP24
*CALL COMMZ
      DIMENSION D1F(NFPAR,4),D2F(NCPAR,4),D2P(NCPAR,2)
C  Zero out storage of 2nd difference of conserved variables and
C  storage for pressure switch
      DO 1 I=1,NCT
       D2F(I,NCO)=0.0
       D2F(I,NXM)=0.0
       D2F(I,NYM)=0.0
       D2F(I,NEN)=0.0
       D2P(I,1)=0.0
       D2P(I,2)=0.0
    1 CONTINUE
C  Compute 1st difference on each edge, and sum to get 2nd
C  difference in cell
      DO 2 I=1,NFT
       N1=NFACE(1,I)
       N2=NFACE(2,I)
       IF(N1.GT.0.AND.N2.GT.0)THEN
        U1=U(N1)
        V1=V(N1)
        P1=P(N1)
        T1=T(N1)
        U2=U(N2)
        V2=V(N2)
        P2=P(N2)
        T2=T(N2)
       ELSE
C    Solid Wall
       IF(N1.EQ.0.OR.N2.EQ.0)THEN
        N1=N1+N2
        U1=U(N1)
        V1=V(N1)
        P1=P(N1)
        T1=T(N1)
        U2=-U1
        V2=-V1
        P2=P1
        T2=T1
       ELSE
C    Inlet
       IF(N1.EQ.-1.OR.N2.EQ.-1)THEN
        N1=N1+N2+1
        N2=N1
        U1=U(N1)
        V1=V(N1)
        P1=P(N1)
        T1=T(N1)
C      Subsonic inlet
        IF(NIBC.EQ.0)THEN
         U2=2.*UI(N1)-U1
         V2=2.*VI(N1)-V1
         P2=P(N2)
         T2=2.*TI(N1)-T1
        ELSE
```

```
C       Supersonic inlet
          IF(NIBC.EQ.2)THEN
            U2=2.*UI(N1)-U1
            V2=2.*VI(N1)-V1
            P2=2.*PI(N1)-P1
            T2=2.*TI(N1)-T1
          ENDIF
          ENDIF
        ELSE
C     Exit
        IF(N1.EQ.-2.OR.N2.EQ.-2)THEN
          N1=N1+N2+2
          N2=N1
          U1=U(N1)
          V1=V(N1)
          P1=P(N1)
          T1=T(N1)
C       Subsonic exit
          IF(NEBC.EQ.1)THEN
            U2=U(N1)
            V2=V(N1)
            P2=2.*PEXIT-P1
            T2=T(N1)
          ELSE
C       Supersonic exit
          IF(NEBC.EQ.2)THEN
            U2=U(N1)
            V2=V(N1)
            P2=P(N1)
            T2=T(N1)
          ENDIF
          ENDIF
        ENDIF
        ENDIF
        ENDIF
        ENDIF

C   Compute conserved variables on either side of edge I

        RHO1 =P1/T1
        RHOU1=RHO1*U1
        RHOV1=RHO1*V1
        EN1  =P1*((CP-R)+.5*(U1**2+V1**2)/T1)
        RHO2 =P2/T2
        RHOU2=RHO2*U2
        RHOV2=RHO2*V2
        EN2  =P2*((CP-R)+.5*(U2**2+V2**2)/T2)

C   Compute 1st difference on edge I and store in D1F for later use.

        DRHO =RHO1-RHO2
        DRHOU=RHOU1-RHOU2
        DRHOV=RHOV1-RHOV2
        DEN  =EN1-EN2

        D1F(I,NCO)=DRHO
        D1F(I,NXM)=DRHOU
        D1F(I,NYM)=DRHOV
        D1F(I,NEN)=DEN

C   2nd difference in cell(used later in computing 3rd difference)
```

```
        D2F(N1,NCO)=D2F(N1,NCO) - DRHO
        D2F(N1,NXM)=D2F(N1,NXM) - DRHOU
        D2F(N1,NYM)=D2F(N1,NYM) - DRHOV
        D2F(N1,NEN)=D2F(N1,NEN) - DEN
        D2F(N2,NCO)=D2F(N2,NCO) + DRHO
        D2F(N2,NXM)=D2F(N2,NXM) + DRHOU
        D2F(N2,NYM)=D2F(N2,NYM) + DRHOV
        D2F(N2,NEN)=D2F(N2,NEN) + DEN
C  1st difference in pressure and sum of pressure
        DP=P1-P2
        SP=P1+P2
C  2nd difference of pressure and sum of pressure for switch in
C  each cell
        D2P(N1,1)=D2P(N1,1) - DP
        D2P(N2,1)=D2P(N2,1) + DP
        D2P(N1,2)=D2P(N1,2) + SP
        D2P(N2,2)=D2P(N2,2) + SP
    2 CONTINUE
C  Compute switch based on pressure
       DO 3 I=1,NCT
        D2P(I,1)=ABS(D2P(I,1))/D2P(I,2)
    3 CONTINUE
C  Compute 1st and third difference on edges, and sum to get 2nd
C  and 4th
       DO 4 I=1,NFT
        N1=NFACE(1,I)
        N2=NFACE(2,I)
C  Compute coefficients for dissipation
        IF(N1.GT.0.AND.N2.GT.0)THEN
          VDT=VOL(N1)/DT(N1) + VOL(N2)/DT(N2)
          D2P1=D2P(N1,1)
          D2P2=D2P(N2,1)
          VIS1=VDT*E1*AMAX1(D2P1,D2P2)
          VIS3=VDT*E3
          VIS3=DIM(VIS3,VIS1)
         D2F1CO=D2F(N1,NCO)
         D2F1XM=D2F(N1,NXM)
         D2F1YM=D2F(N1,NYM)
         D2F1EN=D2F(N1,NEN)
         D2F2CO=D2F(N2,NCO)
         D2F2XM=D2F(N2,NXM)
         D2F2YM=D2F(N2,NYM)
         D2F2EN=D2F(N2,NEN)
        ELSE
        IF(N1.EQ.0.OR.N2.EQ.0)THEN
         N1=N1+N2
         N2=0
          VDT=VOL(N1)/DT(N1) + VOL(N1)/DT(N1)
          D2P1=D2P(N1,1)
          D2P2=D2P(N1,1)
          VIS1=VDT*E1*AMAX1(D2P1,D2P2)
          VIS3=VDT*E3
          VIS3=DIM(VIS3,VIS1)
         D2F1CO=D2F(N1,NCO)
         D2F1XM=D2F(N1,NXM)
         D2F1YM=D2F(N1,NYM)
         D2F1EN=D2F(N1,NEN)
         D2F2CO=D2F(N1,NCO)
         D2F2XM=D2F(N1,NXM)
         D2F2YM=D2F(N1,NYM)
         D2F2EN=D2F(N1,NEN)
        ELSE
        IF(N1.EQ.-1.OR.N2.EQ.-1)THEN
         N1=N1+N2+1
         N2=-1
          VDT=VOL(N1)/DT(N1) + VOL(N1)/DT(N1)
          D2P1=D2P(N1,1)
          D2P2=D2P(N1,1)
          VIS1=VDT*E1*AMAX1(D2P1,D2P2)
          VIS3=VDT*E3
          VIS3=DIM(VIS3,VIS1)
         D2F1CO=D2F(N1,NCO)
         D2F1XM=D2F(N1,NXM)
         D2F1YM=D2F(N1,NYM)
         D2F1EN=D2F(N1,NEN)
         D2F2CO=D2F(N1,NCO)
         D2F2XM=D2F(N1,NXM)
         D2F2YM=D2F(N1,NYM)
         D2F2EN=D2F(N1,NEN)
        ELSE
        IF(N1.EQ.-2.OR.N2.EQ.-2)THEN
         N1=N1+N2+2
         N2=-2
          VDT=VOL(N1)/DT(N1) + VOL(N1)/DT(N1)
          D2P1=D2P(N1,1)
          D2P2=D2P(N1,1)
          VIS1=VDT*E1*AMAX1(D2P1,D2P2)
          VIS3=VDT*E3
          VIS3=DIM(VIS3,VIS1)
         D2F1CO=D2F(N1,NCO)
         D2F1XM=D2F(N1,NXM)
         D2F1YM=D2F(N1,NYM)
         D2F1EN=D2F(N1,NEN)
         D2F2CO=D2F(N1,NCO)
         D2F2XM=D2F(N1,NXM)
         D2F2YM=D2F(N1,NYM)
         D2F2EN=D2F(N1,NEN)
        ENDIF
        ENDIF
        ENDIF
        ENDIF
C  1st difference on edge I known, and 3rd difference on edge
C  I is computed
```

```
        D13RHO =VIS1*D1F(I,NCO)
      .        -VIS3*(D2F1CO-D2F2CO)
        D13RHOU=VIS1*D1F(I,NXM)
      .        -VIS3*(D2F1XM-D2F2XM)
        D13RHOV=VIS1*D1F(I,NYM)
      .        -VIS3*(D2F1YM-D2F2YM)
        D13EN  =VIS1*D1F(I,NEN)
      .        -VIS3*(D2F1EN-D2F2EN)
C  Sum 1st and 3rd to get 2nd and 4th in each cell

        B(N1,NCO)=B(N1,NCO) - D13RHO
        B(N1,NXM)=B(N1,NXM) - D13RHOU
        B(N1,NYM)=B(N1,NYM) - D13RHOV
        B(N1,NEN)=B(N1,NEN) - D13EN
        B(N2,NCO)=B(N2,NCO) + D13RHO
        B(N2,NXM)=B(N2,NXM) + D13RHOU
        B(N2,NYM)=B(N2,NYM) + D13RHOV
        B(N2,NEN)=B(N2,NEN) + D13EN
    4 CONTINUE
      RETURN
      END



*DECK DAMP4
      SUBROUTINE DAMP4
*CALL COMMZ
      DIMENSION D1F(NFPAR,4),D2F(NCPAR,4)
       omega=e3/16.
C  Zero out storage of 2nd difference of conserved variables
      DO 1 I=1,NCT
       D2F(I,NCO)=0.0
       D2F(I,NXM)=0.0
       D2F(I,NYM)=0.0
       D2F(I,NEN)=0.0
    1 CONTINUE
C  Compute 1st difference on each edge, and sum to get 2nd
C  difference in cell
      DO 2 I=1,NFT
       N1=NFACE(1,I)
       N2=NFACE(2,I)
       IF(N1.GT.0.AND.N2.GT.0)THEN
        U1=U(N1)
        V1=V(N1)
        P1=P(N1)
```

```
          T1=T(N1)
          U2=U(N2)
          V2=V(N2)
          P2=P(N2)
          T2=T(N2)
         ELSE
C       Solid Wall
         IF(N1.EQ.0.OR.N2.EQ.0)THEN
          N1=N1+N2
          U1=U(N1)
          V1=V(N1)
          P1=P(N1)
          T1=T(N1)
          U2=-U1
          V2=-V1
          P2=P1
          T2=T1
         ELSE
C       Inlet
         IF(N1.EQ.-1.OR.N2.EQ.-1)THEN
          N1=N1+N2+1
          N2=N1
          U1=U(N1)
          V1=V(N1)
          P1=P(N1)
          T1=T(N1)
C        Subsonic inlet
          IF(NIBC.EQ.0)THEN
           U2=2.*UI(N1)-U1
           V2=2.*VI(N1)-V1
           P2=P(N2)
           T2=2.*TI(N1)-T1
          ELSE
C        Supersonic inlet
          IF(NIBC.EQ.2)THEN
           U2=2.*UI(N1)-U1
           V2=2.*VI(N1)-V1
           P2=2.*PI(N1)-P1
           T2=2.*TI(N1)-T1
          ENDIF
          ENDIF
         ELSE
C       Exit
         IF(N1.EQ.-2.OR.N2.EQ.-2)THEN
          N1=N1+N2+2
          N2=N1
          U1=U(N1)
          V1=V(N1)
          P1=P(N1)
          T1=T(N1)
C        Subsonic exit
          IF(NEBC.EQ.1)THEN
           U2=U(N1)
           V2=V(N1)
           P2=2.*PEXIT-P1
           T2=T(N1)
          ELSE
C        Supersonic exit
          IF(NEBC.EQ.2)THEN
```

```
           U2=U(N1)
           V2=V(N1)
           P2=P(N1)
           T2=T(N1)
          ENDIF
          ENDIF
         ENDIF
         ENDIF
         ENDIF
         ENDIF
C   Compute conserved variables on either side of edge I
         RHO1 =P1/T1
         RHOU1=RHO1*U1
         RHOV1=RHO1*V1
         EN1  =P1*((CP-R)+.5*(U1**2+V1**2)/T1)
         RHO2 =P2/T2
         RHOU2=RHO2*U2
         RHOV2=RHO2*V2
         EN2  =P2*((CP-R)+.5*(U2**2+V2**2)/T2)
C   Compute 1st difference on edge I and store in D1F for later use.
         DRHO =RHO1-RHO2
         DRHOU=RHOU1-RHOU2
         DRHOV=RHOV1-RHOV2
         DEN  =EN1-EN2

         D1F(I,NCO)=DRHO
         D1F(I,NXM)=DRHOU
         D1F(I,NYM)=DRHOV
         D1F(I,NEN)=DEN
C   2nd difference in cell(used later in computing 3rd difference)
         D2F(N1,NCO)=D2F(N1,NCO) - DRHO
         D2F(N1,NXM)=D2F(N1,NXM) - DRHOU
         D2F(N1,NYM)=D2F(N1,NYM) - DRHOV
         D2F(N1,NEN)=D2F(N1,NEN) - DEN
         D2F(N2,NCO)=D2F(N2,NCO) + DRHO
         D2F(N2,NXM)=D2F(N2,NXM) + DRHOU
         D2F(N2,NYM)=D2F(N2,NYM) + DRHOV
         D2F(N2,NEN)=D2F(N2,NEN) + DEN

      2 CONTINUE

C   Compute third difference on edges, and sum to get 4th
        DO 4 I=1,NFT
         N1=NFACE(1,I)
         N2=NFACE(2,I)

         IF(N1.GT.0.AND.N2.GT.0)THEN
          D2F1CO=D2F(N1,NCO)
          D2F1XM=D2F(N1,NXM)
          D2F1YM=D2F(N1,NYM)
          D2F1EN=D2F(N1,NEN)
          D2F2CO=D2F(N2,NCO)
          D2F2XM=D2F(N2,NXM)
```

```
       D2F2YM=D2F(N2,NYM)
       D2F2EN=D2F(N2,NEN)
      ELSE
      IF(N1.EQ.0.OR.N2.EQ.0)THEN
       N1=N1+N2
       N2=0
       D2F1CO=D2F(N1,NCO)
       D2F1XM=D2F(N1,NXM)
       D2F1YM=D2F(N1,NYM)
       D2F1EN=D2F(N1,NEN)
       D2F2CO=D2F(N1,NCO)
       D2F2XM=D2F(N1,NXM)
       D2F2YM=D2F(N1,NYM)
       D2F2EN=D2F(N1,NEN)
      ELSE
      IF(N1.EQ.-1.OR.N2.EQ.-1)THEN
       N1=N1+N2+1
       N2=-1
       D2F1CO=D2F(N1,NCO)
       D2F1XM=D2F(N1,NXM)
       D2F1YM=D2F(N1,NYM)
       D2F1EN=D2F(N1,NEN)
       D2F2CO=D2F(N1,NCO)
       D2F2XM=D2F(N1,NXM)
       D2F2YM=D2F(N1,NYM)
       D2F2EN=D2F(N1,NEN)
      ELSE
      IF(N1.EQ.-2.OR.N2.EQ.-2)THEN
       N1=N1+N2+2
       N2=-2
       D2F1CO=D2F(N1,NCO)
       D2F1XM=D2F(N1,NXM)
       D2F1YM=D2F(N1,NYM)
       D2F1EN=D2F(N1,NEN)
       D2F2CO=D2F(N1,NCO)
       D2F2XM=D2F(N1,NXM)
       D2F2YM=D2F(N1,NYM)
       D2F2EN=D2F(N1,NEN)
      ENDIF
      ENDIF
      ENDIF
      ENDIF

C  3rd difference on edge I is computed
      D13RHO =-(D2F1CO-D2F2CO)
      D13RHOU=-(D2F1XM-D2F2XM)
      D13RHOV=-(D2F1YM-D2F2YM)
      D13EN  =-(D2F1EN-D2F2EN)
C  Sum 3rd to get 4th in each cell
      B(N1,NCO)=B(N1,NCO) - omega*D13RHO
      B(N1,NXM)=B(N1,NXM) - omega*D13RHOU
```

```
        B(N1,NYM)=B(N1,NYM) - omega*D13RHOV
        B(N1,NEN)=B(N1,NEN) - omega*D13EN

        B(N2,NCO)=B(N2,NCO) + omega*D13RHO
        B(N2,NXM)=B(N2,NXM) + omega*D13RHOU
        B(N2,NYM)=B(N2,NYM) + omega*D13RHOV
        B(N2,NEN)=B(N2,NEN) + omega*D13EN
    4 CONTINUE
      RETURN
      END




*DECK DAMP4P
      SUBROUTINE DAMP4P
*CALL COMMZ
      DIMENSION D1F(NFPAR,4),D2F(NCPAR,4),D4F(NCPAR,4)
C  Zero out storage of 2nd difference of conserved variables
      DO 1 I=1,NCT
       D2F(I,1)=0.0
       D2F(I,2)=0.0
       D2F(I,3)=0.0
       D2F(I,4)=0.0
       D4F(I,1)=0.0
       D4F(I,2)=0.0
       D4F(I,3)=0.0
       D4F(I,4)=0.0
    1 CONTINUE
C  Compute 1st difference on each edge, and sum to get 2nd
C  difference in cell
      DO 2 I=1,NFT
       N1=NFACE(1,I)
       N2=NFACE(2,I)
       IF(N1.GT.0.AND.N2.GT.0)THEN
        U1=U(N1)
        V1=V(N1)
        P1=P(N1)
        T1=T(N1)
        U2=U(N2)
        V2=V(N2)
        P2=P(N2)
        T2=T(N2)
       ELSE
C     Solid Wall
       IF(N1.EQ.0.OR.N2.EQ.0)THEN
        N1=N1+N2
        U1=U(N1)
        V1=V(N1)
        P1=P(N1)
        T1=T(N1)
        U2=-U1
```

```
            V2=-V1
            P2=P1
            T2=T1
           ELSE
C         Inlet
           IF(N1.EQ.-1.OR.N2.EQ.-1)THEN
            N1=N1+N2+1
            N2=N1
            U1=U(N1)
            V1=V(N1)
            P1=P(N1)
            T1=T(N1)
C          Subsonic inlet
            IF(NIBC.EQ.0)THEN
             U2=2.*UI(N1)-U1
             V2=2.*VI(N1)-V1
             P2=P(N2)
             T2=2.*TI(N1)-T1
            ELSE
C          Supersonic inlet
            IF(NIBC.EQ.2)THEN
             U2=2.*UI(N1)-U1
             V2=2.*VI(N1)-V1
             P2=2.*PI(N1)-P1
             T2=2.*TI(N1)-T1
            ENDIF
            ENDIF
           ELSE
C         Exit
           IF(N1.EQ.-2.OR.N2.EQ.-2)THEN
            N1=N1+N2+2
            N2=N1
            U1=U(N1)
            V1=V(N1)
            P1=P(N1)
            T1=T(N1)
C          Subsonic exit
            IF(NEBC.EQ.1)THEN
             U2=U(N1)
             V2=V(N1)
             P2=2.*PEXIT-P1
             T2=T(N1)
            ELSE
C          Supersonic exit
            IF(NEBC.EQ.2)THEN
             U2=U(N1)
             V2=V(N1)
             P2=P(N1)
             T2=T(N1)
            ENDIF
            ENDIF
           ENDIF
           ENDIF
           ENDIF
           ENDIF
C   Compute 1st difference on edge I and store in D1F for later use.
           D1P =P1-P2
           D1U =U1-U2
           D1V =V1-V2
```

```
      D1T =T1-T2

      D1F(I,1)=D1P
      D1F(I,2)=D1U
      D1F(I,3)=D1V
      D1F(I,4)=D1T
C  2nd difference in cell(used later in computing 3rd difference)
      D2F(N1,1)=D2F(N1,1) - D1P
      D2F(N1,2)=D2F(N1,2) - D1U
      D2F(N1,3)=D2F(N1,3) - D1V
      D2F(N1,4)=D2F(N1,4) - D1T
      D2F(N2,1)=D2F(N2,1) + D1P
      D2F(N2,2)=D2F(N2,2) + D1U
      D2F(N2,3)=D2F(N2,3) + D1V
      D2F(N2,4)=D2F(N2,4) + D1T

    2 CONTINUE

C  Compute third difference on edges, and sum to get 4th
     DO 4 I=1,NFT
      N1=NFACE(1,I)
      N2=NFACE(2,I)

      IF(N1.GT.0.AND.N2.GT.0)THEN
       D2F1P=D2F(N1,1)
       D2F1U=D2F(N1,2)
       D2F1V=D2F(N1,3)
       D2F1T=D2F(N1,4)
       D2F2P=D2F(N2,1)
       D2F2U=D2F(N2,2)
       D2F2V=D2F(N2,3)
       D2F2T=D2F(N2,4)
      ELSE
      IF(N1.EQ.0.OR.N2.EQ.0)THEN
       N1=N1+N2
       N2=0
       D2F1P=D2F(N1,1)
       D2F1U=D2F(N1,2)
       D2F1V=D2F(N1,3)
       D2F1T=D2F(N1,4)
       D2F2P=D2F(N1,1)
       D2F2U=-D2F(N1,2)
       D2F2V=-D2F(N1,3)
       D2F2T=D2F(N1,4)
      ELSE
      IF(N1.EQ.-1.OR.N2.EQ.-1)THEN
       N1=N1+N2+1
       N2=-1
       D2F1P=D2F(N1,1)
       D2F1U=D2F(N1,2)
       D2F1V=D2F(N1,3)
       D2F1T=D2F(N1,4)
```

```
           D2F2P=D2F(N1,1)
           D2F2U=D2F(N1,2)
           D2F2V=D2F(N1,3)
           D2F2T=D2F(N1,4)
          ELSE
          IF(N1.EQ.-2.OR.N2.EQ.-2)THEN
           N1=N1+N2+2
           N2=-2
           D2F1P=D2F(N1,1)
           D2F1U=D2F(N1,2)
           D2F1V=D2F(N1,3)
           D2F1T=D2F(N1,4)
           D2F2P=D2F(N1,1)
           D2F2U=D2F(N1,2)
           D2F2V=D2F(N1,3)
           D2F2T=D2F(N1,4)
          ENDIF
          ENDIF
          ENDIF
          ENDIF

C  3rd difference on edge I is computed
          D13P =D2F1P-D2F2P
          D13U =D2F1U-D2F2U
          D13V =D2F1V-D2F2V
          D13T =D2F1T-D2F2T
C  Sum 3rd to get 4th in each cell
          D4F(N1,1)=D4F(N1,1) - D13P
          D4F(N1,2)=D4F(N1,2) - D13U
          D4F(N1,3)=D4F(N1,3) - D13V
          D4F(N1,4)=D4F(N1,4) - D13T

          D4F(N2,1)=D4F(N2,1) + D13P
          D4F(N2,2)=D4F(N2,2) + D13U
          D4F(N2,3)=D4F(N2,3) + D13V
          D4F(N2,4)=D4F(N2,4) + D13T
    4 CONTINUE
C      Compute smoothing coefficient and add to cell
         DO 5 I=1,NCT
          S=VOL(I)
          DTAU=CFL*AMIN1(DT(I),DTMIN)
          omega=e3/DTAU*S/T(I)/16.
C  Include 4th dissipation in each cell
          B(I,NCO)=B(I,NCO) - omega*(R*D4F(I,1)
     .                         -P(I)/T(I)*D4F(I,4))
          B(I,NXM)=B(I,NXM) - omega*(R*U(I)*D4F(I,1)
     .                                  +P(I)*D4F(I,2)
     .                         -P(I)*U(I)/T(I)*D4F(I,4))
```

```
        B(I,NYM)=B(I,NYM) - omega*(R*V(I)*D4F(I,1)
     .                              +P(I)*D4F(I,3)
     .                    -P(I)*V(I)/T(I)*D4F(I,4))
        B(I,NEN)=B(I,NEN) -
     . omega*(R*((CP-R)*T(I)+.5*(U(I)**2+V(I)**2))*D4F(I,1)
     .                                   +P(I)*U(I)*D4F(I,2)
     .                                   +P(I)*V(I)*D4F(I,3)
     .              -.5*P(I)/T(I)*(U(I)**2+V(I)**2)*D4F(I,4))
    5 CONTINUE
      RETURN
      END



*DECK OUTPUT
      SUBROUTINE OUTPUT
*CALL COMMZ
      WRITE(15,100)
      DO 1 I=1,NCT
       WRITE(15,110)I,U(I),V(I),P(I),T(I)
    1 CONTINUE
  100 FORMAT(///,2X,'Solution Vector',//,3X,'Cell',6X,
     . 'U(I)',8X,'V(I)',8X,'P(I)',8X,'T(I)')
  110 FORMAT(I6,4F12.5)
      RETURN
      END



*DECK PLOUT1
      SUBROUTINE PLOUT1
C     Print dimensional output for SGI based graphic plotting
C     routines
*CALL COMMZ
      WRITE(30,*)NCT
      WRITE(30,*)G,RO
      DO 1 I=1,NCT
      N1=NCELL(4,I)
      N2=NCELL(5,I)
      N3=NCELL(6,I)
      UOUT=U(I)*UO
      VOUT=V(I)*UO
      POUT=P(I)*RHOO*UO**2
      TOUT=T(I)*TO
       EU=ABS(XI(I,1))
       EV=ABS(XI(I,2))
       EP=ABS(XI(I,3))
       ET=ABS(XI(I,4))
      WRITE(30,*)X(N1),Y(N1),X(N2),Y(N2),X(N3),Y(N3)
      WRITE(30,*)UOUT,VOUT,POUT,TOUT,(RESXI(I,K),K=1,4)
c      WRITE(30,*)UOUT,VOUT,POUT,TOUT,EU,EV,EP,ET
```

```
    1 CONTINUE
      RETURN
      END




*DECK PLOUT2
      SUBROUTINE PLOUT2
*CALL COMMZ
C     Print output for Grafic plotting routines
      DIMENSION DAT(6,NNPAR),NDAT(NNPAR)
      DO 2 N=1,NCT
       PRES=P(N)*RHOO*UO**2
       UVEL=U(N)*UO
       VVEL=V(N)*UO
       TEMP=T(N)*TO
        RHO=PRES/RO/TEMP
        RHOU=RHO*UVEL
        RHOV=RHO*VVEL
        ET=PRES/RHO/(G-1.)+.5*(UVEL**2+VVEL**2)
        ET=RHO*ET
       DO 2 NN=4,6
        IN=NCELL(NN,N)
        DAT(3,IN)=DAT(3,IN)+RHO
        DAT(4,IN)=DAT(4,IN)+RHOU
        DAT(5,IN)=DAT(5,IN)+RHOV
        DAT(6,IN)=DAT(6,IN)+ET
        NDAT(IN)=NDAT(IN)+1
    2 CONTINUE
C     Fix node point values on boundary for viscous solution
C     Reset A and B nodes to 0.0
       DO 3 N=1,NCT
        N1=NCELL(4,N)
        N2=NCELL(5,N)
        NCF=NCELL(1,N)
        NCC=NFACE(1,NCF)+NFACE(2,NCF)-N
        IF(NCC.EQ.0)THEN
         DAT(4,N1)=0.0
         DAT(5,N1)=0.0
         DAT(4,N2)=0.0
         DAT(5,N2)=0.0
        ENDIF
    3  CONTINUE
       DO 5 N=1,NNT
        DAT(1,N)=X(N)
        DAT(2,N)=Y(N)
        DAT(3,N)=DAT(3,N)/NDAT(N)
        DAT(4,N)=DAT(4,N)/NDAT(N)
        DAT(5,N)=DAT(5,N)/NDAT(N)
        DAT(6,N)=DAT(6,N)/NDAT(N)
    5 CONTINUE
      WRITE(1,"(A22)")"unstructured grid data"
      WRITE(1,*)NCT
      NCORNERS=3
      DO 6 I=1,NCT
```

```
         I1=NCELL(4,I)
         I2=NCELL(5,I)
         I3=NCELL(6,I)
         WRITE(1,*)NCORNERS,I1,I2,I3
    6   CONTINUE
        WRITE(1,*)NNT
         RINF=1.0
         RUINF=1.0
         RVINF=1.0
         ETINF=1.0
        WRITE(1,*)RINF,RUINF,RVINF,ETINF
        DO 7 I=1,NNT
         WRITE(1,*)(DAT(J,I),J=1,6)
    7   CONTINUE
        NBODIES=0
c        NBODIES=1
        WRITE(1,*)NBODIES
        NBOD=0
c        NBOD=49
        WRITE(1,*)NBOD
        DO 8 I=101,148
        IP=I+1
        WRITE(1,*)I,IP
    8   CONTINUE
c        I=299
c        IP=251
c        WRITE(1,*)I,IP
c        NBOD=49
c        WRITE(1,*)NBOD
c        DO 9 I=300,347
c        IP=I+1
c        WRITE(1,*)I,IP
c    9  CONTINUE
        I=149
        IP=101
        WRITE(1,*)I,IP
        NBOUNDA=100
        WRITE(1,*)NBOUNDA
        DO 10 I=1,NBOUNDA-1
        IP=I+1
        WRITE(1,*)I,IP
   10   CONTINUE
        I=NBOUNDA
        IP=1
        WRITE(1,*)I,IP
C       Compute Solid Wall pressure for plotxy
        DO 11 I=101,125
        NL=I
        RHO=DAT(3,NL)
        UVEL=DAT(4,NL)/RHO
        VVEL=DAT(5,NL)/RHO
        ET=DAT(6,NL)
        PRES=(G-1.)*(ET-.5*RHO*(UVEL**2+VVEL**2))
        WRITE(40,*)X(NL),PRES
   11   CONTINUE
        RETURN
        END
```

```
*DECK REREAD
      SUBROUTINE REREAD
*CALL COMMZ
      READ(35)DTMIN,CP,R,PEXIT,NCOUNT
      READ(35)POI,SC1,SC2,TW
      DO 1 I=1,NILT
      N=NCELLIL(I)
      READ(35)UI(N),VI(N),PI(N),TI(N)
    1 CONTINUE
      DO 2 I=1,NCT
      READ(35)U(I),V(I),P(I),T(I),DT(I),VOL(I)
    2 CONTINUE
C   Residual information
      READ(35)NRES
      DO 3 I=1,NRES
       READ(60,*)RES5(1,I),RES5(2,I)
    3 CONTINUE
C   Initialize A matrix and b vector
      DO 4 J=0,10
      DO 4 I=0,NCT
      A(I,J,1,1)=0.0
      A(I,J,1,2)=0.0
      A(I,J,1,3)=0.0
      A(I,J,1,4)=0.0
      A(I,J,2,1)=0.0
      A(I,J,2,2)=0.0
      A(I,J,2,3)=0.0
      A(I,J,2,4)=0.0
      A(I,J,3,1)=0.0
      A(I,J,3,2)=0.0
      A(I,J,3,3)=0.0
      A(I,J,3,4)=0.0
      A(I,J,4,1)=0.0
      A(I,J,4,2)=0.0
      A(I,J,4,3)=0.0
      A(I,J,4,4)=0.0
    4 CONTINUE
      DO 5 I=0,NCT
      B(I,1)=0.0
      B(I,2)=0.0
      B(I,3)=0.0
      B(I,4)=0.0
      UP(I)=U(I)
      VP(I)=V(I)
      PP(I)=P(I)
      TP(I)=T(I)
    5 CONTINUE
      RETURN
      END


*DECK REWRITE
```

```
      SUBROUTINE REWRITE
*CALL COMMZ
      REWIND(35)
      REWIND(60)
      WRITE(35)DTMIN,CP,R,PEXIT,NCOUNT
      WRITE(35)POI,SC1,SC2,TW
      DO 1 I=1,NILT
      N=NCELLIL(I)
      WRITE(35)UI(N),VI(N),PI(N),TI(N)
    1 CONTINUE
      DO 2 I=1,NCT
      WRITE(35)U(I),V(I),P(I),T(I),DT(I),VOL(I)
    2 CONTINUE
C   Residual information
      WRITE(35)NRES
      DO 3 I=1,NRES
       WRITE(60,*)RES5(1,I),RES5(2,I)
    3 CONTINUE
      RETURN
      END



*DECK energy
      SUBROUTINE energy
*CALL COMMZ
C  Added to compute isothermal flow.
      do 1 l=1,4
      do 1 j=1,ncpl
      do 1 i=1,nct
        a(I,J,NEN,L)=0.0
        a(I,J,L,NEN)=0.0
    1 continue

      do 2 i=1,nct
        b(I,NEN)=0.0
    2 continue

      do 3 i=1,nct
        a(I,1,NEN,NEN)=1.0
    3 continue

      RETURN
      END



*COMDECK COMMZ
      PARAMETER(NCPAR=10000,NFPAR=15000,NNPAR=6000,
     . NBLOCK=4,NCPL=10,LIWORK=NCPAR*200,LWORK=LIWORK)
      COMMON/VAR/NTTS,NLIN,CFL,NDTT,NIBC,NEBC,CPO,RO,XMUO,PR,
     . PO,TO,PSRAT,UT,UTANG,TW,CP,CV,G,R,XMU,XLREF,RHOO,
     . UO,RENO,NROWB,NFT,NCT,NNT,NILT,NSI,KBV,
     . PEXIT,NSOLVE,RSQ,DELT,DTMIN,
     . NRST,NCOUNT,NDAMP,E1,E3,POI,SC1,SC2,NXM,NYM,NEN,NCO,
     . NPRET,IGRID,NRES,PSEUDO,DTAU
      COMMON/ARRAY/X(NNPAR),Y(NNPAR),NCELL(6,NCPAR),NFACE(2,NFPAR),
. U(NCPAR),V(NCPAR),P(NCPAR),T(NCPAR),UP(NCPAR),VP(NCPAR),
. PP(NCPAR),TP(NCPAR),A(0:NCPAR,0:10,4,4),B(0:NCPAR,4),
. NUMEL(NCPAR,10),NCELLIL(NCPAR),NRGBY(4),NCOLOR(4,NCPAR),
. DT(NCPAR),VOL(NCPAR),XI(NCPAR,4),
. RESXI(NCPAR,4),RES5(2,10000),NPERM(3,3),ICS(3),
. UI(NCPAR),VI(NCPAR),PI(NCPAR),TI(NCPAR)
 COMMON/SMS/IC,IR,NEQNS,IB(NCPAR,NCPL),JB(NCPAR,NCPL),
. NBC(NCPAR),ROWIND(NCPAR*NCPL*4*4),COLPTR(NCPAR*NCPL+1),
. AC(NCPAR*NCPL*4*4),RHS(NCPAR*4),XR(NCPAR*4)
 INTEGER ROWIND,COLPTR
```