# Design and Analysis of Internal Flowfields Using a Two Stream Function Formulation

by

**Mark Graham Turner**

B.S. Mechanical Engineering, Virginia Polytechnic Institute and State University (1979)
S.M. Department of Aerospace Engineering and Applied Mechanics, University of Cincinnati (1986)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

**Doctor of Science**
in
**Aerodynamics**
**Department of Aeronautics and Astronautics**
at the
**Massachusetts Institute of Technology**

January 1990

Signature of Author ______________________
Department of Aeronautics and Astronautics
12 January, 1990

Certified by ______________________
Professor Michael B. Giles
Thesis Supervisor, Department of Aeronautics and Astronautics

Certified by ______________________
Professor Alan H. Epstein
Department of Aeronautics and Astronautics

Certified by ______________________
Professor Lloyd N. Trefethen
Department of Mathematics

Accepted by ______________________
Professor Harold Y. Wachman
Chairman, Department Graduate Committee

# Design and Analysis of Internal Flowfields Using a Two Stream Function Formulation

by

Mark Graham Turner

Submitted to the Department of Aeronautics and Astronautics on
12 January, 1990
in partial fulfillment of the requirements for the Degree of
Doctor of Science in Aerodynamics

## Abstract

A method is developed for the solution of the steady 3D Euler equations based on a two stream function formulation, primarily for turbomachinery applications. Surfaces of constant stream function represent stream surfaces, so the use of this approach automatically satisfies the continuity equation. Correct shock capturing is permitted by using a conservative form of the discrete equations along with a pressure upwinding scheme. This adds a small amount of artificial viscosity in supersonic regions, which ensures the well-posedness of the problem. The method uses a fixed grid, although each blade-to-blade grid line is allowed a degree of freedom in the tangential direction to solve for the stagnation stream surfaces and to provide a design option capability. A blade can be designed by specifying its thickness and loading.

The discrete equation system is solved using Newton's method, which provides rapid convergence of the solution. Newton's method requires a matrix solution at each iteration, and a new matrix solution procedure is presented which can use a previous matrix factor as a preconditioner to a conjugate-gradient-like algorithm called GMRES. This reduces the total number of matrix factorizations required.

Second-order accuracy of the method is demonstrated in 2D. Several cascade solutions are also presented and compare favorably with analytic, experimental and other numerical results. The 3D results are presented for a low pressure turbine nozzle with uniform total pressure. Streamwise vorticity can cause excessive warping of the stream surfaces, which makes the solution diverge. To overcome this limitation, two test cases are presented which have just one volume blade-to-blade. The results are compared to experimental data of the exit angle or tangential velocity distribution. The design option capability is also presented in both 2D and 3D.

Thesis Supervisor: Michael B. Giles
Title: Assistant Professor of Aeronautics and Astronautics

## Acknowledgements

It is impossible to undertake a project such as this without the help and support from many people. I would like to thank everyone who has helped me in this endeavor over the past six years.

First and foremost, I would like to thank my wife, Nancy (Beam, the light and love of my life). You have provided a haven of sanity in this otherwise hectic and weird world of higher learning. Thanks for the understanding and patience.

Next, I would like to thank my advisor, Professor Mike Giles, for supporting my ideas in this work, providing his own ideas and good advise, and editing this thesis. I would also like to thank the other members of my committee, Professors Alan Epstein and Nick Trefethen who have provided good guidance and teaching for this research. I want to acknowledge my appreciation to Professor Mark Drela for being a reader and for suggesting the use of two stream functions and his other ideas and discussions. Thanks also to my other inside reader, Dr. Choon Tan. To Jim Keith, who was my outside reader as well as friend, colleague, first boss at GE and first mentor in the field of CFD–I am deeply grateful for all your help. To Bob Haimes, thanks for the sneakernet service as well as being another commuter who already realizes how depressing the Central Artery is. Thanks also go to the other professors in the department who have been such excellent teachers including Professors Murman, Covert and Greitzer.

I have been lucky to have been supported by the General Electric Company through the Advanced Course in Engineering program. Through this program, I have been a full time student at MIT for over three years while still employed by GE. Tuition, books and computers have also been provided. To the people at GE, I am very grateful to both the support you've given and the confidence you've had in me. These people include my managers Mike Suo and Roy Smith, the Advanced Course supervisors Samir Sayegh and Mark Herder, and the late Dick Novak, who was my first manager after moving to Boston.

The fellow students I've come to know during my tenure here have provided the necessary outlets for support, complaining, technical discussions (i.e., bullshitting) and other insanity releases for which I would never have finished. Thanks to Steve Allmaras, my first officemate and good friend. To Dana Lindquist, my friend and current officemate, who helped tremendously by proof-reading this thesis. To Earl, Cathy and Rob for their friendship as well as taking care of Tara when we needed to get away. And to my other friends Pete, Matt, Mike, Yannis, André, Hélène, Kousuke, Rich, Sandy, Josh, Bernard, Gerd, Dave, Peter, Yuan, Gerry, Eric, Sean, John, Ken, Tom, Sharon, Todd, Björn and Yucef.

For the support I've received from my family, I am truly grateful. Thanks Mom, Dad, Bob, Caroline, Cary, Laurie, Amy, Russ, Jim, Nancy, Nancy (yes, three Nancys), Eric, Dave, Brenda, Rich, Patrice, Bill and Kathleen. And finally to our dog Tara, who's unconditional devotion and affection kept us all sane.

To the memory of my grandfather, Rear Admiral Lucien McKee Grant, (M.S., MIT 1923), early U.S. naval aviator, aeronautical engineer and member of the NACA.

# Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| $A$ | relates to the direction approximately normal to $\xi^2$ and $\xi^3$ |
| $\overline{A}$ | face area vector |
| $\mathbf{A}$ | general matrix |
| $b$ | stream tube height in third dimension |
| $\mathbf{b}$ | matrix equation right hand side |
| $B$ | relates to the direction approximately normal to $\xi^1$ and $\xi^3$ |
| $\mathbf{B}$ | generalized coordinate transformation matrix or the permuted matrix |
| $c$ | speed of sound |
| $\mathbf{c}$ | permuted matrix right hand side vector |
| $C$, $C_1$, $C_2$ | curvature terms in auxiliary pressure equations |
| $\overline{C}$ | absolute velocity vector |
| $C_v$, $C_p$ | specific heats at constant volume, pressure |
| $\mathbf{d}$ | constant vector of linear operator |
| $\hat{e}$ | general unit base vector |
| $err$ | entropy error |
| $EM$ | 1D momentum equation residual |
| $ELIM$ | linear equation elimination function |
| $f_i$ | the $i$'th equation in $\mathbf{F}$ |
| $\mathbf{F}$ | vector of nonlinear equations |
| $\overline{F}_b$ | body force |
| $h$ | static enthalpy |
| $H$ | total enthalpy |
| $I$ | rothalpy |
| $\mathbf{I}$ | identity matrix |
| $i$ | spatial mode |
| $i$, $j$, $k$ | computational indices (blade-to-blade, hub-to-tip, streamwise) |
| $\hat{i}$, $\hat{j}$, $\hat{k}$ | Cartesian unit vectors |
| $\mathbf{J}$ | Jacobian matrix |
| $k$ | Newton iteration counter or the constant in the stability analysis ODE |
| $l_2$ | least squares norm |
| $\check{L}_n$ | reference length for nondimensionalizing |
| $\mathbf{L}$ | lower triangular matrix |
| $\dot{m}$ | mass flow |
| $m'$ | coordinate normal to $\theta$ which is a normalized arc length in the $r$-$z$ plane |
| $M$ | Mach number |
| $\mathbf{M}$ | general linear operator |
| $n$ | grid size number in accuracy study |
| $\overline{n}$ | outward unit vector normal to boundary or face |
| $n1e$ | total number of $\psi_1$ values |
| $n2e$ | total number of $\psi_2$ values |
| $naf$ | total number of $A$ faces |
| $nbf$ | total number of $B$ faces |
| $ndx1$ | number of grid movement degrees of freedom |
| $ni$ | total number of grid points in $\xi^1$ direction |

| | |
|---|---|
| $nj$ | total number of grid points in $\xi^2$ direction |
| $nk$ | total number of grid points in $\xi^3$ direction |
| $nm$ | total number of grid volumes |
| $nnleq$ | number of nonlinear equations |
| $nsbc$ | number of entropy boundary conditions |
| $nsf$ | total number of $S$ faces |
| $nsi1up$ | number of $\psi_1$ angle boundary conditions |
| $nsi2up$ | number of $\psi_2$ angle boundary conditions |
| $nsle$ | number of leading edge arc length degrees of freedom |
| $nwall1$ | number of $\psi_1$ wall boundary conditions |
| $nwall2$ | number of $\psi_2$ wall boundary conditions |
| NEQ | size of a matrix |
| NZs | number of nonzeros in a matrix |
| $O$ | order of symbol |
| $P$ | static pressure |
| $\tilde{P}$ | upwinded static pressure |
| $P_{cA}, P_{cB}$ | correction term for the $A$ or $B$ auxiliary pressure equations |
| $\mathbf{P}$ | permutation matrix |
| $r$ | radius |
| $\bar{r}$ | position vector |
| $R$ | perfect gas constant |
| $\Re^n$ | $n$ dimensional Euclidean space |
| $s$ | entropy or surface of control volume |
| $\mathbf{s}$ | vector of changes in solution vector determined by a Newton step |
| $\overline{s}$ | vector connecting cell edge centers |
| $\hat{s}$ | entropy variable which is defined for both compressible and incompressible flow |
| $S$ | relates to the streamwise direction |
| $\overline{S}$ | rotational source term in momentum equation |
| $S_1$,$S_2$ | Wu's stream surfaces |
| $\bar{t}$ | vector connecting face centers |
| $t$ | time |
| $T$ | static temperature |
| $u$ | velocity magnitude |
| $\overline{U}$ | coordinate system velocity vector |
| $\mathbf{U}$ | upper triangular matrix |
| $V$ | velocity magnitude |
| $\overline{V}$ | velocity vector |
| $Vol$ | volume |
| $\mathcal{V}$ | control volume |
| $W$ | relative velocity magnitude |
| $\overline{W}$ | relative velocity vector |
| $\mathbf{x}$ | nonlinear equation solution vector or general matrix solution vector |
| $x$, $y$, $z$ | Cartesian position components, $z$ is the axial component |
| $\mathbf{y}$ | permuted matrix solution vector |
| $\mathbf{z}$ | intermediate matrix solution vector |

| | |
|---|---|
| $\alpha$ | absolute tangential flow angle |
| $\beta$ | relative tangential flow angle |
| $\gamma$ | ratio of specific heats, $\gamma = C_p/C_v$ |
| $\Delta$ | difference between two quantities |
| $\Delta s_{le}$ | leading edge arc length |
| $\Delta s_{legl}$ | leading edge arc length on a grid line |
| $\Delta x_1$ | grid movement degree of freedom blade-to-blade |
| $\delta$ | change in a variable in an iteration |
| $\varepsilon$ | small distance or error tolerance |
| $\eta$ | 2D computational coordinate normal to $\xi$ |
| $\theta$ | cylindrical coordinate direction |
| $\varsigma'$ | viscous stress tensor |
| $\lambda$ | damping factor or stream tube height in 1D analysis |
| $\mu$ | artificial viscosity coefficient |
| $\nu$ | coefficient of kinematic viscosity |
| $\xi$ | streamwise computational coordinate in 2D |
| $\xi^1$, $\xi^2$, $\xi^3$ | computational coordinates in 3D, $\xi^1$ is blade-to-blade, $\xi^2$ is hub-to-tip, and $\xi^3$ is streamwise |
| $\rho$ | static density |
| $\omega$ | angular velocity magnitude |
| $\overline{\omega}$ | vorticity vector |
| $\overline{\Omega}$ | angular velocity vector |
| $\phi$ | flow angle in $r$-$z$ plane |
| $\phi'$ | angle in $r$-$z$ plane of a meridional surface |
| $\psi$ | 2D stream function |
| $\psi_1$, $\psi_2$ | 3D stream functions |
| $\psi_3$ | third dimension of a 3D space with $\psi_1$, $\psi_2$ as the other dimensions |

## subscripts

| | |
|---|---|
| $(\ )_0$ | node location or initial |
| $(\ )_1$ | first coordinate direction or node location |
| $(\ )_2$ | second coordinate direction or node location |
| $(\ )_3$ | third coordinate direction or node location |
| $(\ )_A$ | relates to the direction approximately normal to $\xi^2$ and $\xi^3$ |
| $(\ )_{A1}$, $(\ )_{A2}$ | relates to two of the $A$ faces |
| $(\ )_B$ | relates to the direction approximately normal to $\xi^1$ and $\xi^3$ |
| $(\ )_{B1}$, $(\ )_{B2}$ | relates to two of the $B$ faces |
| $(\ )_c$ | correction or critical or centered |
| $(\ )_i$ | general index |
| $(\ )_i$, $(\ )_j$, $(\ )_k$ | computational indices (blade-to-blade, hub-to-tip, streamwise) |
| $(\ )_k$ | Newton iteration counter |
| $(\ )_n$ | reference value for nondimensionalizing or refers to node |
| $(\ )_{ref}$ | reference conditions |

| | |
|---|---|
| $(\ )_S$ | relates to the streamwise direction |
| $(\ )_{S1}$, $(\ )_{S2}$ | relates to two of the streamwise faces |
| $(\ )_r$ | $r$ component |
| $(\ )_s$ | static quantity |
| $(\ )_T$ | total quantity |
| $(\ )_x$ | $x$ component |
| $(\ )_y$ | $y$ component |
| $(\ )_z$ | $z$ component |
| $(\ )_\theta$ | $\theta$ component |
| $(\ )_\psi$ | 2D or 3D $\psi$ (stream function) space |

## superscripts

| | |
|---|---|
| $\check{(\ )}$ | dimensional quantity |
| $\hat{(\ )}$ | base vector or approximate matrix factor |
| $\bar{(\ )}$ | average |
| $\tilde{(\ )}$ | upwinded value |
| $(\ )^1$ | first coordinate direction |
| $(\ )^2$ | second coordinate direction |
| $(\ )^3$ | third coordinate direction |
| $(\ )^n$ | iteration counter |
| $(\ )^{\mathrm{T}}$ | transpose |
| $(\ )'$ | refers to reduced matrix system or see $m'$, $\phi'$, $\varsigma'$ or perturbation quantity |

# Chapter 1
# Introduction

There is a large incentive to reduce the weight, size and part count of an advanced gas turbine engine while still maintaining or increasing efficiency. Therefore the compressor and turbine blades are becoming more highly loaded with lower aspect ratios, and the inter-blade spacings are getting shorter. The resulting flowfields are quite complex, and are highly unsteady, viscous, and three dimensional. The performance of the turbomachinery components can be improved by understanding these complex flowfields and by using accurate numerical prediction techniques integrated with designer experience and hardware testing.

Full 3D simulations of unsteady blade row interactions were first performed by Koya and Kotake [40] using the Euler equations and more recently by Rai [52] using the Reynolds averaged Navier-Stokes equations. These are time consuming and expensive calculations which cannot yet be performed within a design loop on a day-to-day basis. Steady 3D blade row interaction methods solving the Euler equations have been described by Ni [44] and Celestina, Mulac and Adamczyk [10]. These are useful for analysis and are important in design if interaction effects are strong, such as with the design of a counter-rotating unducted fan [57]. However, these solutions are obtained only as a final check of a design.

Other 3D methods can be broken into several categories including potential, time marching and pressure correction. A potential method for turbomachinery applications has been presented by Prince [51] and Laskaris [42]. The potential method application to turbomachinery is limited due to the highly rotational nature of the flowfield inside a blade row and the vorticity generated by shocks. The solvers can be adapted to handle rotational flow, but often at the expense of robustness and speed. This is especially true in determining the correct entropy rise across a shock.

Several time marching methods are described by Ni [43], Jameson [36], and Holmes [34]. These employ a relatively simple algorithm which marches the hyperbolic equations in time. They are easily vectorizable for supercomputer applications, but suffer from time step restrictions due to stability requirements. Stability also requires the addition of dissipative smoothing to kill unwanted modes of the solution. This smoothing often causes large numerical errors to be generated. These errors can be monitored in inviscid flow by detecting false entropy generation. However, in viscous simulations these dissipative errors often mask the physical dissipation.

Pressure correction methods for turbomachine applications have been presented by Hah [30] and Rhie [53]. These methods solve a pressure equation derived from the divergence of the momentum equation and continuity equation. This equation, the momentum equations and other scalar equations are solved sequentially using a line relaxation implicit procedure. The methods were initially developed for incompressible applications, and accurate robust transonic extensions are not trivial.

Again, the use of these 3D methods is as a final check of a design. The blades are actually designed using a quasi-3D approach. In this approach, the 2D axisymmetric through-flow solution and a series of 2D blade-to-blade solutions are superimposed. Some true 3D effects such as secondary flow mixing can be added to the through-flow analysis [2], but many of the 3D effects are not accounted for. Oates [47] describes several through-flow methods. The blade-to-blade solvers include the 3D methods already described reduced to 2D, in addition to streamline curvature methods [45] and a related fixed grid approach [60]. Giles and Drela [25] [27] [16] have developed ISES, a 2D intrinsic grid Euler solver which can be used for both design and analysis.

ISES has many attractive properties. It uses Newton's method to solve the non-linear set of equations. This provides rapid convergence and fast solution times. It is accurate and requires only a small amount of added numerical dissipation in supersonic regions. The discretization is conservative so the correct Rankine-Hugoniot shock jump conditions are satisfied in the limit of infinite grid resolution. In addition to being an analysis tool, it can be used in an inverse mode where the pressure on each blade surface is prescribed to produce a blade shape. Alternatively, a combination of inverse and analysis modes can be used. An effective boundary layer coupling strategy has also been implemented with this code.

This thesis is an extension of the ISES approach to both fixed grids and 3D. The intent has been to develop an algorithm which has the attributes of ISES and can be used for actual 3D design. It is an attempt to extend the design process beyond the quasi-3D approach.

ISES uses an intrinsic grid (the grid lines are streamlines) so the grid movement is a dependent variable. Going to a fixed grid in 2D requires that the stream function $\psi$ becomes a dependent variable. An intrinsic grid in 3D would require working with 3D stream surfaces, which requires a surface modeling capability that was not readily available. A two stream function formulation has been chosen as the way to extend the ISES approach to 3D with a fixed grid. The inverse design capability has been maintained by allowing whole grid lines a degree of freedom in the blade-to-blade direction. This maintains a prescribed thickness which is often a constraint of the compressor or turbine designer. The loading ($\Delta P$) is prescribed and essentially the camber line is the result.

The two stream function formulation is based on a mathematical identity such that if

$$\rho\overline{W} = \nabla\psi_1 \times \nabla\psi_2, \tag{1.1}$$

then the mass flux vector $\rho\overline{W}$ is divergence free and satisfies the incompressible or steady compressible continuity equation. The scalars $\psi_1$ and $\psi_2$ are the stream functions, and surfaces of constant $\psi_1$ or $\psi_2$ represent stream surfaces. The intersection of two stream surfaces is a streamline, so two points which have the same values of both $\psi_1$ and $\psi_2$ are on the same streamline. It is believed that this is the first time a two stream function formulation has been used as the basis of a numerical solver.

The use of two stream functions is similar to the $S_1$, $S_2$ surfaces described by Wu

[66]. This is an intrinsic grid approach, but applications have been implemented with the quasi-3D assumption in which the $S_1$ surfaces are axisymmetric. Other researchers have extended the 2D stream function to 3D [32] [14], but these do not satisfy Eq. (1.1) and are referred to as stream-like functions.

The system of discrete equations which describes the flowfield is solved using Newton's method. This requires both the calculation of the Jacobian matrix and the solution of this matrix. The resulting matrix is large and sparse. Wigton [63] has applied the ISES algorithm using Newton's method to multielement airfoils. He used MACSYMA (a symbolic algebraic manipulation package) to differentiate the equations symbolically and automatically generate the FORTRAN code. He also explored the use of sparse matrix methods to solve the matrix equation. The present work has extended these ideas. SMP (Symbolic Manipulation Program) has been used instead of MACSYMA, and fewer mistakes have been made in the program development than if the derivatives were derived and coded manually. Also, automatic procedures have been used which make changes in the equation system easy. Sparse matrix methods have been explored as well as a new approach which uses GMRES to solve the matrix with a previous matrix factor as the preconditioner. GMRES is a conjugate-gradient-like method used to accelerate and stabilize an iterative procedure. It was first developed by Saad and Schultz [55] for solving linear systems and Wigton, et al. [64] extended the method to non-linear equation systems. In this thesis, GMRES has been applied using a FORTRAN subroutine library written at Boeing [8], [9]. The linear matrix equation is solved using GMRES with an iterative procedure that uses a matrix factor from a prior Newton iteration.

In Chapter 2, the governing equations using the two stream functions are derived and the boundary conditions formulated. Chapter 3 describes the discretization procedure, and Chapter 4 presents the auxiliary pressure equations as well as the discretized boundary conditions which are required to close the system of equations. Chapter 5 describes the solution method. This includes Newton's method with the use of SMP, a method of combining equations to reduce the matrix size, and the matrix solvers used and investigated. It also discusses the use of GMRES as part of a matrix solution method.

Two dimensional results are presented in Chapter 6. An accuracy study has been performed using a $\sin^2(\pi x)$ bump test case. This case has also been used to investigate several matrix solution methods. The incompressible Gostelow cascade results are presented, as well as a high turning turbine cascade (T7) and a supercritical cascade designed by Garabedian. A demonstration of the design option capability is also presented. A blade is designed which has a NACA 0012 thickness distribution and a prescribed loading and inlet angle.

The 3D results are presented in Chapter 7. The first case describes the stream function development in a square duct with uniform streamwise vorticity. The limitations of the method for handling rotational flows are discussed. Results for the NASA/General Electric Energy Efficient Engine ($E^3$) turbine nozzle and NASA 67 transonic fan are then presented. Following this is a demonstration of the 3D design capability.

Chapter 8 presents some final conclusions and describes possible extensions to this thesis.

# Chapter 2
# Governing Equations and Boundary Conditions

## 2.1 The Continuity Equation and Two Stream Functions

The continuity equation in differential form is

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \overline{W}\right) = 0, \tag{2.1}$$

where $\rho$ is the density, $t$ is time and $\overline{W}$ is the relative velocity vector. $\overline{W}$ is related to the absolute velocity vector $\overline{C}$ and the velocity of the coordinate system $\overline{U}$ (assumed to be some combination of steady translation and rotation) by

$$\overline{C} = \overline{W} + \overline{U}. \tag{2.2}$$

For steady flow, or if the flow is incompressible, the flow is divergence free and satisfies

$$\nabla \cdot \left(\rho \overline{W}\right) = 0. \tag{2.3}$$

The mass flux vector $\rho\overline{W}$ can be related to the gradient of two scalars such that

$$\rho \overline{W} = \nabla \psi_1 \times \nabla \psi_2, \tag{2.4}$$

which implicitly satisfies Eq. (2.3). These scalars are the stream functions $\psi_1$ and $\psi_2$ (see [38] and [24]).

The use of a stream function for analyzing 2D flows is very common (see [48] and [3]). However, the use of two stream functions in 3D is discussed theoretically in some textbooks and papers, but this thesis is the first application for a numerical method that the author is aware of.

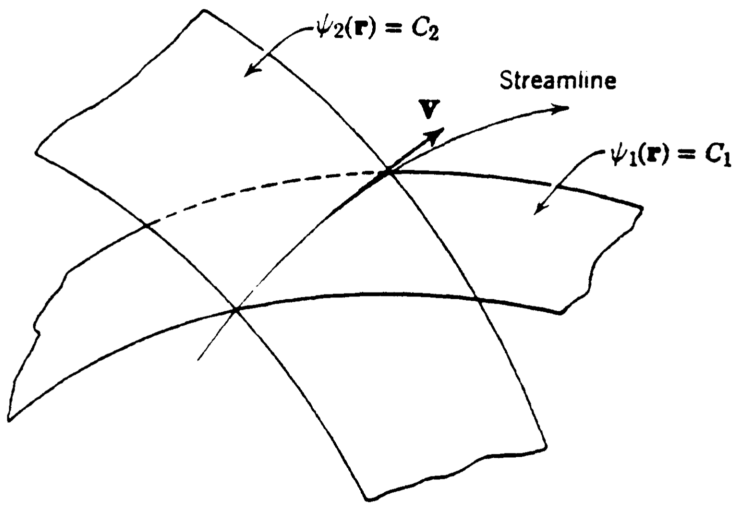


Figure 2.1: Stream functions and streamline. From Fig. 4.4 in [38].

Figure 2.1 shows the surfaces of constant $\psi_1$ and $\psi_2$ which are stream surfaces. The intersection of these two surfaces represents a streamline.

The use of two stream functions is very similar to the $S_1$ and $S_2$ surfaces described by Wu in [66]. These are shown in Fig. 2.2. The difference is that in Wu's approach, the grid coordinates are the dependent variables. These grid coordinates correspond to a stream surface such that

$$x = x(S_1, S_2) = x(\psi_1, \psi_2), \tag{2.5a}$$

$$y = y(S_1, S_2) = y(\psi_1, \psi_2), \tag{2.5b}$$

$$z = z(S_1, S_2) = z(\psi_1, \psi_2). \tag{2.5c}$$

For two dimensions, this is the approach taken in ISES [25] [27] [16], where

$$x = x(\psi_1), \tag{2.6a}$$

$$y = y(\psi_1). \tag{2.6b}$$

For the current approach, the grid is fixed[1] and the stream functions are calculated

$$\psi_1 = \psi_1(x, y, z), \tag{2.7a}$$

[1]The grid coordinates of the stagnation stream surfaces actually are calculated for reasons that will be described later.

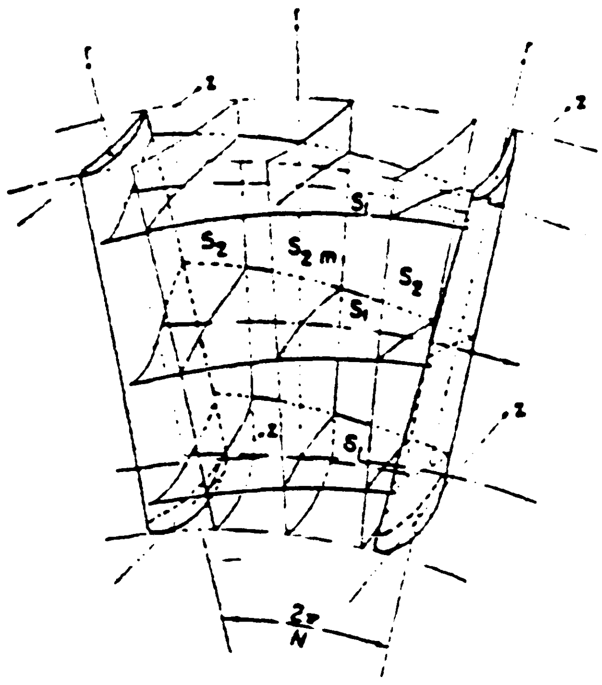


Figure 2.2: Stream surfaces in a blade passage. From Fig. 1 in [66].

$$\psi_2 = \psi_2(x, y, z). \tag{2.7b}$$

The two approaches are very similar, and each one has advantages and disadvantages. With the $S_1$, $S_2$ approach, separated viscous flows are not possible or are very difficult because the stream surface location becomes multi-valued. However, there is never any mass or momentum flux across the $S_1$, $S_2$ surfaces. This can simplify the corresponding calculations and lead to an algorithm with less numerical diffusion.

In both methods, the amount of grid resolution required to accurately solve the flowfield is related to the gradients and curvature of the $S_1$, $S_2$ surfaces or $\psi_1$, and $\psi_2$. In 2D, this is related to pressure gradients, and other important flow variables. However, in 3D, this is also related to the amount of secondary flow. The stream surfaces can warp dramatically in the streamwise direction due to streamwise vorticity while the flowfield and pressure field can remain essentially unchanged. This will be discussed further in the results section.

The $\psi_1$, $\psi_2$ approach has been chosen in this thesis because extensions of the formulation to separated flow is desired, at least for 2D flow.

The mass flux through an area can be related directly to the integral

$$\dot{m} = \iint d\psi_1 d\psi_2 \tag{2.8}$$

as derived in [24]. For the sake of completeness, it is re-derived here in a different way.

Given a 3D space $\psi_1$, $\psi_2$, $\psi_3$ for which there is a mapping from physical $x$, $y$, $z$ space, the differential volume in this space is

$$\iiint_{\mathcal{V}_\psi} d\psi_1 \, d\psi_2 \, d\psi_3 = \iiint_{\mathcal{V}} (\nabla\psi_1 \times \nabla\psi_2) \cdot \nabla\psi_3 dx \, dy \, dz. \tag{2.9}$$

If $\psi_3 \equiv l$ is the arc length from a surface $s$ and the volume $\mathcal{V}_\psi$ is made up of two identical parallel faces $\varepsilon$ apart, then

$$\iiint_{\mathcal{V}_\psi} d\psi_1 \, d\psi_2 \, d\psi_3 = \varepsilon \iint_{s_\psi} d\psi_1 \, d\psi_2. \tag{2.10}$$

In the physical domain, the volume $\mathcal{V}$ also has two identical faces $\varepsilon$ apart, so that

$$\iiint_{\mathcal{V}} (\nabla\psi_1 \times \nabla\psi_2) \cdot \nabla\psi_3 dx \, dy \, dz = \varepsilon \iint_s (\nabla\psi_1 \times \nabla\psi_2) \cdot \bar{n} ds, \tag{2.11}$$

where $\bar{n}$ is the outward unit normal of surface $s$. Therefore

$$\dot{m} = \iint_s \rho\overline{W} \cdot \bar{n} ds = \iint_s (\nabla\psi_1 \times \nabla\psi_2) \cdot \bar{n} ds = \iint_{s_\psi} d\psi_1 \, d\psi_2. \tag{2.12}$$

The two stream functions can be used to reduce the number of dependent variables by one. The three components of the mass flux vector can be related to two scalar fields. Also, applying any convective equation just requires tracking $\psi_1$ and $\psi_2$. The convected quantity is constant along a streamline where $\psi_1$ and $\psi_2$ are constant.

The stream function values which satisfy the equations of motion and flowfield boundary conditions are not unique. A unique solution is obtained only by appropriately specifying the boundary conditions. The levels of $\psi_1$ and $\psi_2$ are arbitrary and must be set. At the exit, either $\psi_1$ or $\psi_2$ can be specified arbitrarily, and the solution is determined completely. There are, however, distributions of $\psi_1$ or $\psi_2$ downstream which are better than others and introduce smaller numerical errors.

## 2.2 Momentum Equation

The integral form of the momentum equation in a rotating reference frame for the control volume $\mathcal{V}$ bounded by surface s is

$$\iiint_{\mathcal{V}} \frac{\partial\left(\rho\overline{W}\right)}{\partial t} d\mathcal{V} + \iint_s \left(\rho\overline{W}\left(\overline{W}\cdot\overline{n}\right) + P\overline{n}\right) ds = -\iiint_{\mathcal{V}} \rho\overline{S} d\mathcal{V} + \iint_s \overline{n}\cdot\varsigma' ds + \iiint_{\mathcal{V}} \overline{F}_b d\mathcal{V} \tag{2.13}$$

which is equation (5.50) in [22]. $\overline{W}$ is again the relative velocity vector, $\overline{n}$ is the outward unit normal, $\varsigma'$ is the viscous stress tensor, $\overline{S}$ represents a source term due to Coriolis and centrifugal terms, and $\overline{F}_b$ is the body force. The rotational source term for a coordinate system moving at a constant angular velocity $\overline{\Omega}$ is

$$\overline{S} = \overline{\Omega} \times \left(\overline{\Omega} \times \overline{r}\right) + 2\overline{\Omega} \times \overline{W}. \tag{2.14}$$

For steady inviscid flow with no body forces other than the Coriolis and centrifugal terms, Eq. (2.13) becomes

$$\iint_{s_\psi} \overline{W} d\psi_1 d\psi_2 + \iint_s P\overline{n} ds = -\iiint_{\mathcal{V}} \rho\overline{S} d\mathcal{V} \tag{2.15}$$

after substituting Eq. (2.8). The unsteady term need only be dropped for compressible flow. The viscous force term has been dropped only because the inviscid formulation is all that has been investigated in this thesis. The use of the two stream functions does not preclude viscosity.

If the relative coordinate system is oriented such that the $z$ axis is the axis of rotation, then

$$\overline{\Omega} = \omega\hat{k}, \tag{2.16}$$

and the components of velocity are

$$\overline{W} = W_x\hat{\imath} + W_y\hat{\jmath} + W_z\hat{k} \tag{2.17}$$

where $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$ are the unit vectors in the $x$, $y$, and $z$ directions. The position vector is then

$$\overline{r} = x\hat{\imath} + y\hat{\jmath} + z\hat{k}. \tag{2.18}$$

By applying Eqs. (2.16), (2.17), and (2.18) to Eq. (2.14),

$$\overline{S} = \left(-\omega^2 x - 2\omega W_y\right)\hat{\imath} + \left(-\omega^2 y + 2\omega W_x\right)\hat{\jmath}. \tag{2.19}$$

The $x$ momentum equation therefore becomes

$$\iint_{s_\psi} W_x d\psi_1 d\psi_2 + \iint_s P n_x ds = -\iiint_{\mathcal{V}} \left(-\rho\omega^2 x - 2\omega\rho W_y\right) d\mathcal{V}. \tag{2.20}$$

The $y$ momentum equation is

$$\iint_{s_\psi} W_y d\psi_1 d\psi_2 + \iint_s P n_y ds = -\iiint_{\mathcal{V}} \left(-\rho\omega^2 y + 2\omega\rho W_x\right) d\mathcal{V}. \tag{2.21}$$

And the $z$ momentum equation is

$$\iint_{s_\psi} W_z d\psi_1 d\psi_2 + \iint_s P n_z ds = 0, \tag{2.22}$$

where $n_x$, $n_y$ and $n_z$ are the $x$, $y$ and $z$ components of the outward unit normal.

### 2.2.1 Energy Equation and the Equation of State

For incompressible flow,

$$\rho = \text{constant}. \tag{2.23}$$

The energy equation is uncoupled from the momentum and continuity equations and therefore is not applied.

For compressible flow, a calorically perfect gas satisfies

$$h = C_p T, \tag{2.24}$$

$$P = \rho R T, \tag{2.25}$$

$$\gamma = \frac{C_p}{C_v}, \tag{2.26}$$

$$C_p - C_v = R. \tag{2.27}$$

The ideal gas law is applied

$$\rho = \frac{\gamma}{\gamma - 1}\frac{P}{h}, \tag{2.28}$$

where the enthalpy is determined by applying the energy equation.

In reference [66], Wu defines rothalpy as

$$I = h + \frac{W^2}{2} - \frac{(\omega r)^2}{2}, \tag{2.29}$$

where r is the radius about the z axis

$$r^2 = x^2 + y^2. \tag{2.30}$$

For steady, inviscid, non heat-conducting flows, rothalpy satisfies

$$\overline{W} \cdot \nabla I = 0. \tag{2.31}$$

Therefore, $I$ is conserved along a streamline in the relative frame. For the present formulation, a distribution of $I$ upstream can be defined in terms of $\psi_1$ and $\psi_2$ upstream. Then

$$I = I(\psi_1, \psi_2)|_{upstream}. \tag{2.32}$$

Once $I$ is known, the enthalpy can be calculated

$$h = I - \frac{W^2}{2} + \frac{(\omega r)^2}{2}, \tag{2.33}$$

and the ideal gas law becomes

$$\rho = \frac{\gamma}{(\gamma - 1)} \frac{P}{\left(I - \frac{W^2}{2} + \frac{(\omega r)^2}{2}\right)}. \tag{2.34}$$

The relative velocity is determined from Eq. (2.4), and

$$W^2 = \frac{(\rho W)^2}{\rho^2}. \tag{2.35}$$

The equation for density given the pressure, rothalpy, radius and $(\rho W)^2$ satisfies the quadratic equation

$$(\gamma - 1)\left(I + \frac{(\omega r)^2}{2}\right)\rho^2 - \gamma P \rho - \frac{(\gamma - 1)}{2}(\rho W)^2 = 0. \tag{2.36}$$

Eq. (2.36) is solved for $\rho$:

$$\rho = \frac{\gamma P + \sqrt{(\gamma P)^2 + 2(\gamma - 1)^2 \left(I + \frac{(\omega r)^2}{2}\right)(\rho W)^2}}{2(\gamma - 1)\left(I + \frac{(\omega r)^2}{2}\right)}. \tag{2.37}$$

Only the positive root is valid because density is positive.

The relative Mach number is defined

$$M^2 = \frac{W^2}{c^2} = \frac{W^2}{\gamma RT} = \frac{C_p W^2}{\gamma Rh} \tag{2.38}$$

$$M^2 = \frac{W^2}{(\gamma - 1)h}. \tag{2.39}$$

### 2.2.2 Use of the Munk and Prim Substitution Principle

For the angular velocity $\omega = 0$, the rothalpy is the total enthalpy $H$. In this case, the Munk and Prim substitution principle (see [29]) can be applied. This principle states that if a steady, isentropic flowfield is determined for a specified geometry and total pressure distribution, then the streamline shapes, Mach number, and the static and total pressures will be the same for any total temperature distribution. This principle is valid for steady, adiabatic, inviscid flow of a perfect gas with constant specific heats. Also, there must be no body forces (i.e. non-rotating). It can be applied by calculating the solution for a given geometry, back pressure, and total pressure inlet boundary condition and with a constant total enthalpy flowfield. Any given total enthalpy distribution can then be applied by convecting $H$ along a streamline and using the Mach number, static pressure, normalized velocity vector and total pressure distribution to get the velocities, densities and flow rates. The energy equation is effectively uncoupled from the other equations of motion.

## 2.3 Solid Wall Boundary Conditions

Along a solid wall, the mass flux

$$\dot{m} = 0. \tag{2.40}$$

This implies the momentum flux is also zero at a solid wall. For inviscid flow, no other wall boundary condition needs to be applied.

There are several ways to implement this boundary condition with the $\psi_1$, $\psi_2$ approach. Either $\psi_1$ or $\psi_2$ can be set to a constant, or Eq. (2.8) can be applied. For the current turbomachinery application, $\psi_1$ is set to a constant on each blade surface, and $\psi_2$ is set to a constant on the hub and casing.

## 2.4 Upstream Boundary Conditions

The number of boundary conditions applied on the upstream boundary depends on the number of incoming and outgoing characteristics (see [26]). This depends on the Mach number component normal to the inlet. For turbomachinery, this is the axial Mach number. Therefore, for

$$M_{normal} > 1, \tag{2.41}$$

everything can be specified at the inlet. For

$$M_{normal} < 1, \tag{2.42}$$

one condition must float at the inlet.

In this thesis, only the subsonic normal boundary condition has been implemented. In all cases, an entropy variable is specified at the inlet. For incompressible flow, this entropy variable is the rotary total pressure or total rotating pressure as described in [33], [17] and [37]. This is similar to the total pressure, except it is brought to rest in the relative frame and brought to zero radius. It is

$$\hat{s} = P + \frac{\rho W^2}{2} - \frac{\rho\,(\omega r)^2}{2}, \tag{2.43}$$

which is conserved along a streamline in the relative frame for inviscid flow.

For compressible flow, the entropy variable comes from the thermodynamic relation

$$T ds = dh - \frac{dP}{\rho}. \tag{2.44}$$

Combining Equations (2.44) and (2.24) through (2.27):

$$ds \;=\; C_p \frac{dh}{h} - R\frac{dP}{P}, \tag{2.45a}$$

$$\frac{ds}{R} = \frac{C_p}{R}\frac{dh}{h} - \frac{dP}{P}, \tag{2.45b}$$

$$\frac{ds}{R} = \frac{\gamma}{\gamma - 1} d(\ln h) - d(\ln P). \tag{2.45c}$$

Integrating between some reference point such that

$$s = 0 \text{ at } P_{ref} \text{ and } h_{ref}, \tag{2.46}$$

then

$$\frac{s}{R} = \frac{\gamma}{\gamma - 1}\ln\left(\frac{h}{h_{ref}}\right) - \ln\left(\frac{P}{P_{ref}}\right). \tag{2.47}$$

Defining the entropy variable for compressible flow as

$$\hat{s} = \exp(-s/R), \tag{2.48}$$

then

$$\hat{s} = \frac{P}{P_{ref}}\left(\frac{h}{h_{ref}}\right)^{\frac{-\gamma}{\gamma-1}}, \tag{2.49}$$

where $h$ is determined from Eq. (2.33). This is also conserved along a streamline in the relative frame in the absence of viscosity and shocks.

For compressible flow, the rothalpy is defined as the inlet boundary condition for the energy equation.

The flow angles in two directions are applied to the inlet boundary. For the two stream function approach, this means that the angle of the stream surface can be specified and that either $\Delta\psi_1$ is zero or $\Delta\psi_2$ is zero in the streamwise direction. The inlet boundary must be placed far enough away from a leading edge so a uniform flow angle is realistic.

For a rotor, the tangential flow angle is not necessarily known, but the absolute tangential velocity is. Therefore, one angle is specified and the other is determined as part of the solution for a given tangential velocity distribution.

## 2.5 Downstream Boundary Conditions

For a subsonic normal Mach number at the exit plane, there is one physical boundary condition which can be specified. Due to the non-uniqueness of the $\psi_1$, $\psi_2$ solution, the distribution of either of these must be specified. Therefore, specifying both $\psi_1$ and $\psi_2$ downstream allows both the physical and arbitrary boundary conditions to be implemented. Physically this can be interpreted as specifying the flow rate distribution downstream. This is often difficult to specify, especially for rotational or choked flows.

Another option is to specify the distribution of either $\psi_1$ or $\psi_2$ and to extrapolate the pressure gradients.

## 2.6 Periodic Boundary Condition

Only one passage is being analyzed by this algorithm. It is assumed that each passage has the same flowfield as every other. Therefore, periodicity must be enforced outside of the blade region. However, in 3D, a wake is shed from the trailing edge, and except for a small class of geometries, the shed vorticity is not uniform spanwise. Under the inviscid assumption, this wake can support a shear layer and velocity vectors which are in different directions. The static pressure, however, must be equal across this wake. The easiest way to accommodate this wake is to follow it. Therefore, the position of the wake will be determined and $\psi_1$ will be a constant on it. Upstream, the stagnation stream surface will also be determined. The stagnation line along the leading edge will be determined by a pressure matching condition similar to that used in ISES. The $S_1$ surface corresponding to the wake and stagnation stream surface therefore is determined.

Periodicity is applied by setting the pressures to be the same across the periodic boundary. The location of the stagnation stream surface, or wake, of one blade is translated by the blade pitch to the other blade.

## 2.7 Design Option

The boundary conditions described so far are for analyzing a flowfield for a given geometry. When designing a blade, some type of flowfield description is generally desired and the geometry is unknown. The blade can be "inverse designed" such that the geometry is determined from the given flowfield condition.

A design option where the pressure loading and blade thickness are specified can be applied using the framework of the periodic boundary condition. The camber line will be the result.

The blade surface can be treated just like the wake, except that rather than specifying a zero pressure jump, a finite pressure jump is specified which corresponds to the blade loading. Also, the blade thickness must be accounted for when translating to the other boundary.

## 2.8 Nondimensionalization

The equations presented are applicable for either dimensional or dimensionless quantities. Using dimensionless quantities has several advantages:

1. unit conversion factors are not required;
2. scaling laws are generally implicit in the dimensionless quantities; and,
3. the errors in the equations are scaled so that a convergence tolerance is independent of units.

For certain applications, a suitable dimensionless system can be applied. If a dimensional definition of the application is used to describe the problem, then the system is nondimensionalized in the following way (a $\check{}$ implies a dimensional quantity):

$$\check{L}_n = \text{reference chord (hub)}. \tag{2.50}$$

$$\check{V}_n = \text{reference velocity} = \sqrt{2\check{I}}\ \text{(hub inlet)}. \tag{2.51}$$

$$\check{P}_n = \text{reference pressure} = \check{P}_T\text{(hub inlet)}. \tag{2.52}$$

Geometry and flow variables are normalized as follows:

$$P = \frac{\check{P}}{\check{P}_n}; \quad V = \frac{\check{V}}{\check{V}_n}; \quad r = \frac{\check{r}}{\check{L}_n};$$
$$z = \frac{\check{z}}{\check{L}_n}; \quad I = \frac{\check{I}}{\check{V}_n^2}; \quad \omega = \frac{\check{\omega}\check{L}_n}{\check{V}_n};$$
$$h = \frac{\check{h}}{\check{V}_n^2}; \quad \rho = \frac{\check{\rho}\check{V}_n^2}{\check{P}_n}; \quad \dot{m} = \check{\dot{m}}\frac{\check{V}_n}{\check{P}_n\check{L}_n^2}.$$

# Chapter 3
# Discretized Finite Volume Equations

## 3.1 Coordinate Systems

The algorithm described in this thesis has been developed for turbomachinery applications. The flowfields are solved in a relative coordinate system which rotates with a rotor and is stationary for a stator. For many components, a relative cylindrical coordinate system is useful. Periodicity can be applied easily. Also, boundary conditions are easily applied along a constant radius. The relative Cartesian coordinate is also useful. The unit vectors, and therefore the $x$, $y$ and $z$ momentum fluxes in this system, do not change direction. These equations are more straightforward since they do not contain any added curvature terms. The relation between the Cartesian and cylindrical system is the same as that used at GE, and is shown in Fig. 3.1.

The relations are not standard, but allow both $r$-$z$ and $\theta$-$z$ to be viewed conveniently:

$$x = r\sin\theta, \tag{3.1}$$

$$y = r\cos\theta, \tag{3.2}$$

$$r = \sqrt{x^2 + y^2}, \tag{3.3}$$

$$\theta = \arctan\left(\frac{x}{y}\right), \tag{3.4}$$

$$\hat{e}_\theta = \cos\theta\hat{\imath} - \sin\theta\hat{\jmath}, \tag{3.5}$$

$$\hat{e}_r = \sin\theta\hat{\imath} + \cos\theta\hat{\jmath}. \tag{3.6}$$

The angular velocity of the coordinate system is about the $z$ axis. Rotation in positive $\theta$ produces a negative $\omega$. Both coordinate systems are right-handed if thought of as $(x, y, z)$ and $(\theta, r, z)$.

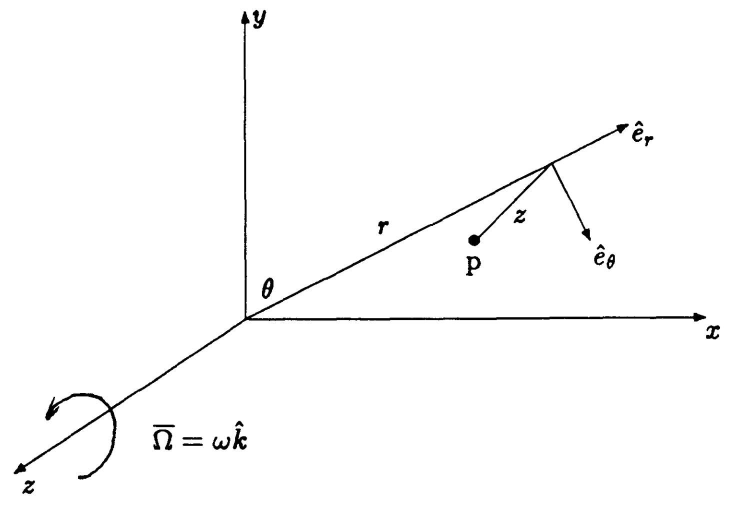


Figure 3.1: Cartesian and cylindrical coordinate systems.

Another useful coordinate system can be defined because the hub and tip are generally axisymmetic, and the blade geometry can be defined on surfaces of revolution. These surfaces can be defined by a relation:

$$r = r(z) \text{ or } z = z(r). \tag{3.7}$$

The coordinate, $m'$ is defined as

$$m' = \int \frac{dm}{r} = \int \frac{\sqrt{(dr)^2 + (dz)^2}}{r}, \tag{3.8}$$

and is a normalized arc length in the $r$-$z$ plane. The advantage is that $m'$-$\theta$ represents a conformal mapping of the surface of revolution so angles are preserved. Both $m'$ and $\theta$ are dimensionless (radians). They are similar to the $x^*$, $y^*$ coordinates discussed in reference [62].

Another advantage of $m'$-$\theta$ is that the geometry can be considered as strips of 2D geometry defined as lines rather than 3D geometry defined as surfaces. This is applied by defining

$$r = r(m') \text{ and } z = z(m'), \tag{3.9}$$

for every spanwise station of the geometry. Then the grid is also defined in $m'$-$\theta$ coordinates for each spanwise station. The movement of grid lines and the application of design option are facilitated using this approach. Also, because angles are preserved in $m'$-$\theta$, a 2D elliptic grid generator[1] can be applied for each spanwise station rather than using a true 3D grid generator. Because $m'$ is defined as an integral, the zero value is defined at the inlet for convenience.

The coordinates $(x^1, x^2, x^3, m')$ will be used in the code and in further discussions. For Cartesian coordinates:

$$x^1 = x, \tag{3.10a}$$

$$x^2 = y, \tag{3.10b}$$

$$x^3 = z, \tag{3.10c}$$

$$m' = \int \sqrt{(dy)^2 + (dz)^2}. \tag{3.10d}$$

And in cylindrical coordinates:

$$x^1 = \theta, \tag{3.11a}$$

$$x^2 = r, \tag{3.11b}$$

$$x^3 = z, \tag{3.11c}$$

$$m' = \int \frac{\sqrt{(dr)^2 + (dz)^2}}{r}. \tag{3.11d}$$

The Cartesian option is used for linear cascades, rectangular ducts and 2D blade-to-blade problems. For ducts, where there are no periodic boundaries, the grid is fixed and $m'$ is not needed.

The distinction between Cartesian and cylindrical coordinate systems will be made when the velocity calculation is discussed. Because this involves gradients, attention to discretization detail is important for accurate calculations.

The fourth coordinate system is the computational coordinate system: $\xi^1$, $\xi^2$ and $\xi^3$. For this application, $\xi^3$ varies predominately in the streamwise direction, $\xi^1$ blade-to-blade and $\xi^2$ hub-to-tip.

[1]A version of the ISES grid generator modified for multiple spanwise stations and described in [27] is actually used.

### 3.1.1 Generalized Coordinate Transformations

In this thesis, it will be necessary to transform between coordinate systems. This mainly is needed to calculate physical partial derivatives from derivatives in computational space. For example, an $x^1$ derivative is determined from the chain rule:

$$\frac{\partial}{\partial x^1} = \frac{\partial \xi^1}{\partial x^1}\frac{\partial}{\partial \xi^1} + \frac{\partial \xi^2}{\partial x^1}\frac{\partial}{\partial \xi^2} + \frac{\partial \xi^3}{\partial x^1}\frac{\partial}{\partial \xi^3}. \tag{3.12}$$

The inverse transform grid derivatives $\frac{\partial \xi^1}{\partial x^1}$, $\frac{\partial \xi^2}{\partial x^1}$ and $\frac{\partial \xi^3}{\partial x^1}$ are determined from the 9 actual grid derivatives such as $\frac{\partial x^1}{\partial \xi^1}$, $\frac{\partial x^2}{\partial \xi^1}$ and $\frac{\partial x^3}{\partial \xi^1}$. This is based on the ability to transform from one coordinate system to another

$$\hat{a}_i = \frac{\partial x^j}{\partial \xi^i}\hat{c}_j, \tag{3.13}$$

where $\hat{a}_i$ and $\hat{c}_j$ are the covariant base vectors of the computational and physical coordinate systems respectively (see [6], [61] and [3]). The inverse transformation is then

$$\hat{c}_j = \frac{\partial \xi^i}{\partial x^j}\hat{a}_i. \tag{3.14}$$

In terms of differentials, if $\frac{\partial x^i}{\partial \xi^j}$ is represented as the matrix $\mathbf{B}$, then

$$\begin{bmatrix} dx^1 \\ dx^2 \\ dx^3 \end{bmatrix} = \mathbf{B} \begin{bmatrix} d\xi^1 \\ d\xi^2 \\ d\xi^3 \end{bmatrix}, \tag{3.15}$$

and

$$\begin{bmatrix} d\xi^1 \\ d\xi^2 \\ d\xi^3 \end{bmatrix} = \mathbf{B}^{-1} \begin{bmatrix} dx^1 \\ dx^2 \\ dx^3 \end{bmatrix}. \tag{3.16}$$

The terms $\frac{\partial \xi^i}{\partial x^j}$ are determined by inverting the matrix representing the grid derivatives $\frac{\partial x^i}{\partial \xi^j}$. The Jacobian

$$J = \det(\mathbf{B}) = \det\left(\frac{\partial x^i}{\partial \xi^j}\right), \tag{3.17}$$

is used to calculate the inverse using Cramer's rule. It also physically represents the volume in physical space of a unit cube in computational space.

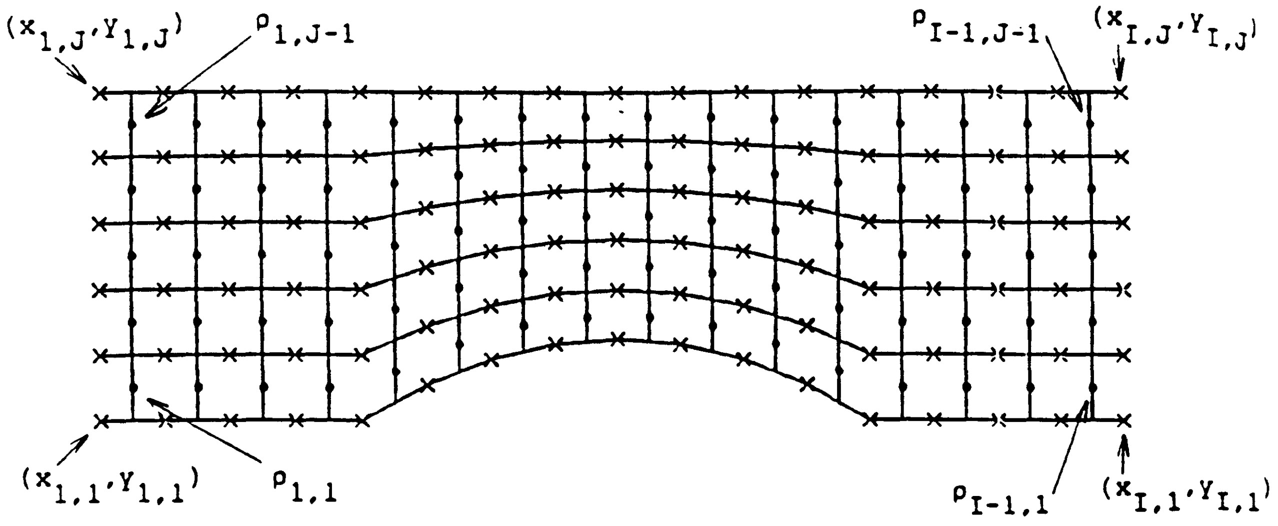


Figure 3.2: ISES grid showing location of unknowns $\rho$ at a • and (x,y) at a $\times$. From Giles in [27].

## 3.2 Velocity Calculation

The 3D discretization and velocity calculation is difficult to visualize. Therefore, the corresponding 2D approach will be described first along with how it relates to ISES in [27] and [16]. The approach will then be extended to 3D.

### 3.2.1 Two Dimensional Velocity Calculations

The grid for ISES in [27] is shown in Fig. 3.2. It shows the location of the unknowns $\rho$ and the streamline displacement which corresponds to the grid node $(x, y)$. The grid as drawn actually represents the edges of the control volumes.

For a 2D application of the current algorithm, the grid is very similar to the ISES grid. The grid is fixed for a duct and the stream function $\psi$ is stored at the $\times$, and pressure is stored at the •. The use of pressure rather than density is to allow for incompressible flows. The streamwise grid lines are not streamlines, so momentum fluxes need to be calculated across these surfaces. In 2D, Eq. (2.4) reduces to

$$\rho\overline{W} = \nabla\psi \times (\hat{k}) \tag{3.18}$$

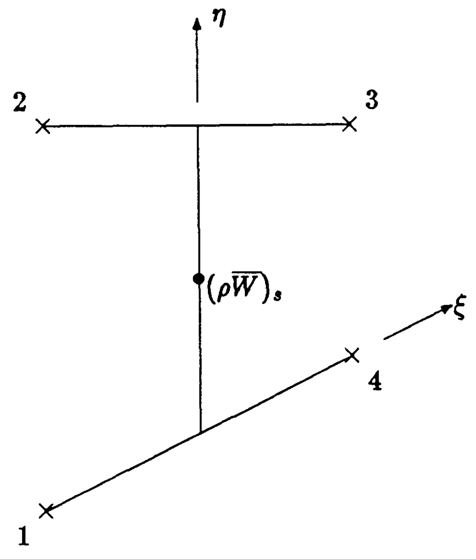


Figure 3.3: Stencil for an $S$ face in 2D.

where

$$\psi_1 = \psi(x,y) \text{ and } \psi_2 = z, \tag{3.19}$$

as shown in [38]. Therefore,

$$\rho\overline{W} = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ \frac{\partial\psi}{\partial x} & \frac{\partial\psi}{\partial y} & 0 \\ 0 & 0 & 1 \end{vmatrix}, \tag{3.20}$$

and

$$\rho W_x = \frac{\partial\psi}{\partial y}, \tag{3.21}$$

$$\rho W_y = -\frac{\partial\psi}{\partial x}, \tag{3.22}$$

which is the normal 2D application of stream functions for planar flow.

At the center of each control volume face the velocity is determined by calculating the derivatives $\frac{\partial\psi}{\partial x}$ and $\frac{\partial\psi}{\partial y}$. For a streamwise, or $S$, face as shown in Fig. 3.3, the derivatives in the computational coordinates are discretized

$$\left.\frac{\partial\psi}{\partial\xi}\right|_S = \frac{1}{2}\left[(\psi_3-\psi_2)+(\psi_4-\psi_1)\right], \tag{3.23a}$$

$$\left.\frac{\partial\psi}{\partial\eta}\right|_S = \frac{1}{2}\left[(\psi_2+\psi_3)-(\psi_1+\psi_4)\right]. \tag{3.23b}$$

The chain rule is then used to determine the derivatives which are desired:

$$\frac{\partial\psi}{\partial x} = \frac{\partial\psi}{\partial\eta}\frac{\partial\eta}{\partial x} + \frac{\partial\psi}{\partial\xi}\frac{\partial\xi}{\partial x}. \tag{3.24}$$

The inverse transformation grid derivatives $\frac{\partial\xi}{\partial x}$, $\frac{\partial\xi}{\partial y}$, $\frac{\partial\eta}{\partial x}$, $\frac{\partial\eta}{\partial y}$, are determined by calculating the actual grid derivatives $\frac{\partial x}{\partial\xi}$, $\frac{\partial y}{\partial\xi}$, $\frac{\partial x}{\partial\eta}$, $\frac{\partial y}{\partial\eta}$, and transforming them. For this 2D example, $\frac{\partial x^i}{\partial\xi^j}$ is represented by the matrix $\mathbf{B}$:

$$\mathbf{B} = \begin{pmatrix} \frac{\partial x}{\partial\xi} & \frac{\partial x}{\partial\eta} \\ \frac{\partial y}{\partial\xi} & \frac{\partial y}{\partial\eta} \end{pmatrix}. \tag{3.25}$$

The Jacobian

$$J = \frac{\partial x}{\partial\xi}\frac{\partial y}{\partial\eta} - \frac{\partial x}{\partial\eta}\frac{\partial y}{\partial\xi}. \tag{3.26}$$

The inverse is

$$\mathbf{B}^{-1} = \begin{pmatrix} \frac{\partial\xi}{\partial x} & \frac{\partial\xi}{\partial y} \\ \frac{\partial\eta}{\partial x} & \frac{\partial\eta}{\partial y} \end{pmatrix} = \frac{1}{J}\begin{pmatrix} \frac{\partial y}{\partial\eta} & -\frac{\partial y}{\partial\xi} \\ -\frac{\partial x}{\partial\eta} & \frac{\partial x}{\partial\xi} \end{pmatrix}. \tag{3.27}$$

If $\psi$ is linear in $x$ and $y$, then $\rho W_x$ and $\rho W_y$ are uniform. To ensure that uniform flow can be handled correctly, the grid derivatives $\frac{\partial x}{\partial\xi}$ and $\frac{\partial y}{\partial\xi}$ should be discretized the same as $\frac{\partial\psi}{\partial\xi}$. Likewise, $\frac{\partial x}{\partial\eta}$ and $\frac{\partial y}{\partial\eta}$ should be discretized the same as $\frac{\partial\psi}{\partial\eta}$. Therefore,

$$\left.\frac{\partial x}{\partial\xi}\right|_S = \frac{1}{2}\left[(x_3-x_2)+(x_4-x_1)\right], \tag{3.28a}$$

$$\left.\frac{\partial y}{\partial\xi}\right|_S = \frac{1}{2}\left[(y_3-y_2)+(y_4-y_1)\right], \tag{3.28b}$$

$$\left.\frac{\partial x}{\partial\eta}\right|_S = \frac{1}{2}\left[(x_2+x_3)-(x_1-x_4)\right], \tag{3.28c}$$

$$\left.\frac{\partial y}{\partial\eta}\right|_S = \frac{1}{2}\left[(y_2+y_3)-(y_1-y_4)\right]. \tag{3.28d}$$

The mass flux vector $\rho\overline{W}$ can now be computed at the $S$ face. It is a second order accurate calculation for each derivative because it is evaluated at the center of the four points where $\psi$ is stored.

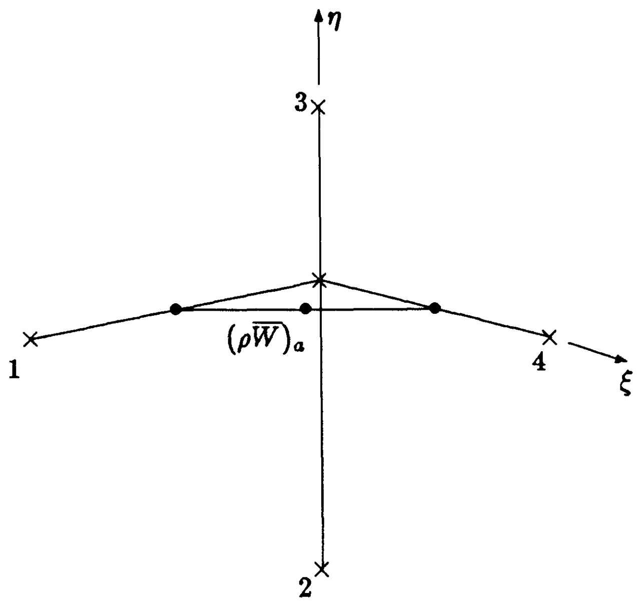


Figure 3.4: Stencil for an $A$ face in 2D.

The other face, which will be denoted as an $A$ face, is sketched in Fig. 3.4. The $\psi$ derivatives and grid derivatives are discretized in computational coordinates by central differencing:

$$\left.\frac{\partial\psi}{\partial\xi}\right|_A = \frac{\psi_4 - \psi_1}{2}, \tag{3.29a}$$

$$\left.\frac{\partial x}{\partial\xi}\right|_A = \frac{x_4 - x_1}{2}, \tag{3.29b}$$

$$\left.\frac{\partial y}{\partial\xi}\right|_A = \frac{y_4 - y_1}{2}, \tag{3.29c}$$

$$\left.\frac{\partial\psi}{\partial\eta}\right|_A = \frac{\psi_3 - \psi_2}{2}, \tag{3.29d}$$

$$\left.\frac{\partial x}{\partial\eta}\right|_A = \frac{x_3 - x_2}{2}, \tag{3.29e}$$

$$\left.\frac{\partial y}{\partial\eta}\right|_A = \frac{y_3 - y_2}{2}. \tag{3.29f}$$

These are only first order accurate formulas for non-uniform grids. However, it is assumed that the main flow is aligned in the $\xi$ direction so the total errors will be small because the mass flux across the $A$ face is small. The velocity is not needed at the walls, so no one-sided differencing is required.

### 3.2.2 Three Dimensional Velocity Calculation

For a 3D rectangular duct, the grid is shown in Fig. 3.5. The grid nodes correspond to the corners of the hexahedra shown in this figure. These hexahedra are not the control volumes. The control volumes are offset in the $\xi^3$ direction. An $S$ face which is one side of a control volume is also shown in Fig. 3.5. The location of $\psi_1$ and $\psi_2$ in the grid is shown. These do not correspond to the grid points, but are staggered. If the problem is reduced to 2D, such that $\psi_2 = 0$ on a lower wall and $\psi_2 = 1$ on an upper wall, the grid reduces to the ISES grid for $\psi_1$ and $P$.

In this algorithm, the $x$, $y$ and $z$ components of mass flux are needed. They must be determined for either a Cartesian coordinate system:

$$\begin{aligned}
\rho\overline{W} &= \rho W_x\hat{\imath} + \rho W_y\hat{\jmath} + \rho W_z\hat{k}, \\
&= \nabla\psi_1 \times \nabla\psi_2, \\
&= \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ \frac{\partial\psi_1}{\partial x} & \frac{\partial\psi_1}{\partial y} & \frac{\partial\psi_1}{\partial z} \\ \frac{\partial\psi_2}{\partial x} & \frac{\partial\psi_2}{\partial y} & \frac{\partial\psi_2}{\partial z} \end{vmatrix}, \\
&= \left(\frac{\partial\psi_1}{\partial y}\frac{\partial\psi_2}{\partial z} - \frac{\partial\psi_1}{\partial z}\frac{\partial\psi_2}{\partial y}\right)\hat{\imath} \\
&\quad + \left(\frac{\partial\psi_1}{\partial z}\frac{\partial\psi_2}{\partial x} - \frac{\partial\psi_1}{\partial x}\frac{\partial\psi_2}{\partial z}\right)\hat{\jmath} \\
&\quad + \left(\frac{\partial\psi_1}{\partial x}\frac{\partial\psi_2}{\partial y} - \frac{\partial\psi_1}{\partial y}\frac{\partial\psi_2}{\partial x}\right)\hat{k},
\end{aligned} \tag{3.30}$$

or a cylindrical coordinate system:

$$\begin{aligned}
\rho\overline{W} &= \rho W_\theta\hat{e}_\theta + \rho W_r\hat{e}_r + \rho W_z\hat{k}, \\
&= \nabla\psi_1 \times \nabla\psi_2, \\
&= \begin{vmatrix} \hat{e}_\theta & \hat{e}_r & \hat{k} \\ \frac{1}{r}\frac{\partial\psi_1}{\partial \theta} & \frac{\partial\psi_1}{\partial r} & \frac{\partial\psi_1}{\partial z} \\ \frac{1}{r}\frac{\partial\psi_2}{\partial \theta} & \frac{\partial\psi_2}{\partial r} & \frac{\partial\psi_2}{\partial z} \end{vmatrix}, \\
&= \left(\frac{\partial\psi_1}{\partial r}\frac{\partial\psi_2}{\partial z} - \frac{\partial\psi_1}{\partial z}\frac{\partial\psi_2}{\partial r}\right)\hat{e}_\theta
\end{aligned}$$

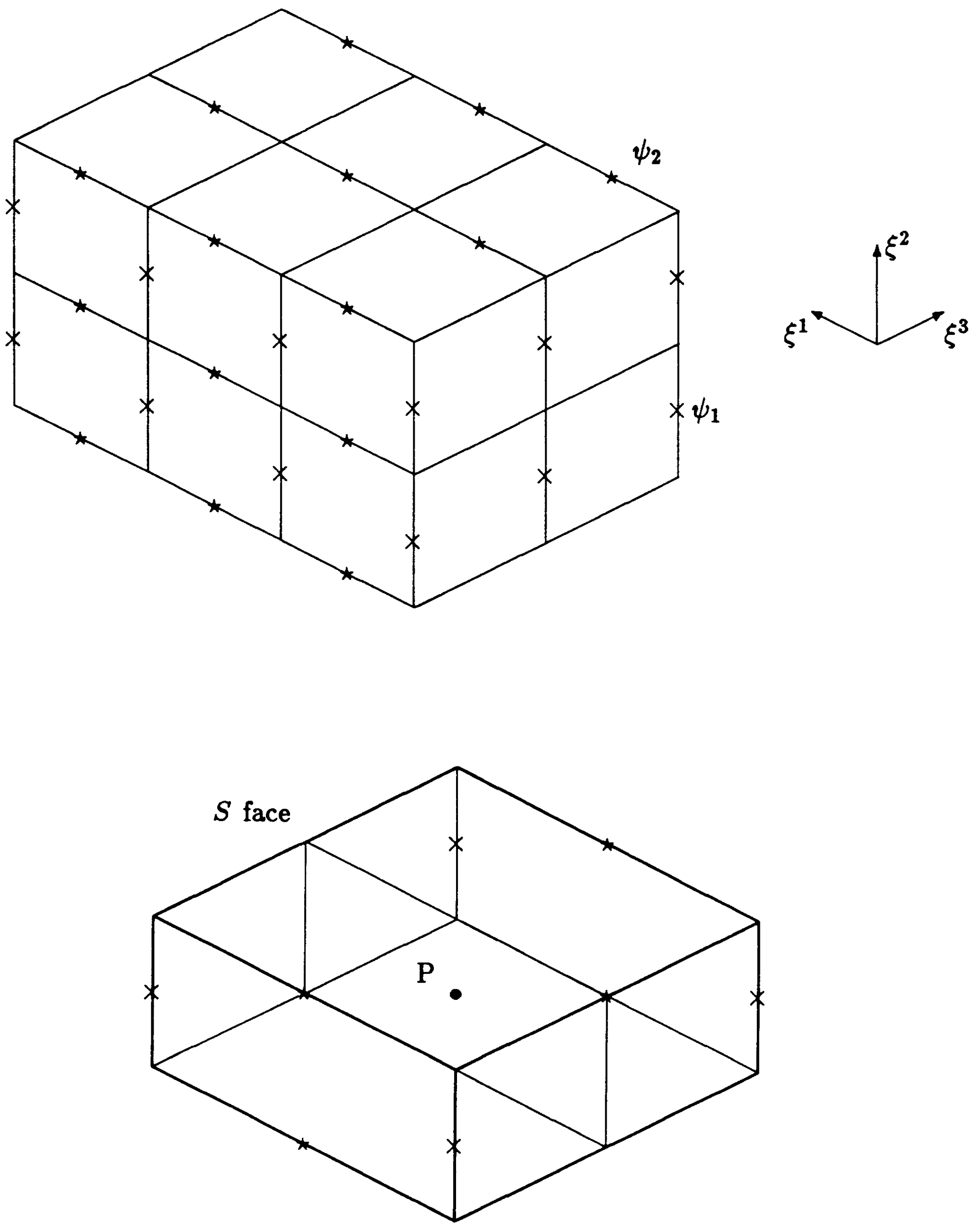


Figure 3.5: A hidden line view of the grid in 3D and one cell showing an $S$ face.

$$+\frac{1}{r}\left(\frac{\partial\psi_1}{\partial z}\frac{\partial\psi_2}{\partial\theta}-\frac{\partial\psi_1}{\partial\theta}\frac{\partial\psi_2}{\partial z}\right)\hat{e}_r$$

$$+\frac{1}{r}\left(\frac{\partial\psi_1}{\partial\theta}\frac{\partial\psi_2}{\partial r}-\frac{\partial\psi_1}{\partial r}\frac{\partial\psi_2}{\partial\theta}\right)\hat{k}. \tag{3.31}$$

From Eqs. (3.1) and (3.6),

$$\rho W_x = \rho W_r\frac{x}{r}+\rho W_\theta\frac{y}{r}, \tag{3.32}$$

and

$$\rho W_y = \rho W_r\frac{y}{r}-\rho W_\theta\frac{x}{r}. \tag{3.33}$$

The $x^1$, $x^2$, $x^3$ $(x,y,z$ or $\theta,r,z)$ derivatives of $\psi_1$ and $\psi_2$ are calculated using the chain rule. For example,

$$\frac{\partial\psi_1}{\partial x^1}=\frac{\partial\psi_1}{\partial\xi^1}\frac{\partial\xi^1}{\partial x^1}+\frac{\partial\psi_1}{\partial\xi^2}\frac{\partial\xi^2}{\partial x^1}+\frac{\partial\psi_1}{\partial\xi^3}\frac{\partial\xi^3}{\partial x^1}. \tag{3.34}$$

The inverse grid transformation derivatives are determined by inverting the matrix of 3D grid derivatives:

$$\mathbf{B}=\begin{pmatrix}\frac{\partial x^1}{\partial\xi^1} & \frac{\partial x^1}{\partial\xi^2} & \frac{\partial x^1}{\partial\xi^3}\\ \frac{\partial x^2}{\partial\xi^1} & \frac{\partial x^2}{\partial\xi^2} & \frac{\partial x^2}{\partial\xi^3}\\ \frac{\partial x^3}{\partial\xi^1} & \frac{\partial x^3}{\partial\xi^2} & \frac{\partial x^3}{\partial\xi^3}\end{pmatrix}=\begin{pmatrix}b_1 & b_2 & b_3\\ b_4 & b_5 & b_6\\ b_7 & b_8 & b_9\end{pmatrix}, \tag{3.35}$$

$$J = \det\mathbf{B} \tag{3.36}$$

$$= b_1b_5b_9+b_2b_6b_7+b_4b_8b_3-b_3b_5b_7-b_1b_6b_8-b_2b_4b_9. \tag{3.37}$$

$$\mathbf{B}^{-1}=\begin{pmatrix}\frac{\partial\xi^1}{\partial x^1} & \frac{\partial\xi^1}{\partial x^2} & \frac{\partial\xi^1}{\partial x^3}\\ \frac{\partial\xi^2}{\partial x^1} & \frac{\partial\xi^2}{\partial x^2} & \frac{\partial\xi^2}{\partial x^3}\\ \frac{\partial\xi^3}{\partial x^1} & \frac{\partial\xi^3}{\partial x^2} & \frac{\partial\xi^3}{\partial x^3}\end{pmatrix}, \tag{3.38}$$

$$\frac{\partial\xi^i}{\partial x^j}=\frac{\text{cofactor of }\mathbf{B}_{ij}\text{ in }J}{J}, \tag{3.39}$$

$$\frac{\partial\xi^1}{\partial x^1} = \frac{1}{J}\left(b_5b_9-b_6b_8\right), \tag{3.40a}$$

$$\frac{\partial\xi^1}{\partial x^2} = -\frac{1}{J}\left(b_4b_9-b_6b_7\right), \tag{3.40b}$$

$$\frac{\partial \xi^1}{\partial x^3} = \frac{1}{J}(b_4 b_8 - b_5 b_7), \quad (3.40c)$$

$$\frac{\partial \xi^2}{\partial x^1} = -\frac{1}{J}(b_2 b_9 - b_3 b_8), \quad (3.40d)$$

$$\frac{\partial \xi^2}{\partial x^2} = \frac{1}{J}(b_1 b_9 - b_3 b_7), \quad (3.40e)$$

$$\frac{\partial \xi^2}{\partial x^3} = -\frac{1}{J}(b_1 b_8 - b_2 b_7), \quad (3.40f)$$

$$\frac{\partial \xi^3}{\partial x^1} = \frac{1}{J}(b_2 b_6 - b_3 b_5), \quad (3.40g)$$

$$\frac{\partial \xi^3}{\partial x^2} = -\frac{1}{J}(b_1 b_6 - b_3 b_4), \quad (3.40h)$$

$$\frac{\partial \xi^3}{\partial x^3} = \frac{1}{J}(b_1 b_5 - b_2 b_4). \quad (3.40i)$$

A stencil for an interior $S$ face is shown in Fig. 3.6. Based on this stencil, the stream function derivatives are discretized:

$$\left.\frac{\partial \psi_1}{\partial \xi^1}\right|_S = \frac{1}{2}\left[(\psi_1|_4 + \psi_1|_{10}) - (\psi_1|_3 + \psi_1|_9)\right], \quad (3.41a)$$

$$\left.\frac{\partial \psi_1}{\partial \xi^2}\right|_S = \frac{1}{8}\left[(\psi_1|_5 + \psi_1|_6 + \psi_1|_{11} + \psi_1|_{12}) - (\psi_1|_1 + \psi_1|_2 + \psi_1|_7 + \psi_1|_8)\right], (3.41b)$$

$$\left.\frac{\partial \psi_1}{\partial \xi^3}\right|_S = \frac{1}{2}\left[(\psi_1|_9 + \psi_1|_{10}) - (\psi_1|_3 + \psi_1|_4)\right], \quad (3.41c)$$

$$\left.\frac{\partial \psi_2}{\partial \xi^1}\right|_S = \frac{1}{8}\left[(\psi_2|_3 + \psi_2|_6 + \psi_2|_9 + \psi_2|_{12}) - (\psi_2|_1 + \psi_2|_4 + \psi_2|_7 + \psi_2|_{10})\right], (3.42a)$$

$$\left.\frac{\partial \psi_2}{\partial \xi^2}\right|_S = \frac{1}{2}\left[(\psi_2|_5 + \psi_2|_{11}) - (\psi_2|_2 + \psi_2|_8)\right], \quad (3.42b)$$

$$\left.\frac{\partial \psi_2}{\partial \xi^3}\right|_S = \frac{1}{2}\left[(\psi_2|_8 + \psi_2|_{11}) - (\psi_2|_2 + \psi_2|_5)\right]. \quad (3.42c)$$

The grid derivatives are evaluated using identical stencils as the $\psi_1$ and $\psi_2$ stencils. This means that $\frac{\partial x^1}{\partial \xi^1}$ is evaluated for both the $\psi_1$ derivatives and $\psi_2$ derivatives. Again this is done to ensure a uniform velocity is calculated when the stream function is linear. The coordinates are averaged from the grid point locations to the $\psi_1$ or $\psi_2$ locations.

Nine other stencils exist for the $S$ face to accommodate corners and 2D flows. The 2D stencil is sketched in Appendix D.

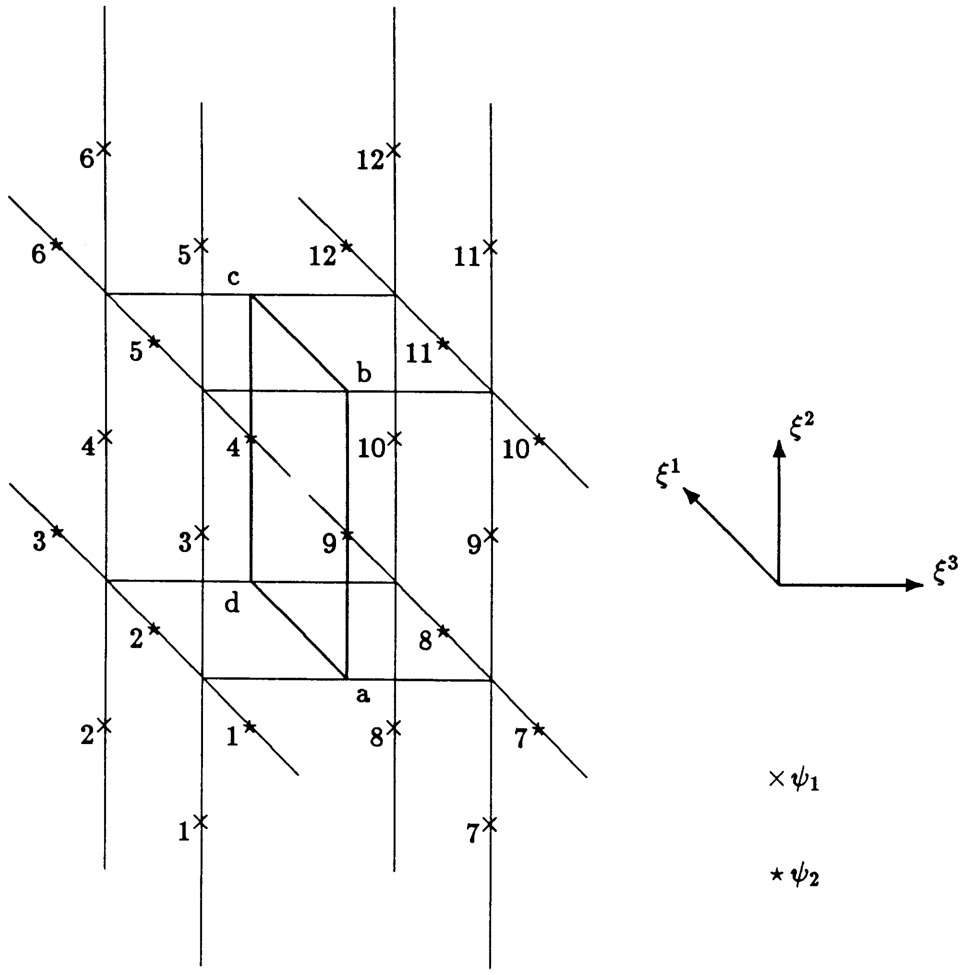


Figure 3.6: Stencil for $\psi_1$ and $\psi_2$ on an interior $S$ face (type 0).

The $S$ face is the contravariant area projection in the $\xi^3$ direction. The $A$ face in 3D is the projection in the $\xi^1$ direction, and the $B$ face is the projection in the $\xi^2$ direction. The $A$ and $B$ faces are similar to the $S$ face just as the $A$ face was similar to the $S$ face in 2D. The primary stencils and discretized equations are also in Appendix D.

Once $\rho\overline{W}$ is known, $\rho$ can be calculated from Eq. (2.37) and the relative velocity vector is known. The geometrical terms are averaged appropriately and the rothalpy is obtained by a convection equation which is discussed in Section 3.7.

From the stencil shown in Fig. 3.6 and the discrete form of the derivatives, it can be seen that there is a preferred direction of the differencing. It is assumed that $\psi_1$ varies mainly in the $\xi^1$ direction and that $\psi_2$ varies mainly in the $\xi^2$ direction. The staggering of the stream functions was necessary to create a system which has enough equations for the unknowns. It also reduces to the 2D discretization if $ni = 2$ or $nj = 2$. Chapter 5 contains a discussion on the way the momentum equations are combined. These equation combinations produce a second difference operator (1 -2 1) of $\psi_1$ and $\psi_2$ in the $\xi^3$ direction. However, the $\psi_1$ second difference operator is basically multiplied by $\frac{\partial\psi_2}{\partial\xi^2}$, and the $\psi_2$ operator is multiplied by $\frac{\partial\psi_1}{\partial\xi^1}$. As the stream surfaces rotate through 90 degrees, these derivatives go to zero and the system of equations becomes ill-conditioned. This is discussed further in Section 7.1.

## 3.3 Mass Flux

The mass flux across a face is needed in the momentum equation because the momentum flux component is determined by multiplying the mass flux by the velocity component. The mass flux through this face is

$$\dot{m} = \iint d\psi_1 d\psi_2. \tag{3.43}$$

In computational space, this face is the square shown in Fig. 3.7. A mapping of this face to the $\psi_1$, $\psi_2$ space might look like Fig. 3.8 where linear distributions are used. The signed area of the quadrilateral in $\psi_1$, $\psi_2$ space is the sum of the signed areas of

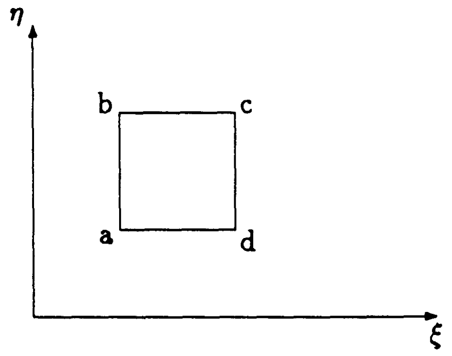


Figure 3.7: Face of a control volume in computational space. For positive $\dot{m}$, the flow is out of the paper.

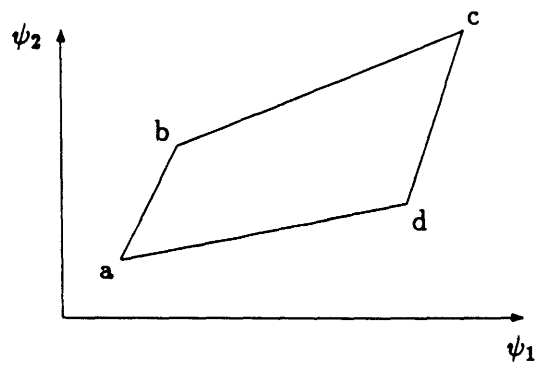


Figure 3.8: Face of a control volume in $\psi_1$, $\psi_2$ space.

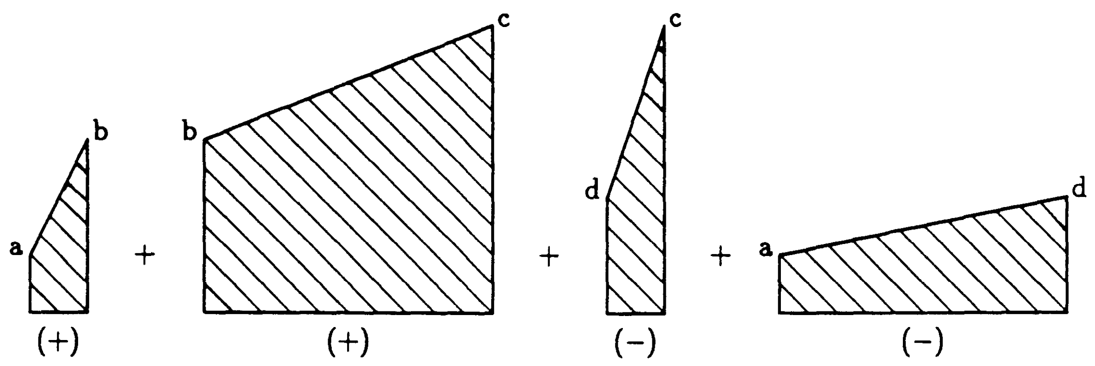


Figure 3.9: The mass flux is the sum of the areas of four trapezoids.

the trapezoids shown in Fig. 3.9. The mass flux is therefore

$$\dot{m} = \tfrac{1}{2}\left[ (\psi_2|_b + \psi_2|_a)(\psi_1|_b - \psi_1|_a) + (\psi_2|_c + \psi_2|_b)(\psi_1|_c - \psi_1|_b) \right. \tag{3.44}$$

$$\left. + (\psi_2|_d + \psi_2|_c)(\psi_1|_d - \psi_1|_c) + (\psi_2|_a + \psi_2|_d)(\psi_1|_a - \psi_1|_d) \right], \tag{3.45}$$

for $\psi_1$ and $\psi_2$ linear in $\psi_1$, $\psi_2$ space. This equation can cause round-off errors due to the subtraction of numbers which are the same to several significant figures. This is the formula coded in the computer program and for this application, it is not felt to have caused any problems. However, a formula which is identical if exact arithmetic is used, but causes less round-off error, is one derived from the cross product of two vectors. This is similar to the face area equation presented in the next section. $A_z$ in Eq. (3.53) represents a 2D area, and it can be shown to be equivalent to the trapezoidal area formula.

Equation (3.45) has been checked by calculating the net mass flux into and out of a hexahedral control volume. The net flux is identically zero. This demonstrates that $\nabla \cdot (\nabla\psi_1 \times \nabla\psi_2)$ is identically zero in a discretized sense. Also, if this equation is used in calculating momentum flux for the same control volumes, continuity and consistency is ensured.

One problem with this formulation is that the stream functions are not located at the vertices of the control volumes. The vertices of the $S$ face are shown in Fig. 3.5. The following simple interpolation formulas are used:

$$\psi_1|_a = \frac{1}{4}(\psi_1|_1 + \psi_1|_3 + \psi_1|_7 + \psi_1|_9), \tag{3.46a}$$

$$\psi_1|_b = \frac{1}{4}(\psi_1|_3 + \psi_1|_5 + \psi_1|_9 + \psi_1|_{11}), \tag{3.46b}$$

$$\psi_1|_c = \frac{1}{4}(\psi_1|_4 + \psi_1|_6 + \psi_1|_{10} + \psi_1|_{12}), \tag{3.46c}$$

$$\psi_1|_d = \frac{1}{4}(\psi_1|_2 + \psi_1|_4 + \psi_1|_8 + \psi_1|_{10}), \tag{3.46d}$$

$$\psi_2|_a = \frac{1}{4}(\psi_2|_1 + \psi_2|_2 + \psi_2|_7 + \psi_2|_8), \tag{3.46e}$$

$$\psi_2|_b = \frac{1}{4}(\psi_2|_4 + \psi_2|_5 + \psi_2|_{10} + \psi_2|_{11}), \tag{3.46f}$$

$$\psi_2|_c = \frac{1}{4}(\psi_2|_5 + \psi_2|_6 + \psi_2|_{11} + \psi_2|_{12}), \tag{3.46g}$$

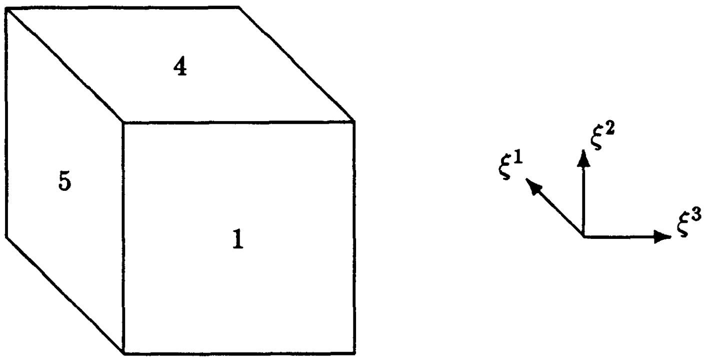


Figure 3.10: Hexahedron control volume in computational space. Faces 1 and 2 (2 is across from 1) are $A$ faces; faces 3 (3 is across from 4) and 4 are $B$ faces and faces 5 and 6 (6 is across from 5) are $S$ faces.

$$\psi_2|_d = \frac{1}{4}\left(\psi_2|_2 + \psi_2|_3 + \psi_2|_8 + \psi_2|_9\right). \tag{3.46h}$$

The corner vertices use a simple extrapolation formula. For faces that share vertices, consistent formulas are used.

## 3.4 Momentum Equations

The momentum equation in Eq. (2.15) can be discretized for a hexahedron control volume as shown in computational space in Fig. 3.10. The fluxes of momentum through each of the six faces must be balanced by the pressure force exerted on each face plus the volume integral of the centrifugal and Coriolis forces.

In Section 3.2, the calculation of velocity on each of the three types of faces was discussed. The mass flux across a face was discussed in the previous section.

The momentum flux is approximated as

$$\iint_{s_\psi} \overline{W} d\psi_1 d\psi_2 \simeq \sum_{i=1}^{6} (-1)^i \dot{m}_i \overline{W}_i \tag{3.47}$$

where $i$ is the face number.

The pressure force then is approximated as

$$\iint_s P\bar{n}ds \simeq \sum_{i=1}^{6}(-1)^i P_i \overline{A}_i. \tag{3.48}$$

where $P$ is the pressure and $\overline{A}$ is the area projection. The pressure is an independent variable defined at the center of each face. There are three types of faces. As shown in Fig. 3.10, faces 1 and 2 are $A$ faces, faces 3 and 4 are $B$ faces, and faces 5 and 6 are $S$ faces. The orientation is irrelevant for the momentum equation, but is important in the method of solution which is discussed in Chapter 5 and for the auxiliary pressure equations discussed in the next chapter. $P_1$ and $P_2$ are $A$ face pressures $P_A$, $P_3$ and $P_4$ are $B$ face pressures $P_B$, and $P_5$ and $P_6$ are $S$ face pressures $P_S$. The area projection for the face shown in Fig. 3.11 is determined by taking the cross product of the vectors $\bar{s}_1$ and $\bar{s}_2$:

$$\bar{s}_1 = dx_1\hat{\imath} + dy_1\hat{\jmath} + dz_1\hat{k}, \tag{3.49}$$

$$\bar{s}_2 = dx_2\hat{\imath} + dy_2\hat{\jmath} + dz_2\hat{k}, \tag{3.50}$$

$$\begin{aligned}
dx_1 &= \frac{1}{2}\left((x_c + x_d) - (x_a + x_b)\right), && (3.51a)\\
dy_1 &= \frac{1}{2}\left((y_c + y_d) - (y_a + y_b)\right), && (3.51b)\\
dz_1 &= \frac{1}{2}\left((z_c + z_d) - (z_a + z_b)\right), && (3.51c)
\end{aligned}$$

$$\begin{aligned}
dx_2 &= \frac{1}{2}\left((x_b + x_c) - (x_a + x_d)\right), && (3.52a)\\
dy_2 &= \frac{1}{2}\left((y_b + y_c) - (y_a + y_d)\right), && (3.52b)\\
dz_2 &= \frac{1}{2}\left((z_b + z_c) - (z_a + z_d)\right), && (3.52c)
\end{aligned}$$

$$\begin{aligned}
\overline{A} &= \bar{s}_1 \times \bar{s}_2,\\
&= \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ dx_1 & dy_1 & dz_1 \\ dx_2 & dy_2 & dz_2 \end{vmatrix},
\end{aligned}$$

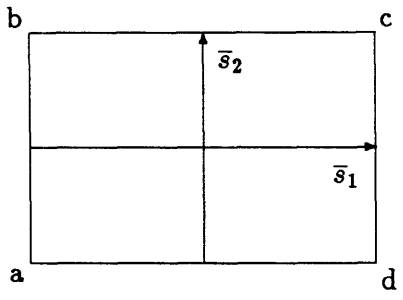


Figure 3.11: Area is determined from the vector cross product of $\bar{s}_1 \times \bar{s}_2$.

$$\begin{aligned} &= (dy_1dz_2 - dy_2dz_1)\hat{\imath} \\ &\quad +(dx_2dz_1 - dx_1dz_2)\hat{\jmath} \\ &\quad +(dx_1dy_2 - dx_2dy_1)\hat{k}, \\ &= A_x\hat{\imath} + A_y\hat{\jmath} + A_z\hat{k}. \end{aligned} \tag{3.53}$$

The values of $x$, $y$ and $z$ at the face nodes a - d are the average of the grid nodes in the $\xi^3$ direction.

The Coriolis and centrifugal source terms are approximated by averaging $\overline{S}$ at the two $S$ faces 5 and 6, and multiplying by the volume:

$$\iiint_\tau \rho\overline{S}d\tau = \frac{Vol}{2}\sum_{i=5}^{6}\left[\left(-\rho\omega^2x - 2\omega\rho W_y\right)_i\hat{\imath} + \left(-\rho\omega^2y + 2\omega\rho W_x\right)_i\hat{\jmath}\right]. \tag{3.54}$$

Only the $S$ faces are used for this source term because the calculations at an $S$ face are more accurate and mass fluxes are zero at the wall and periodic boundaries. This results in velocities not being needed at every $A$ or $B$ face.

The volume is calculated from the vector triple product of $\bar{t}_1$, $\bar{t}_2$ and $\bar{t}_3$ as shown in Fig. 3.12

$$\bar{t}_1 = dx_1\hat{\imath} + dy_1\hat{\jmath} + dz_1\hat{k}, \tag{3.55a}$$

$$\bar{t}_2 = dx_2\hat{\imath} + dy_2\hat{\jmath} + dz_2\hat{k}, \tag{3.55b}$$

$$\bar{t}_3 = dx_3\hat{\imath} + dy_3\hat{\jmath} + dz_3\hat{k}, \tag{3.55c}$$

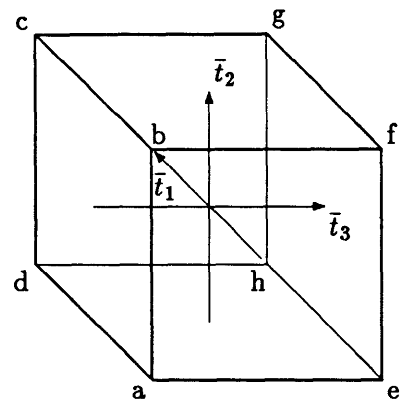


Figure 3.12: Volume is determined from vector triple product $(\bar{t}_1 \times \bar{t}_2) \cdot \bar{t}_3$.

where for the volume

$$dx_1 = \frac{1}{4}\left((x_c + x_d + x_g + x_h) - (x_a + x_b + x_e + x_f)\right), \tag{3.56a}$$

$$dx_2 = \frac{1}{4}\left((x_b + x_c + x_f + x_g) - (x_a + x_d + x_e + x_h)\right), \tag{3.56b}$$

$$dx_3 = \frac{1}{4}\left((x_e + x_f + x_g + x_h) - (x_a + x_b + x_c + x_d)\right), \tag{3.56c}$$

with corresponding equations used for $dy_1$, $dy_2$, $dy_3$, $dz_1$, $dz_2$ and $dz_3$.

$$\begin{aligned} Vol &= \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ dx_1 & dy_1 & dz_1 \\ dx_2 & dy_2 & dz_2 \end{vmatrix} \cdot (dx_3\hat{\imath} + dy_3\hat{\jmath} + dz_3\hat{k}), \\ &= (dy_1 dz_2 - dy_2 dz_1)dx_3 \\ &\quad +(dx_2 dz_1 - dx_1 dz_2)dy_3 \\ &\quad +(dx_1 dy_2 - dx_2 dy_1)dz_3. \end{aligned} \tag{3.57}$$

The discretized $x$ momentum equation is

$$\sum_{i=1}^{6}(-1)^i \left(W_x|_i \dot{m}_i + P_i\, A_x|_i\right) + \frac{Vol}{2}\sum_{i=5}^{6}\left(-\rho\omega^2 x - 2\omega\rho W_y\right)_i = 0. \tag{3.58}$$

The discretized $y$ momentum equation is

$$\sum_{i=1}^{6}(-1)^i \left(W_y|_i \dot{m}_i + P_i\, A_y|_i\right) + \frac{Vol}{2}\sum_{i=5}^{6}\left(-\rho\omega^2 y + 2\omega\rho W_x\right)_i = 0. \tag{3.59}$$

And the discretized $z$ momentum equation is

$$\sum_{i=1}^{6}(-1)^i \left(W_z|_i \, \dot{m}_i + P_i \, A_z|_i\right) = 0. \tag{3.60}$$

## 3.5 $S$, $A$ and $B$ Components of the Momentum Equation

The Cartesian components of the momentum equations are not actually solved, but a linear combination of these are.

The $S$ momentum equation is the component of momentum in the $\bar{t}_3$ direction (refer to Fig. 3.12). It is therefore

$$S \text{ momentum} = \frac{\bar{t}_3}{|\bar{t}_3|} \cdot (x \text{ momentum } \hat{\imath} + y \text{ momentum } \hat{\jmath} + z \text{ momentum } \hat{k}) = 0 \tag{3.61}$$

where the $x$, $y$ and $z$ momentum equations are Eqs. (3.58), (3.59) and (3.60) respectively. If the entropy convection option is specified, then the entropy convection equation is solved instead of the $S$ momentum equation.

The $A$ momentum equation is that component normal to $\bar{t}_2$ and $\bar{t}_3$. Therefore

$$A \text{ momentum} = \frac{\bar{t}_2 \times \bar{t}_3}{|\bar{t}_2 \times \bar{t}_3|} \cdot (x \text{ momentum } \hat{\imath} + y \text{ momentum } \hat{\jmath} + z \text{ momentum } \hat{k}) = 0. \tag{3.62}$$

The $B$ momentum equation is normal to $\bar{t}_3$ and $\bar{t}_1$. Therefore

$$B \text{ momentum} = \frac{\bar{t}_3 \times \bar{t}_1}{|\bar{t}_3 \times \bar{t}_1|} \cdot (x \text{ momentum } \hat{\imath} + y \text{ momentum } \hat{\jmath} + z \text{ momentum } \hat{k}) = 0. \tag{3.63}$$

The $S$, $A$ and $B$ components of the momentum equations are actually solved rather than the Cartesian components because:

1. it allows for the entropy convection option, and
2. it allows the equation system to be reduced.

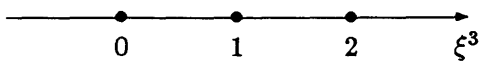


Figure 3.13: Pressure upwinding stencil.

For cases in which the grid moves to follow the stagnation stream surface, the unit vectors which define the $S$, $A$ and $B$ components of the momentum equations are stored at the first iteration. Changes in these vectors therefore are not needed when linearizing the equations.

## 3.6 Upwinded Pressure

The discrete equations as presented so far are ill-posed in supersonic regions without any artificial dissipation. This is shown in Appendix B. Giles [27] used "artificial compressibility" and upwinded density; Drela [16] used an upwinded velocity which is analogous to a bulk viscosity.

In this thesis, because pressure is a dependent variable, an upwinded pressure formulation is used. It is applied only on $S$ faces. The $S$ face pressures used in the momentum equations are $\tilde{P}$ such that $\tilde{P}_2$ is applied at node 2 in Fig. 3.13:

$$\tilde{P}_2 = P_2 + \mu_2(P_2 - P_1) - \mu_{c2}(P_1 - P_0). \tag{3.64}$$

The signs in this equation are correct. They are different from the signs used if an upwinded density is applied and are consistent with the stability analysis presented in Appendix B.

Equation (3.64) is a general formulation based on Giles [27]. If $\mu_{c2} = \mu_2$, the dissipation error produced by upwinding is second order accurate on smooth grids. If $\mu_{c2} = 0$, the error is first order accurate, and is used for most applications. However, the second order correction has been applied by first converging with $\mu_{c2} = 0$, and then setting $\mu_{c2} = \mu_2$ except in the direct vicinity of a shock. The term $\mu_2$ is defined as

$$\mu_2 = \begin{cases} \frac{M_2^2}{1+(\gamma-1)M_2^2}\left(1 - \frac{M_c^2}{M_2^2}\right) & ; \ M_2^2 \geq M_1^2 \text{ and } M_2^2 > M_c^2 \\ \frac{M_1^2}{1+(\gamma-1)M_1^2}\left(1 - \frac{M_c^2}{M_1^2}\right) & ; \ M_1^2 \geq M_2^2 \text{ and } M_1^2 > M_c^2 \\ 0 & ; \ \text{otherwise} \end{cases} \tag{3.65}$$

where

$$M^2 = \frac{W^2}{c^2} = \frac{W^2}{(\gamma-1)(I - \frac{W^2}{2} + \frac{\omega^2 r^2}{2})}. \tag{3.66}$$

The analysis in Appendix B derives the term in Equation (3.65). In most applications

$$M_c^2 = 0.9. \tag{3.67}$$

## 3.7 Entropy and Rothalpy Convection Equation

The streamwise momentum equation reduces to an entropy convection equation in the absence of shocks. The pressure upwinding must be applied in supersonic regions for stability, so in subsonic regions the entropy convection can be substituted for the streamwise momentum equation. This helps reduce errors associated with artificial entropy generation especially around leading edges.

The energy equation is also a convection equation. A two-dimensional table-lookup is used to apply the convection equation. The $\psi_1$, $\psi_2$, rothalpy and entropy are known upstream after a given iteration and put in a 2D interpolation table. The value of $\psi_1$ and $\psi_2$ are also known anywhere in the field. Rothalpy and entropy are then obtained by interpolation using the local $\psi_1$ and $\psi_2$ and upstream interpolation table.

# Chapter 4
# Auxiliary Pressure Equations and Discretized Boundary Conditions

## 4.1 Complete System of Equations

In the previous chapter, the discretized finite volume momentum equations were derived. There are three equations per volume. For $ni$ blade-to-blade, $nj$ spanwise and $nk$ streamwise grid nodes, there are

$$nm = (ni-1)(nj-1)(nk-2) \tag{4.1}$$

volumes where $i$ goes from 1 to $ni-1$, $j$ from 1 to $nj-1$ and $k$ from 2 to $nk-1$,

$$n1e = ni(nj-1)nk \tag{4.2}$$

$\psi_1$ values where $i$ goes from 1 to $ni$, $j$ from 1 to $nj-1$ and $k$ from 1 to $nk$,

$$n2e = (ni-1)nj\ nk \tag{4.3}$$

$\psi_2$ values where $i$ goes from 1 to $ni-1$, $j$ from 1 to $nj$ and $k$ from 1 to $nk$,

$$nsf = (ni-1)(nj-1)(nk-1) \tag{4.4}$$

$S$ face pressures where $i$ goes from 1 to $ni-1$, $j$ from 1 to $nj-1$ and $k$ from 1 to $nk-1$,

$$naf = ni(nj-1)(nk-2) \tag{4.5}$$

$A$ face pressures where $i$ goes from 1 to $ni$, $j$ from 1 to $nj-1$ and $k$ from 2 to $nk-1$, and

$$nbf = (ni-1)nj(nk-1) \tag{4.6}$$

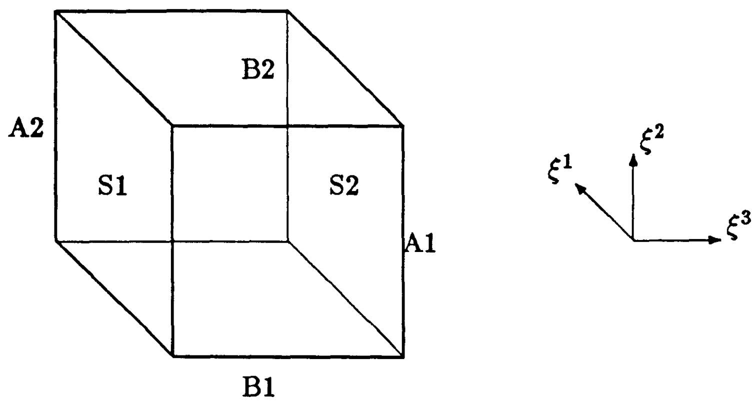


Figure 4.1: Pressures applied to the faces of a control volume.

$B$ face pressures where $i$ goes from 1 to $ni - 1$, $j$ from 1 to $nj$ and $k$ from 2 to $nk - 1$. Essentially, five equations are needed per volume plus boundary conditions to have a complete system of equations which can satisfy the unknowns. In this chapter, auxiliary pressure equations are derived. There are two of these per volume which make a total of five equations per volume.

## 4.2 Auxiliary Pressure Equations

Figure 4.1 shows the pressures on each face of a control volume.

Two constraint equations are applied to the pressures on the control volume:

$$P_{S1} + P_{S2} = P_{A1} + P_{A2} + 2P_{cA}, \tag{4.7}$$

and

$$P_{S1} + P_{S2} = P_{B1} + P_{B2} + 2P_{cB}. \tag{4.8}$$

Eq. (4.7) is identical to that used in ISES [25] [27] [16]. Eq. (4.8) is the added equation for 3D. Essentially, they constrain the average values of $P_B$ and $P_A$. The terms $P_{cA}$ and $P_{cB}$ are required to eliminate odd-even modes in the $\psi$ values which are shown in Fig. 4.2.

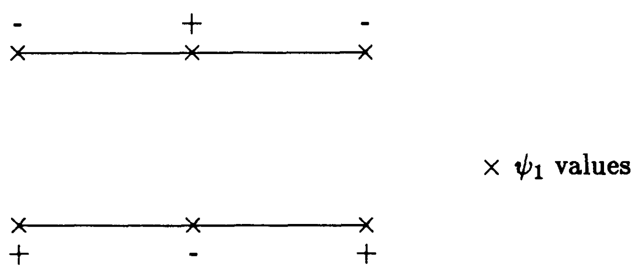


Figure 4.2: Odd-even modes of $\psi_1$ which must be constrained.

These modes are constrained by the far-field and wall boundary conditions, but are allowed by the periodic boundary conditions. To constrain them with any boundary condition, the pressure correction term is used. The correction term can be interpreted as a second difference operator:

$$2P_c = P_{S1} - 2\overline{P} + P_{S2} \tag{4.9}$$

where $P_c$ is either $P_{cA}$ or $P_{cB}$ depending on the equation and

$$\overline{P} = \frac{P_{A1} + P_{A2}}{2} \tag{4.10}$$

or

$$\overline{P} = \frac{P_{B1} + P_{B2}}{2}. \tag{4.11}$$

This second difference is related to the second derivative along the $\xi^3$ grid line:

$$2P_c \approx \frac{\partial^2 P}{\partial(\xi^3)^2} \tag{4.12}$$

For isentropic flow, constant rothalpy and negligible change in radius,

$$\frac{\partial^2 P}{\partial(\xi^3)^2} = \frac{\partial}{\partial \xi^3}\left[\left.\frac{\partial P}{\partial A}\right|_{s,H} \frac{\partial A}{\partial(\rho W)} \frac{\partial(\rho W)}{\partial \xi^3}\right]. \tag{4.13}$$

For compressible flow,

$$\left.\frac{\partial P}{\partial A}\right|_{s,H} = \frac{P}{A}\frac{\gamma M^2}{(1-M^2)}. \tag{4.14}$$

For incompressible flow,

$$\frac{\partial P}{\partial A} = \frac{\rho W^2}{A}. \tag{4.15}$$

For

$$\dot{m} = \rho W A = \text{const}, \tag{4.16}$$

$$\frac{\partial A}{\partial(\rho W)} = -\frac{\dot{m}}{(\rho W)^2} = -\frac{A}{\rho W}. \tag{4.17}$$

Therefore, for compressible flow, $P_c$ could be defined by

$$2P_c = -\frac{P\gamma M^2}{(1-M^2)}C, \tag{4.18}$$

and for incompressible flow

$$2P_c = -\rho W^2 C. \tag{4.19}$$

where $C$ is the curvature term:

$$C = \frac{\partial^2 \ln(\rho W)}{\partial(\xi^3)^2}. \tag{4.20}$$

Now from the stream function definition and assuming $\psi_1$ varies only in $\xi_1$ and $\psi_2$ varies only in $\xi_2$

$$\ln(\rho W) \approx \ln\left(\frac{\partial\psi_1}{\partial\xi_1}\right) + \ln\left(\frac{\partial\psi_2}{\partial\xi_2}\right) + \text{constants}. \tag{4.21}$$

It is best to separate this into two parts. $C_1$ for the $A$ constraint equation

$$C_1 = \frac{1}{4}\frac{\partial^2 \ln\left(\frac{\partial\psi_1}{\partial\xi_1}\right)}{\partial(\xi^3)^2}, \tag{4.22}$$

and $C_2$ for the $B$ constraint equation

$$C_2 = \frac{1}{4}\frac{\partial^2 \ln\left(\frac{\partial\psi_2}{\partial\xi_2}\right)}{\partial(\xi^3)^2}. \tag{4.23}$$

After some manipulating and discretizing using the stencils in Fig. 4.3:

$$C_1 = \left[\frac{(\psi_1|_4 - \psi_1|_1) - 2(\psi_1|_5 - \psi_1|_2) + (\psi_1|_6 - \psi_1|_3)}{(\psi_1|_4 + \psi_1|_5 + \psi_1|_6) - (\psi_1|_1 + \psi_1|_2 + \psi_1|_3)}\right], \tag{4.24}$$

and

$$C_2 = \left[\frac{(\psi_2|_4 - \psi_2|_1) - 2(\psi_2|_5 - \psi_2|_2) + (\psi_2|_6 - \psi_2|_3)}{(\psi_2|_4 + \psi_2|_5 + \psi_2|_6) - (\psi_2|_1 + \psi_2|_2 + \psi_2|_3)}\right]. \tag{4.25}$$

From the ISES experience, this $P_c$ term is not needed in supersonic regions, even though $P_c \to \infty$ at $M = 1$. This term will, therefore, be modified to go smoothly to zero at $M = 1$. Also a fraction $\kappa_A$ and $\kappa_B$ of $C_1$ and $C_2$ respectively will be used. The pressure correction term for incompressible flow is:

$$2P_{cA} = -4\rho W^2 \kappa_A C_1, \tag{4.26}$$

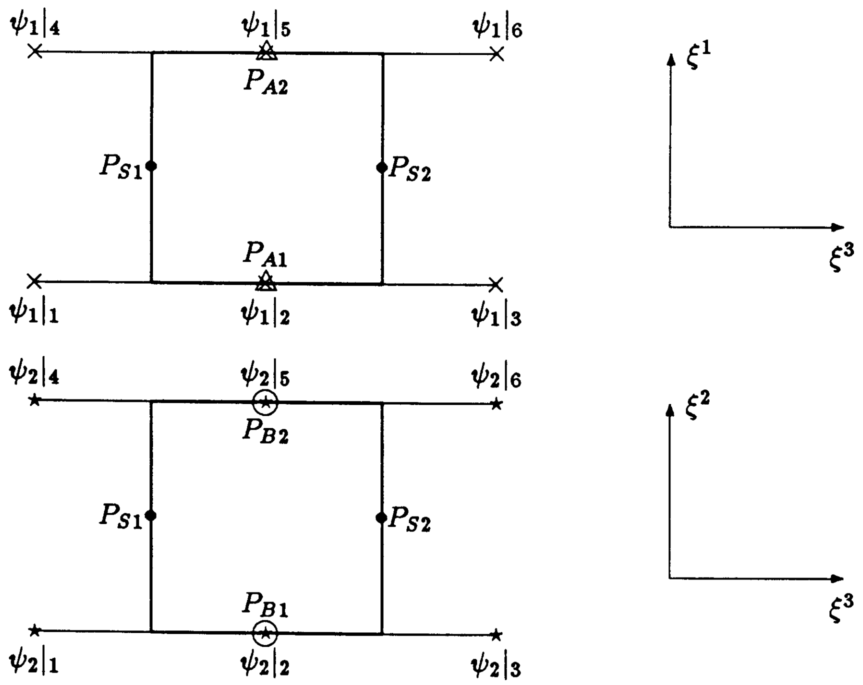


Figure 4.3: Stencils for Auxiliary Pressure equations.

$$2P_{cB} = -4\rho W^2 \kappa_B C_2, \tag{4.27}$$

and for compressible flow:

$$2P_{cA} = \begin{cases} -4\gamma P M^2(1 - M^2)\kappa_A C_1 & M < 1 \\ 0 & M \geq 1 \end{cases} \tag{4.28}$$

$$2P_{cB} = \begin{cases} -4\gamma P M^2(1 - M^2)\kappa_B C_2 & M < 1 \\ 0 & M \geq 1 \end{cases} \tag{4.29}$$

where $W$, $P$ and $M$ are averaged from the $S$ faces.

## 4.3 Entropy Boundary Condition

At every $S$ face of the inlet plane, the entropy is specified. The equation is

$$\hat{s}_{Sface} - \hat{s}_{specified} = 0, \tag{4.30}$$

where $\hat{s}_{S_{face}}$ is determined using Equations (2.49), (2.33), (2.35), and (2.37). The quantity $(\rho W)^2$ can be determined at an $S$ face using the discretization procedures discussed in the previous chapter.

This equation is applied

$$nsbc = (ni - 1)(nj - 1) \tag{4.31}$$

times.

## 4.4 Upstream Angle Boundary Conditions

If the grid is constructed so upstream it is aligned in a prescribed flow direction, then

$$\psi_1|_{i,j,1} - \psi_1|_{i,j,2} = 0, \tag{4.32}$$

and

$$\psi_2|_{i,j,1} - \psi_2|_{i,j,2} = 0. \tag{4.33}$$

This is because the streamwise grid lines are streamlines if they are aligned with the flow and both $\psi_1$ and $\psi_2$ are constant on a streamline.

The relative flow angle is generally known at the inlet and generating a grid with the correct inlet angles is relatively straight forward. Eq. (4.32) is applied

$$nsi1up = (ni - 2)(nj - 1) \tag{4.34}$$

times, and Eq. (4.33) is applied

$$nsi2up = (ni - 1)(nj - 2) \tag{4.35}$$

times.

## 4.5 Wall Boundary Conditions

For duct flow with four walls, the simplest way to implement the no-flow boundary condition is to specify

$$\psi_1|_{left} = 0, \tag{4.36}$$

$$\psi_1|_{right} = 1, \tag{4.37}$$

$$\psi_2|_{lower} = 0, \tag{4.38}$$

and

$$\psi_2|_{upper} = \psi_2|_{max} = \dot{m}|_{duct}, \tag{4.39}$$

where left and right are at $i = 1$ and $i = ni$ respectively.

At the left and right wall, this represents

$$nwall1 = 2\,nk(nj - 1) \tag{4.40}$$

equations, and at the upper and lower walls, this represents

$$nwall2 = 2\,nk(ni - 1) \tag{4.41}$$

equations.

## 4.6 Downstream Boundary Conditions

Figure 4.4 shows the $\psi_1$ and $\psi_2$ values at the exit cross-flow plane and the pressures $P_S$ at the furthest downstream $S$ face. Physically, a flow rate is specified or a static pressure at one location downstream is used to set the flow rate (for choked flow, the boundary condition must be modified). However, the flow rate has already been applied as the wall boundary condition. Based on the number of unknowns, which are the sum of Eq. (4.2) through Eq. (4.6), and the $5\,nm$ control volume equations plus

$$nsbc + nsi1up + nsi2up + nwall1 + nwall2 \tag{4.42}$$

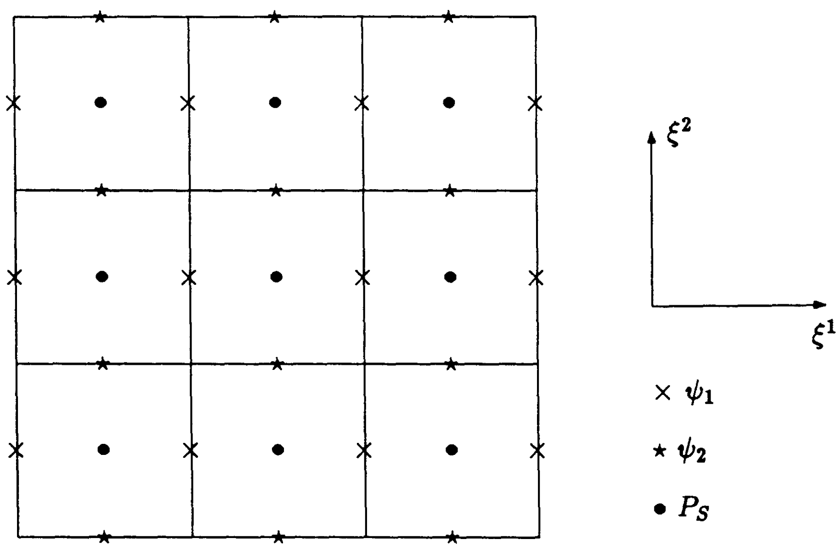


Figure 4.4: A schematic of the exit plane showing the locations of $\psi_1$, $\psi_2$ and $P_S$. $\xi^3$ is out of the paper.

boundary condition equations already discussed leaves

$$(ni-2)(nj-1)+(ni-1)(nj-2) \tag{4.43}$$

conditions which still must be satisfied. These conditions must specify the downstream distribution of $\psi_1$ and $\psi_2$.

One option is to specify $\psi_1$ and $\psi_2$ everywhere downstream. This would complete the system of equations. However, this implies a known flow rate distribution which is in general not known. The actual boundary condition must indirectly specify $\psi_1$ and $\psi_2$ while allowing the flow to exit the boundary smoothly. One downstream boundary condition which has been used with some success is:

- Specify the distribution of $\psi_1$ in $\xi^2$:

$$\psi_1|_{i,j,nk} - \psi_1|_{i,j-1,nk} = 0, \tag{4.44}$$

where $j$ varies from 2 to $nj-1$, and $i$ varies from 2 to $ni-1$.

- Specify that the streamwise difference of the radial pressure difference is zero:

$$(P_S|_{i,j,nk-1} - P_S|_{i,j-1,nk-1}) - (P_S|_{i,j,nk-2} - P_S|_{i,j-1,nk-2}) = 0, \qquad (4.45)$$

  where $j$ varies from 2 to $nj - 1$ and $i$ varies from 1 to $ni - 1$.

- For the other $(ni - 2)$ conditions, specify that the streamwise pressure difference of the $\theta$ pressure difference is zero:

$$(P_S|_{i,j,nk-1} - P_S|_{i-1,j,nk-1}) - (P_S|_{i,j,nk-2} - P_S|_{i-1,j,nk-2}) = 0, \qquad (4.46)$$

  where $i$ varies from 2 to $ni - 1$ and $j = nj - 1$.

For 2D applications, these reduce to applying just the difference equations Eq. (4.46) or Eq. (4.45). Many 2D solvers specify a constant static pressure downstream. This can be applied in this solver, but because of the small amount of artificial viscosity, this condition can only be applied many chord lengths downstream of a trailing edge or else there are noticeable kinks in the solution.

## 4.7 Periodic Boundary Conditions

In Section 2.6, it was mentioned that the stagnation stream surface and wake will be determined by setting the pressures equal across these surfaces. This involves moving the grid. The grid movement along a $\xi^1$ grid line will all involve the same $\Delta x_1|_{j,k}$.

In 3D, the grid movement will be specified at the $\psi_1$ locations which are the center of the grid edges in the $\xi^2$ direction. The values for the grid node movement will be interpolated and extrapolated.

The interpolation formulas are:

$$\Delta x_{1n}|_{j,k} = \begin{cases} \Delta x_{1c}|_k, & nj = 2 \\ \frac{1}{2}(\Delta x_{1c}|_{j,k} + \Delta x_{1c}|_{j-1,k}), & 1 < j < nj, nj > 2 \\ \frac{3}{2}\Delta x_{1c}|_{1,k} - \frac{1}{2}\Delta x_{1c}|_{2,k}, & j = 1, nj > 2 \\ \frac{3}{2}\Delta x_{1c}|_{nj-1,k} - \frac{1}{2}\Delta x_{1c}|_{nj-2,k}, & j = nj, nj > 2 \end{cases} \qquad (4.47)$$

where $\Delta x_{1n}$ is the $\Delta x_1$ at the node and $\Delta x_{1c}$ is the $\Delta x_1$ calculated at a $\psi_1$ location.

If an entire boundary is periodic, $\Delta x_{1c} = 0$ must be specified at some location. The inlet angle boundary condition can be applied upstream and

$$P_A|_{i=1,j,k} - P_A|_{i=ni,j,k} = 0 \tag{4.48}$$

can be applied for $j = 1$, to $nj - 1$ and $k = 2$ to $nk - 1$. There are, therefore, enough equations for each of the new $\Delta x_{1c}$ unknowns of which there are

$$ndx1 = (nj - 1)nk. \tag{4.49}$$

To allow for both a blade and periodic conditions, $\Delta x_c = 0$ on the blade and then Eq. (4.48) is not applied. To establish the correct circulation on the blade, the pressure difference is set to zero at the trailing edge. The wake starts at the trailing edge of the blade which can be either zero or finite thickness[1].

The stagnation point location on the leading edge is not known *a priori*. For very sharp leading edges, it can be assumed to be at the leading edge. For thicker leading edges, assuming the location of the stagnation point can lead to large pressure spikes. The method described in the following section can be used to determine the stagnation point location and eliminate the pressure spike.

## 4.8 Blunt Leading Edge Treatment

The leading edge treatment incorporated in this solver is very similar to that described by Drela [16]. Basically, the stagnation point nodes are allowed to slide along the airfoil surface. To prevent grid crossovers, the rest of the grid slides with the stagnation point.

The airfoil at every spanwise station is defined by a cubic spline in $m', \theta$ coordinates. The stagnation point is specified by the arc length from the trailing edge to the stagna-

[1]A finite thickness trailing edge can be used to simulate a blade with a rounded trailing edge. The wake thickness can then be prescribed to simulate the blockage of the wake.

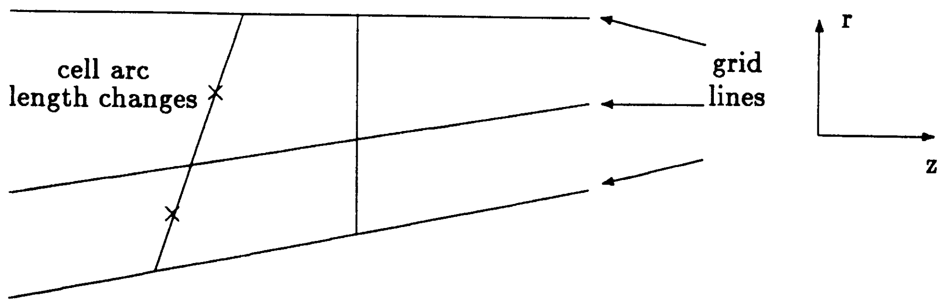


Figure 4.5: Location of cell arc length unknowns at a leading edge with respect to the grid lines. Meridional plane shown with hub and tip grid lines.

tion point in $m', \theta$ units. Changes in arc lengths on the grid lines are determined from the calculated changes at the center of the grid cells as shown in Fig. 4.5. Just as with the grid movement, interpolation and extrapolation formulas are used:

$$\Delta s_{legl}|_j = \begin{cases} \Delta s_{le}, & nj = 2 \\ \frac{1}{2}(\Delta s_{le}|_j + \Delta s_{le}|_{j-1}), & 1 < j < nj, nj > 2 \\ \frac{3}{2}\Delta s_{le}|_1 - \frac{1}{2}\Delta s_{le}|_2, & j = 1, nj > 2 \\ \frac{3}{2}\Delta s_{le}|_{nj-1} - \frac{1}{2}\Delta s_{le}|_{nj-2}, & j = nj, nj > 2 \end{cases} \tag{4.50}$$

where $\Delta s_{legl}$ is the change in arc length of the leading edge on a grid line and $\Delta s_{le}$ is the change in cell arc length. The arc length is determined iteratively such that

$$s_{legl}^{new} = s_{legl}^{old} + \Delta s_{legl}. \tag{4.51}$$

Basically there are

$$nsle = nj - 1 \tag{4.52}$$

unknown values of $\Delta s_{le}$ which are determined by $nsle$ equations which set the pressure equal at the leading edge. This is Eq. (4.48) where $k$ is at the leading edge station.

The movement of the grid lines on the surface of the blades is determined by keeping the relative arc distances of the grid points the same on both pressure and suction surfaces. Away from the blade, the grid movement decays exponentially as a function of the distance from the stagnation point.

## 4.9 Design Option

Often in turbomachinery applications, the desired thickness has been specified due to mechanical constraints such as stress and vibration. A loading distribution can be specified which provides the desired turning. Based on this, the blade shape (particularly the camber line) can be determined. This can be accomplished since Eq. (4.48) can be modified to include a desired pressure loading:

$$P_A|_{i=1,j,k} - P_A|_{i=ni,j,k} - \Delta P_A|_{desired} = 0. \tag{4.53}$$

Using a sharp leading edge, and specifying that the trailing edge is at the leading edge, then the desired blade shape is produced from the resulting loaded wake surfaces. The only restriction is that the grid points on both sides of the blade must have the same axial location. This ensures the pressure jump is related to a change in tangential momentum.

## 4.10 System Closure

In Section 4.6, it was shown that the system of equations was closed for a duct problem. As unknowns were added in Section 4.7 and Section 4.8 for periodicity and leading edges, additional equations were supplied to satisfy these unknowns. Therefore the system of equations is closed for each flow situation discussed.

# Chapter 5
# Solution Method

## 5.1 Newton's Method – Use of SMP

In the previous two chapters, a closed system of nonlinear equations was defined. The solution of this system will be obtained iteratively using Newton's method (also called Newton-Raphson method) [15] [41].

The Newton's method algorithm is:

**Algorithm:** Given $\mathbf{F}$ (a vector of equations $\Re^n \rightarrow \Re^n$ ) and $\mathbf{x}_0 \in \Re^n$ (the initial solution vector) solve at each iteration $k$

$$\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{F}(\mathbf{x}_k), \tag{5.1}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k \tag{5.2}$$

where $\mathbf{J}(\mathbf{x}_k)$ is the Jacobian of $\mathbf{F}$. $\mathbf{J}(\mathbf{x}_k)$ is therefore represented by a matrix of partial derivative components $\frac{\partial f_i}{\partial x_j}$, where $f_i$ is the $i$'th row of $\mathbf{F}$. The iteration continues until the error is below a certain tolerance

$$||\mathbf{F}(\mathbf{x}_k)|| \leq \epsilon. \tag{5.3}$$

One advantage of using Newton's method is its fast convergence rate. Specifically, Newton's method converges quadratically [15] provided it is given a good starting guess. That is

$$||\mathbf{x}_{k+1} - \mathbf{x}_*|| \leq \gamma||\mathbf{x}_k - \mathbf{x}_*||^2, \;\; k = 0, 1, ... \tag{5.4}$$

for the exact solution $\mathbf{x}_*$ and $\gamma$ is a constant. The disadvantages of Newton's method are that it is not globally convergent for all problems (Section 5.5 explains some methods for dealing with this problem). Also, the Jacobian is required for each iteration which may be very complex and costly to compute. A solution of the system of linear equations is also required during each iteration.

To obtain the partial derivative components of the Jacobian, each equation was linearized using a Symbolic Manipulation Program, SMP [13]. A strategy similar to that used by Wigton [63] is used, except that he used the MACSYMA program. The equations are input into SMP, and the equations for the partial derivatives are output as FORTRAN source code. This process is demonstrated in Appendix A, and the SMP input for each equation is in Appendix E.

## 5.2 Reduced System

To decrease the size of the matrix to be solved, and create a better conditioned matrix, the changes in $P_A$ and $P_B$ will be eliminated from the linearized system of equations. A function

$$ELIM(eq1, eq2, var, eqout) \tag{5.5}$$

is defined which combines the linear equations *eq*1 and *eq*2 to eliminate the dependence on *var* to yield equation *eqout*. It is used in the equation reduction discussion which follows, and a FORTRAN implementation of the function is actually used in the solver. This equation reduction is not strictly adding and subtracting equations, but is also based on physical arguments which will also be discussed.

### 5.2.1 Reduced $S$ Momentum Equation

A general discrete $S$ momentum equation contains the pressures on all six sides of a volume which is represented in Fig. 4.1. However, this equation depends only very weakly on $P_{A1}$, $P_{A2}$, $P_{B1}$ and $P_{B2}$. In fact, for a uniform rectangular grid, there is no contribution of the $P_A$'s and $P_B$'s. To eliminate the dependence entirely, a linear combination of the $A$ momentum, $B$ momentum and two auxiliary pressure equations are added. The four weights of the equations being added are determined by the four conditions that the resultant equation does not depend on the four pressures $P_{A1}$, $P_{A2}$, $P_{B1}$ and $P_{B2}$. The $S$, $A$ and $B$ components of the momentum equation and the $A$ and $B$ auxiliary pressure equations, are referred to as *smom*, *amom*, *bmom*, *Paeq*, and *Pbeq*. $P_{A1}$, $P_{A2}$, $P_{B1}$ and $P_{B2}$ can be eliminated as follows:

$$ELIM(smom, Paeq, P_{A1}, rseq1), \tag{5.6a}$$

$$ELIM(rseq1, Pbeq, P_{B1}, rseq2), \tag{5.6b}$$

$$ELIM(amom, Paeq, P_{A1}, rseq3), \tag{5.6c}$$

$$ELIM(rseq3, Pbeq, P_{B1}, rseq4), \tag{5.6d}$$

$$ELIM(bmom, Paeq, P_{A1}, rseq5), \tag{5.6e}$$

$$ELIM(rseq5, Pbeq, P_{B1}, rseq6), \tag{5.6f}$$

$$ELIM(rseq2, rseq4, P_{A2}, rseq7), \tag{5.6g}$$

$$ELIM(rseq4, rseq6, P_{A2}, rseq8), \tag{5.6h}$$

$$ELIM(rseq7, rseq8, P_{B2}, rseq9). \tag{5.6i}$$

The equation *rseq*9 is now the reduced linearized $S$ momentum equation. It is very similar in properties to the streamwise momentum equation in ISES [25] [27] [16].

### 5.2.2 Reduced $A$ Momentum Equation

Fig. 5.1 shows how the discrete equation for two volumes can be combined to eliminate the $P_A$'s and $P_B$'s. The equations *amom*1, *bmom*1, *Paeq*1, and *Pbeq*1 represent the equations for volume I, and *amom*2, *bmom*2, *Paeq*2 and *Pbeq*2 represent the equations for volume II. The physical reasoning behind the following equation manipulation is that these equations should depend only weakly on the $P_B$'s (for a uniform rectangular grid, there is no dependence). These four $P_B$'s are eliminated using the two $B$ momentum equations and the two $B$ auxiliary pressure equations. The $A$ momentum equations depend very strongly on the $P_A$'s. Using the two $A$ auxiliary pressure equations and combining the two $A$ momentum equations replaces the dependence of the $P_A$'s with a

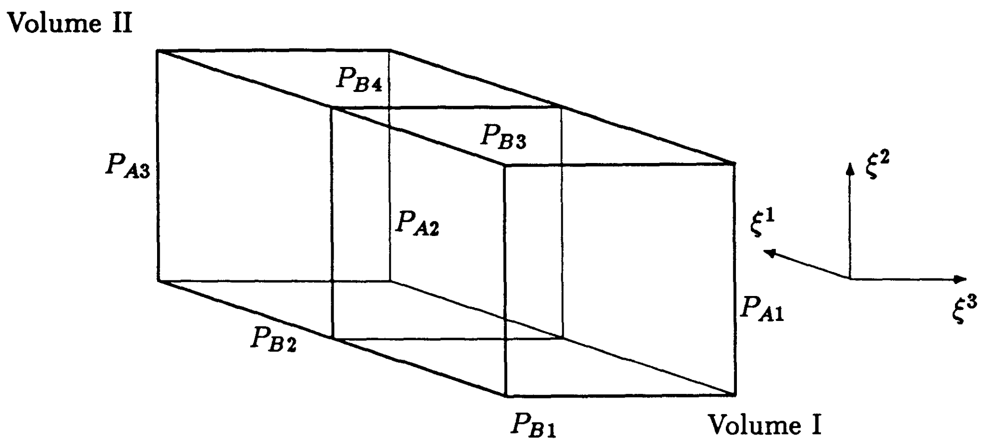


Figure 5.1: Volumes used for the reduced $A$ momentum equation.

dependence on the $P_S$'s. This is sketched in Fig. 5.2 in which the equation manipulation yields a pressure difference in the $\xi^1$ direction similar to $(P_{S2}+P_{S4})-(P_{S1}+P_{S3})$, but with different coefficients so the $P_A$'s are eliminated. This pressure difference is then related to the streamline curvature, and therefore the second derivative of the stream functions.

The elimination of $P_A$ and $P_B$ proceeds as follows:

$$ELIM(amom1, Paeq1, P_{A1}, raeq1), \tag{5.7a}$$

$$ELIM(amom2, Paeq2, P_{A3}, raeq2), \tag{5.7b}$$

$$ELIM(bmom1, Paeq1, P_{A1}, raeq3), \tag{5.7c}$$

$$ELIM(bmom2, Paeq2, P_{A3}, raeq4), \tag{5.7d}$$

$$ELIM(raeq1, Pbeq1, P_{B1}, raeq5), \tag{5.7e}$$

$$ELIM(raeq2, Pbeq2, P_{B2}, raeq6), \tag{5.7f}$$

$$ELIM(raeq3, Pbeq1, P_{B1}, raeq7), \tag{5.7g}$$

$$ELIM(raeq4, Pbeq2, P_{B2}, raeq8), \tag{5.7h}$$

$$ELIM(raeq5, raeq7, P_{B3}, raeq9), \tag{5.7i}$$

$$ELIM(raeq6, raeq8, P_{B4}, raeq10), \tag{5.7j}$$

$$ELIM(raeq9, raeq10, P_{A2}, raeq11). \tag{5.7k}$$

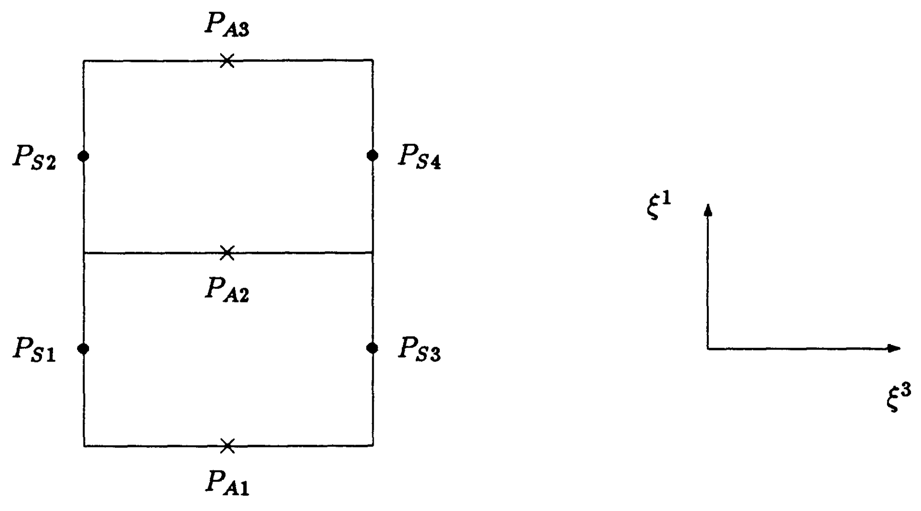


Figure 5.2: Projection of the reduced $A$ momentum control volumes in the $\xi^1$,$\xi^3$ plane.

The equation *raeq*11 is now the reduced $A$ momentum equation. This resulting equation is very similar in properties to the normal momentum equation in ISES. Equation *raeq*9 or *raeq*10 can be used to calculate the change of $P_{A2}$ with respect to other variables not including any $P_A$ or $P_B$. Either of these equations can be used to update $P_{A2}$ once the other variables have been solved.

### 5.2.3 Reduced $B$ Momentum Equation

The reduced $B$ momentum equation is derived using the same approach as for the reduced $A$ momentum equation. A $P_B$ pressure is shared between two volumes stacked in the $\xi^2$ direction as shown is Fig. 5.3.

### 5.2.4 Reduced Matrix Structure

None of the boundary conditions use $P_A$ or $P_B$ except for the periodicity or design option boundary condition which specifies the difference in $P_A$ across a stagnation stream surface. These occurrences of $P_A$ can be eliminated from the system by creating

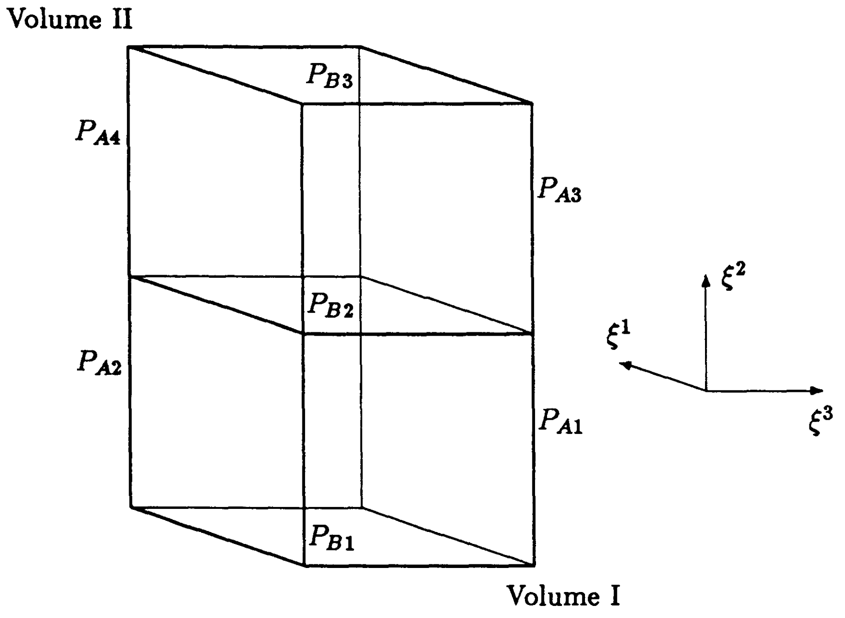


Figure 5.3: Volumes used for the reduced $B$ momentum equation.

a special reduced $A$ momentum equation for a periodic or design option boundary.

It is instructive to count the reduced number of equations and reduced number of unknowns and assign a variable-equation match-up. There are

$$n1e = ni(nj - 1)nk \tag{5.8}$$

$\delta\psi_1$ values. Of this, there are *nwall1* $\psi_1$ wall boundary conditions, *nsi1up* $\psi_1$ angle boundary conditions, and $(ni - 2)(nj - 1)$ downstream pressure[1] or $\psi_1$ boundary conditions which can be applied. There are then $(ni-2)(nj-1)(nk-2)$ reduced $A$ momentum equations which can be applied to satisfy the remainder of the $\delta\psi_1$ values.

There are

$$n2e = (ni - 1)nj\ nk \tag{5.9}$$

$\delta\psi_2$ values. These are satisfied by *nwall2* wall boundary conditions, *nsi2up* $\psi_2$ angle boundary conditions and $(ni - 1)(nj - 2)$ downstream pressure boundary conditions and $(ni - 1)(nj - 2)(nk - 2)$ reduced $B$ momentum equations.

[1]The pressure difference does not contain $\psi_1$ directly, but could be combined with a reduced $S$ momentum equation so it does, or the equations can be ordered differently. This reordering is discussed in Section 5.4.

The $nsf = (ni-1)(nj-1)(nk-1)$ $\delta P_S$ values are satisfied by the $nsbc$ entropy boundary conditions and either $nm$ reduced $S$ momentum equations, or $nm$ entropy convection equations.

The $ndx1$ $\Delta x_{1c}$ values are satisfied by either the blade ($\Delta x_{1c} = 0$) boundary condition, angle boundary condition or the special periodic reduced $A$ momentum equation.

The $nsle$ leading edge movement values are satisfied by a special periodic reduced $A$ momentum equation applied at the leading edge.

The variable-equation match-up is:

$$\psi_1|_{upstream} \leftarrow \psi_1 \text{ angle boundary condition} \tag{5.10a}$$

$$\psi_1|_{walls} \leftarrow \psi_1 \text{ wall boundary condition} \tag{5.10b}$$

$$\psi_1|_{downstream} \leftarrow \text{downstream } P \text{ or } \psi_1 \text{ boundary condition} \tag{5.10c}$$

$$\psi_1|_{interior} \leftarrow \text{reduced } A \text{ momentum equation} \tag{5.10d}$$

$$\psi_2|_{upstream} \leftarrow \psi_2 \text{ angle boundary condition} \tag{5.10e}$$

$$\psi_2|_{walls} \leftarrow \psi_2 \text{ wall boundary condition} \tag{5.10f}$$

$$\psi_2|_{downstream} \leftarrow \text{downstream } P \text{ or } \psi_2 \text{ boundary condition} \tag{5.10g}$$

$$\psi_2|_{interior} \leftarrow \text{reduced } B \text{ momentum equation} \tag{5.10h}$$

$$P_S|_{upstream} \leftarrow \text{entropy boundary condition} \tag{5.10i}$$

$$P_S|_{\text{other than upstream}} \leftarrow \text{reduced } S \text{ momentum equation} \tag{5.10j}$$

$$\Delta x_{1c}|_{\text{periodic}} \leftarrow \text{special reduced } A \text{ momentum equation} \tag{5.10k}$$

$$\Delta x_{1c}|_{\text{on blade}} \leftarrow 0 \text{ boundary condition} \tag{5.10l}$$

$$\Delta s_{le} \leftarrow \text{special reduced } A \text{ momentum equation.} \tag{5.10m}$$

The matrix structure is then defined by this variable-equation match-up when the solution vector

$$\mathbf{s}' = \begin{bmatrix} \delta\psi_1 \\ \delta\psi_2 \\ \delta P_S \\ \Delta x_{1c} \\ \Delta s_{le} \end{bmatrix} . \tag{5.11}$$

The reduced matrix system to be solved is

$$\mathbf{As}' = \mathbf{b}'. \tag{5.12}$$

## 5.3 Direct Matrix Solvers

The use of Newton's method requires the solution of the matrix equation $\mathbf{Ax} = \mathbf{b}$. There are many ways to solve this equation and the choice of the method depends on the type of matrix (i.e. symmetric or nonsymmetric), its condition number, and the matrix size and structure (i.e. is it banded, full or sparse).

The matrix to be solved is nonsymmetric, and is very large and sparse. The structure is discussed in the next subsection. The matrix condition is dependent on Mach number, but is generally well conditioned.

For many problems, a matrix can be solved iteratively. These schemes can be faster and require less storage than the alternative, a direct solver. However, a stable or convergent iterative technique cannot always be derived for a general nonsymmetric matrix. In this thesis research, an iterative scheme was tried. It was based on solving the matrix on a physical plane-by-plane basis. First it would solve $\psi_1$ and $P_S$ coupled, and then $\psi_2$ and $P_S$. It worked well for a simple duct test case, but it was not very robust for more complicated problems, and was therefore abandoned and a modified direct solver has been used.

Direct solvers are becoming increasingly popular for solving problems in CFD which use Newton's method as has been done in ISES [25] [27] [16][63] and other solvers [54] [60]. These are all 2D applications and have been very successful. It was hoped that by applying the latest advances in solving matrices that the use of a direct method for a 3D application would be as successful.

In reference [21], an order estimate for computer time and storage is given for 2D and 3D finite difference problems (it is assumed that these are simple 5 or 7 node local operators) using either a banded or profile method, or a general sparse method. For 2D grids, with $ni^2$ vertices:

- the matrix bandwidth is order $O(ni)$.
- the computer work is $O(ni^4)$ and storage is $O(ni^3)$ for banded or profile elimination.
- the computer work is $O(ni^3)$ and storage is $O(ni^2 \log ni)$ for sparse elimination.

For 3D grids, with $ni^3$ vertices:

- the matrix bandwidth is order $O(ni^2)$.
- the computer work is $O(ni^7)$ and storage is $O(ni^5)$ for banded or profile elimination.
- the computer work is $O(ni^6)$ and storage is $O(ni^4)$ for sparse elimination.

Banded solvers were used in the 2D applications in references [25] [27] [16], and [60]. The sparse elimination scheme was used by Wigton in 2D [63]. There is a break-even point such that for smaller problems, the banded methods are faster, and take less storage. But as the problem gets larger, the general sparse methods get relatively better. The 3D work and storage seems prohibitive with either method, but so did the estimate for 2D ten years ago.

For nonsymmetric matrices, the matrix equation is solved by first factoring $\mathbf{A}$ into lower and upper triangular form $\mathbf{A} = \mathbf{LU}$ and solving

$$\mathbf{Lz} = \mathbf{b} \tag{5.13}$$

for $\mathbf{z}$ and

$$\mathbf{Ux} = \mathbf{z} \tag{5.14}$$

for $\mathbf{x}$, which are easy because $\mathbf{L}$ and $\mathbf{U}$ are triangular matrices.

Reference [18] contains a survey of both algorithms and software for solving sparse systems of linear equations. In it are mentioned two software packages: SPARSPAK [23] [12] available from the University of Waterloo and MA28 from the Harwell subroutine libraries and available through netlib over ARPANET. Both MA28 and SPARSPAK are easy to use packages for solving sparse matrices.

MA28 chooses pivots in the factorization process based on stability and sparsity control. It is very robust, but can be slow due to the pivoting process. In early evaluations, it was determined that the pivoting was not required for stability of the matrix solution and the solution times using MA28 were slower than the other methods looked at.

SPARSPAK provides a user with several options in the matrix solution method. It assumes the matrix has a symmetric structure, but can solve both symmetric and nonsymmetric matrices. The choices of method are a profile (also known as envelope or skyline) solver, a general sparse solver, and a partitioned matrix solver. For each method, the matrix is ordered to take advantage of sparsity in the matrix factors $\mathbf{L}$ and $\mathbf{U}$. Different algorithms are used for determining this order. The SPARSPAK license was only available on a micro VAX II. This availability restriction limited its application.

A nonsymmetric profile storage solver (SKYSOL) first written by Tim Prince while still at General Electric and modified and used by the author [60] was also used. It has been extended to allow two ordering schemes and is coupled with GMRES which is discussed in Section 5.3.2. Because it allows for a nonsymmetric profile of the factored matrix, it can be more efficient than SPARSPAK for certain classes of problems. A comparison between SKYSOL and SPARSPAK is presented in the next chapter for a 2D duct using different grids. For all the other flow calculation results presented, SKYSOL was used because the SPARSPAK license was not available on the GE CRAY.

The ordering of a matrix can be represented by a permutation matrix $\mathbf{P}$ (see [50]), which is a square matrix of order $n$ such that each row or column contains one element equal to 1, and the rest 0. $\mathbf{P}$ can be stored in the computer as an array of integers. A property of $\mathbf{P}$ is that $\mathbf{P}^{\mathbf{T}}\mathbf{P} = \mathbf{I}$, the identity matrix.

To solve the system $\mathbf{Ax} = \mathbf{b}$, the matrix is reordered using the permutation matrix $\mathbf{P}$ such that

$$\mathbf{PAP}^{\mathbf{T}}\mathbf{Px} = \mathbf{Pb}, \tag{5.15}$$

is an equivalent system. If $\mathbf{y} = \mathbf{Px}$, $\mathbf{c} = \mathbf{Pb}$ and $\mathbf{B} = \mathbf{PAP}^{\mathbf{T}}$, then the equivalent system is

$$\mathbf{By} = \mathbf{c}. \tag{5.16}$$

The permutation matrix is chosen so the matrix factors of $\mathbf{B}$ are as sparse as possible.

The algorithms available in SPARSPAK for choosing the permutation matrix $\mathbf{P}$ are all based on increasing sparsity of a matrix with symmetric structure. These include:

1. the Reverse Cuthill-Mckee (RCM) ordering which uses the profile solver,
2. the Minimum Degree ordering which uses the general sparse solver,
3. the Nested Dissection ordering which uses the general sparse solver,
4. One-Way Dissection ordering which uses the partitioned matrix solver, and
5. the Refined Quotient Tree ordering which also uses the partitioned matrix solver.

The SKYSOL orderings include:

1. a natural ordering which orders the unknowns depending on the node number of the structured grid, and
2. an RCM ordering which uses some of the code listed in reference [23] and is also part of SPARSPAK. The code in SKYSOL which sets up the required adjacency structure (this represents the graph of the matrix with symmetric structure) is

not very efficient and therefore produces a slower ordering than SPARSPAK, even though the resulting ordering is the same.

It is noteworthy that a nested dissection ordering is described by Wigton [63] which uses knowledge of the actual computational grid. Therefore, if a general sparse solver is used with this ordering, the ordering could be more efficient for a given problem rather than using the more general techniques available in SPARSPAK.

In the following subsection, the structure of the matrix for this application will be discussed which demonstrates why the nonsymmetric storage method is useful. Following this, the use of GMRES for reusing matrix factors and increasing the numerical accuracy of a matrix solution will be discussed.

### 5.3.1 Matrix Structure

An ordering of the unknowns of the reduced matrix is

$$\mathbf{x} = \begin{bmatrix} \psi_1 \\ \psi_2 \\ P_S \end{bmatrix}, \tag{5.17}$$

where $\boldsymbol{\psi_1}$, $\boldsymbol{\psi_2}$ and $\mathbf{P_S}$ are the vectors of $\psi_1$, $\psi_2$ and $P_S$ values throughout the grid. The structure of the matrix can best be explained by discussing the components of the partitioned matrix

$$\mathbf{Ax} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ P_S \end{bmatrix}. \tag{5.18}$$

Excluding boundary conditions, both $\mathbf{A_{11}}$ and $\mathbf{A_{22}}$ have a symmetric structure, and $\mathbf{A_{12}}$ and $\mathbf{A_{21}}^T$ have similar structure (for the same grid size $ni$ and $nj$). However $\mathbf{A_{33}}$ is close to lower triangular due to the nature of the $S$ momentum equation. The structure of $\mathbf{A_{13}}$, $\mathbf{A_{23}}$, $\mathbf{A_{31}}$ and $\mathbf{A_{32}}$ are unrelated.

To optimize the factorization of this matrix, the knowledge of this structure should be exploited. However, this would have made experimentation difficult, so the best general approach has been used. By using a solver which exploits the nonsymmetric structure, the efficiency of the factorization step is increased. Solving a lower triangular matrix with a routine using symmetric storage is adding computational work.

### 5.3.2 Use of GMRES

Newton's method for solving systems of nonlinear equation requires a matrix solution each iteration. As convergence is approached, these matrices become more like the matrix from the preceding iteration. Because the matrix factorization for these problems can take a large portion of the CPU time, it would be desirable to utilize a matrix factor from a previous iteration. This is not a new idea and has been used in reference [54] by using the previous matrix factor and current right hand side. This, however may effect the quadratic convergence rate of Newton's method. A new idea in this thesis is to use the Generalized Minimum Residual (GMRES) algorithm ([55], [9], and [64]) with the previous matrix factor as a preconditioner to obtain the current iteration matrix solution.

A side benefit is that the accuracy of a matrix solution can be improved. Due to round off errors, the matrix factor is actually only an approximate factor and GMRES can be used to improve the matrix solution given an approximate factor.

As described in the abstract of [9], GMRES is a method for accelerating and stabilizing iterative solutions to certain large scale problems. It is an outgrowth of preconditioned conjugate gradient and Krylov subspace methods. It is required that the underlying iterative method, or preconditioner, reduce the effective dimensionality of the solution space to a small number, such as 10 or 20. FORTRAN 77 codes of this algorithm have been written under NSF contract ASC-8519353 by Boeing Computer Services. These routines are documented in [8] and are available in the public domain. The linear code was used in this thesis.

The GMRES code accelerates an iterative solution method. Given an approximate solution vector $\mathbf{y}$, this iterative method must return an improved solution vector $\mathbf{M}(\mathbf{y})$ where $\mathbf{M}$ is a linear operator. For the iterative method

$$\mathbf{y}^{n+1} = \mathbf{J}\mathbf{y}^{n} + \mathbf{d}, \tag{5.19}$$

$\mathbf{M}$ has the form

$$\mathbf{M}(\mathbf{y}) = \mathbf{J}\mathbf{y} + \mathbf{d}. \tag{5.20}$$

Both $\mathbf{J}\mathbf{y}$ and $\mathbf{M}(\mathbf{y})$ must be returned by the preconditioner subroutine.

An iterative method can be derived for solving the permuted matrix system $\mathbf{B}\mathbf{y} = \mathbf{c}$ using an approach similar to iterative refinement or the error equations [35].

Suppose that given a solution vector $\mathbf{y}^{n}$, a better solution is desired so that

$$\mathbf{y}^{n+1} = \mathbf{y}^{n} + \delta\mathbf{y} \tag{5.21}$$

and

$$\mathbf{B}(\mathbf{y}^{n+1}) = \mathbf{B}(\mathbf{y}^{n} + \delta\mathbf{y}) = \mathbf{c} \tag{5.22}$$

is satisfied. Therefore

$$\mathbf{B}\delta\mathbf{y} = \mathbf{c} - \mathbf{B}\mathbf{y}^{n}. \tag{5.23}$$

Now the approximation comes in. To solve for $\delta\mathbf{y}$, rather than using $\mathbf{B} = \mathbf{L}\mathbf{U}$, use a previous matrix factor $\hat{\mathbf{L}}\hat{\mathbf{U}}$ so

$$\mathbf{B} = \mathbf{L}\mathbf{U} \approx \hat{\mathbf{L}}\hat{\mathbf{U}}. \tag{5.24}$$

Therefore,

$$\delta\mathbf{y} = \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{c} - \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{B}\mathbf{y}^{n}. \tag{5.25}$$

For this to fit in as a GMRES preconditioner,

$$\begin{aligned} \mathbf{y}^{n+1} &= \mathbf{y}^{n} + \delta\mathbf{y} = \mathbf{y}^{n} + \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{c} - \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{B}\mathbf{y}^{n} \\ &= \left[\mathbf{I} - \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{B}\right]\mathbf{y}^{n} + \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{c} \\ &= \mathbf{J}\mathbf{y}^{n} + \mathbf{d}. \end{aligned} \tag{5.26}$$

Therefore

$$\mathbf{J} = \mathbf{I} - \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{B} \tag{5.27}$$

and

$$\mathbf{d} = \hat{\mathbf{U}}^{-1}\hat{\mathbf{L}}^{-1}\mathbf{c}. \tag{5.28}$$

This means **d** is the solution of the matrix equation

$$\hat{\mathbf{L}}\hat{\mathbf{U}}\mathbf{d} = \mathbf{c}, \tag{5.29}$$

and $\mathbf{y}^{n+1} = \mathbf{y}^n + \delta\mathbf{y}$ is determined by solving

$$\hat{\mathbf{L}}\hat{\mathbf{U}}\delta\mathbf{y} = \mathbf{c} - \mathbf{B}\mathbf{y}^n, \tag{5.30}$$

for $\delta\mathbf{y}$ and adding it to $\mathbf{y}^n$. The derivative **Jy** is also needed. This is determined by solving for $\delta\mathbf{y}$ and applying

$$\mathbf{J}\mathbf{y} = \mathbf{y}^n + \delta\mathbf{y} - \mathbf{d}. \tag{5.31}$$

Once a converged solution **y** is determined,

$$\mathbf{x} = \mathbf{P}^{-1}\mathbf{y}, \tag{5.32}$$

which is an inverse ordering of the array.

To ensure the matrices are as similar as possible from one Newton iteration to the next, the matrix is normalized. Each row is normalized by its diagonal.

## 5.4 Ensuring a Strong Diagonal — Use of MC21A

The matrix solution methods are faster if they do not need pivoting to provide numerical stability. Numerical stability is not a problem for symmetric positive definite matrices or diagonally dominant matrices. To reduce the problem of numerical stability for matrices that are not one of these types, a strong diagonal (the diagonal element is large relative to the other elements of the matrix) is required.

The general equation-variable match-up has been discussed. For the interior unknowns and most of the boundary unknowns, the equation used ensures that the diagonal is strong. However at some boundaries, the associated equations do not contain the unknown at all. Some reordering of the equations is therefore required which depends on the choice of boundary condition. A general approach for this reordering is the program MC21A. The algorithm is in reference [19], and the program listing is in [20]. It is a subroutine in the MA28 package.

Basically, MC21A provides the permutation to produce a zero-free diagonal. It is supplied with a pattern of possible diagonals for each equation. This routine has worked very well and allowed generality in boundary condition implementations.

## 5.5 Globally Convergent Methods

One of the problems with Newton's method is that it is not globally convergent. Depending on the initial solution and the nonlinearity of the problem, the method may not converge. Two strategies have been employed in this thesis.

The first strategy is to simply specify a damping factor $\lambda$ which is a constant fraction of the Newton update which will be applied:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda \mathbf{s}_k. \tag{5.33}$$

This factor is specified either for each iteration, or as a constant until convergence is achieved.

The second strategy uses a line search algorithm discussed in reference [15]. Basically, the full Newton step is used if the $l_2$ error norm is reduced. That is

$$||\mathbf{F}(\mathbf{x}_{k+1})|| < ||\mathbf{F}(\mathbf{x}_k)||. \tag{5.34}$$

Otherwise $\lambda$ within the range 0 to 1 is determined such that

$$f(\mathbf{x}_{k+1}) = \frac{1}{2}||\mathbf{F}(\mathbf{x}_{k+1})|| = \frac{1}{2}||\mathbf{F}(\mathbf{x}_k + \lambda \mathbf{s}_k)|| \tag{5.35}$$

is approximately minimized by backtracking $\lambda$ and

$$f(\mathbf{x}_k + \lambda \mathbf{s}_k) \leq f(\mathbf{x}_k) + \alpha \lambda \nabla f(\mathbf{x}_k)^T \cdot \mathbf{s}_k, \tag{5.36}$$

is satisfied. This is similar to Eq. (5.34) except it eliminates the possibility of having very small decreases in $f$ relative to the step size. The gradient of $f$ can be determined from the Jacobian and vector of residuals

$$\nabla f(\mathbf{x}_k) = \mathbf{J}(\mathbf{x}_k)^T \mathbf{F}(\mathbf{x}_k), \tag{5.37}$$

and $\alpha$ is a constant between 0 and 1. The expression $f(\mathbf{x}_k + \lambda \mathbf{s}_k)$ is a function of $\lambda$ which is approximated by a quadratic or cubic polynomial and this polynomial is minimized. The directional derivative, $f'_{\mathbf{s}}(\mathbf{x}_k)$ at $\lambda = 0$ is used which is the steepest descent direction dotted into the Newton direction:

$$f'_{\mathbf{s}}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)^T \cdot \mathbf{s}_k. \tag{5.38}$$

The line search algorithm in reference [15] has been used which is outlined below:

1. Check that $f'_{\mathbf{s}}(\mathbf{x}_k) < 0$ to ensure that $\mathbf{s}_k$ is a descent direction. If not, terminate with an error flag.
2. Set $\lambda = 1$. Set $\alpha < \frac{1}{2}$ ($\alpha = 10^{-4}$ is used).
3. Check that $\mathbf{x}_+ = \mathbf{x}_k + \lambda \mathbf{s}_k$ is satisfactory (i.e., Eq. (5.36) is satisfied). If so, terminate. If not, continue.
4. Decrease $\lambda$ by a factor between 0.1 and 0.5 (backtrack) as follows:
   (a) On the first backtrack: select $\lambda$ such that $\mathbf{x}_+$ is the minimizer of the 1D quadratic which interpolates $f(\mathbf{x}_k)$, $f'_{\mathbf{s}}(\mathbf{x}_k)$ and $f(\mathbf{x}_k + \mathbf{s}_k)$, but constrain $\lambda \geq 0.1$.
   (b) On all subsequent backtracks: select the new $\lambda$ such that $\mathbf{x}_+$ is the minimizer of the 1D cubic which interpolates $f(\mathbf{x}_k)$, $f'_{\mathbf{s}}(\mathbf{x}_k)$, $f(\mathbf{x}_+(\lambda))$ and $f(\mathbf{x}_+(\lambda_{\text{prev}}))$, but constrain $\lambda$ to be in $[0.1 \times \text{old}\lambda, 0.5 \times \text{old}\lambda]$.
5. Return to step 3.

# Chapter 6
# 2D Results

This chapter contains results for some 2D problems. The 3D algorithm reduces to 2D when $nj = 2$ and the rotation speed is set to zero. The grid is the same at $j = 1$ and $j = 2$ and separated by some $\Delta y$ with $\psi_2 = 0$ on the lower wall and $\psi_2 = 1$ on the upper wall. The mass flux is then $\psi_1|_{max}$. The same code is used to run the 2D or 3D problems.

These 2D results are intended to demonstrate the accuracy of the discretization. Because of the large grid sizes needed to demonstrate accuracy, this is done in 2D rather than in 3D. This accuracy study is performed using a duct with a $\sin^2(\pi z)$ bump as a test case. Convergence histories and a comparison of matrix solution methods are also presented for this test case.

Several 2D cascade cases are presented to demonstrate the application of the method. These include the incompressible Gostelow cascade, the T7 turbine nozzle, and the Garabedian transonic compressor cascade. The design option capability is also presented. A blade shape is produced given a NACA 0012 thickness distribution and an arbitrary pressure distribution and inlet angle.

In each case run, the grid is generated using the elliptic grid generator in ISES [27]. The $\psi_1$ values are initially distributed based on the grid stretching and are constant along $\xi^3$ grid lines. The pressure field is initialized by calculating the mass flux vector and assuming the flow field is isentropic. This is a good initial guess to start the Newton iterations. For a grid with a blunt leading edge, the grid moves around the leading edge in an accordion-like fashion as described by Drela [16]. Checks have been added so that if the grids cross over or otherwise become distorted after an iteration, then the grid is regenerated using the elliptic grid generator.

## 6.1 Duct with $\sin^2(\pi x)$ bump

To investigate the accuracy of the method and demonstrate the matrix solution methods used, the solution of the flow through a duct with a $\sin^2(\pi x)$ bump has been obtained with several grids. The geometry is specified as:

$$\text{the lower wall } z_l = \begin{cases} 0.1\sin^2(\pi x) & ; \quad 0 < x < 1 \\ 0 & ; \quad \text{otherwise} \end{cases} \tag{6.1}$$

$$\text{the upper wall } z_u = \begin{cases} 1 - 0.1\sin^2(\pi x) & ; \quad 0 < x < 1 \\ 1 & ; \quad \text{otherwise} \end{cases} \tag{6.2}$$

At the boundaries: $x_{inlet} = -1$; $x_{exit} = 2$; and $M_{inlet} = 0.5$. The total pressure is constant upstream.

The grid has been generated with uniform spacing at the boundaries, and $n$ grid intervals per unit length. Fig. 6.1 shows the duct and grid for $n = 16$. Fig. 6.2 shows contours of Mach number and Fig. 6.3 shows the Mach number distributions on the walls of the duct for the $n = 16$ case. Solutions have been obtained with $n = 4, 8, 12, 16, 24, 32$ and 40.

Table 6.1 shows the iteration history of the residual error norm for 5 iterations where

$$\text{residual error norm} = \left( \frac{\sum_{i=1}^{nnleq} (f_i)^2}{nnleq} \right)^{\frac{1}{2}}, \tag{6.3}$$

and $nnleq$ is the number of nonlinear equations. A CRAY XMP 2/16 was used to obtain these results which allowed 64 bit precision to be used. Notice the rapid convergence rate, and that the convergence is independent of grid size. Machine accuracy is obtained in 4 iterations. On a 32 bit machine, such as a VAX, machine accuracy is at $10^{-7}$, which would be achieved in 2 iterations.

The exact solution of this subsonic inviscid flowfield is isentropic. To obtain a measure of the accuracy of a solution, an error and root mean square (rms) error are

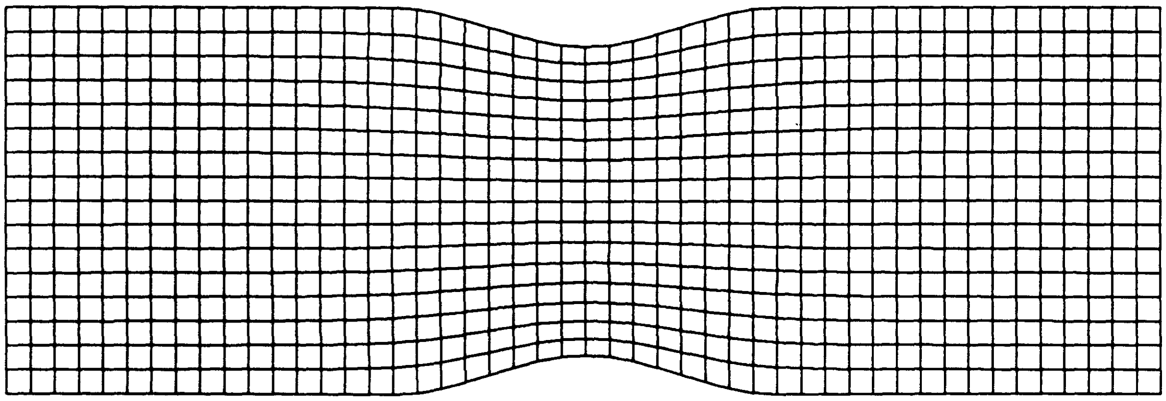


Figure 6.1: Geometry and grid for $\sin^2(\pi x)$ bump for $n = 16$, $ni = 17$, $nk = 49$.

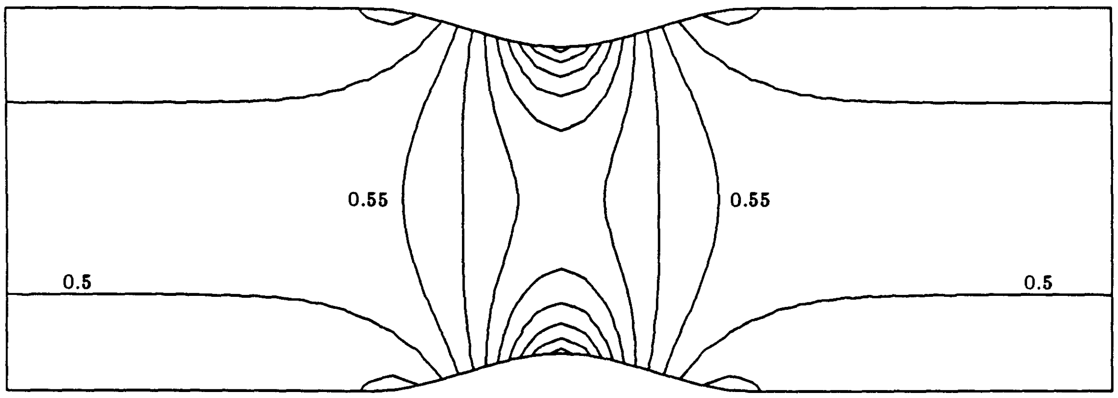


Figure 6.2: Mach number contours for $\sin^2(\pi x)$ bump for $n = 16$, $ni = 17$, $nk = 49$. Contour intervals are 0.05.

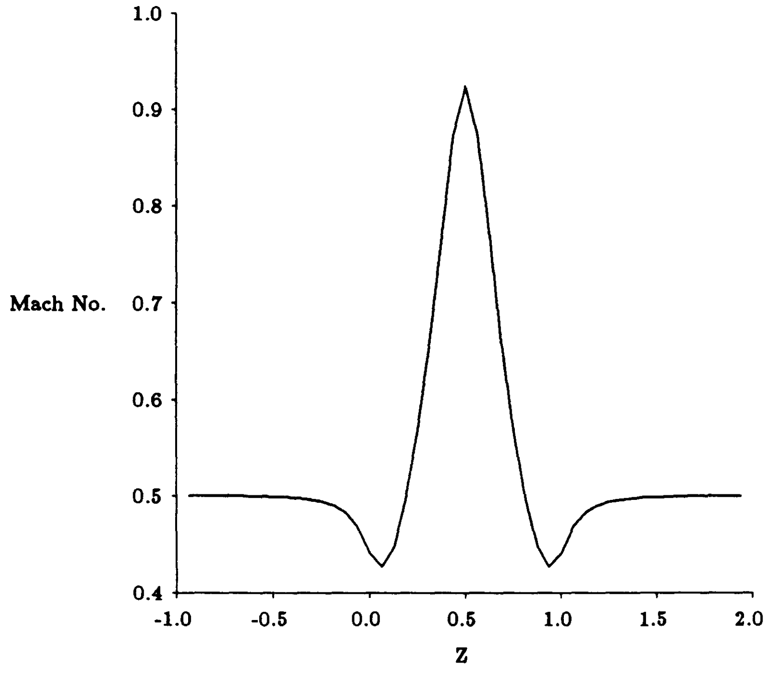


Figure 6.3: Mach number distributions on $\sin^2(\pi z)$ bump duct walls for $n = 16$, $ni = 17$, $nk = 49$.

Table 6.1: Iteration history. Residual error norm for 5 iterations on a CRAY XMP 2/16 for different grids.

| | n | | | | | | |
|---|---|---|---|---|---|---|---|
| iter | 4 | 8 | 12 | 16 | 24 | 32 | 40 |
| 0 | 1.571e-3 | 2.188e-3 | 9.888e-4 | 9.560e-4 | 5 510e-4 | 4.099e-4 | 3.579e-4 |
| 1 | 6.276e-6 | 2.301e-5 | 6.415e-6 | 1.406e-5 | 7.704e-6 | 5.031e-6 | 3.649e-6 |
| 2 | 1.097e-9 | 2.124e-7 | 2.367e-8 | 4.557e-7 | 2.683e-7 | 1.521e-7 | 1.100e-7 |
| 3 | 1.688e-15 | 2.273e-11 | 2.744e-13 | 3.834e-11 | 7.793e-11 | 7.600e-11 | 4.799e-11 |
| 4 | 1.707e-15 | 1.690e-15 | 1.834e-15 | 1.841e-15 | 1.847e-15 | 1.889e-15 | 1.892e-15 |
| 5 | 1.291e-15 | 1.561e-15 | 1.523e-15 | 1.508e-15 | 1.669e-15 | 1.660e-15 | 1.732e-15 |

Table 6.2: RMS and maximum entropy errors for different grid sizes.

| n | rms error | max error |
|---|---|---|
| 4 | 4.894e-4 | 1.649e-3 |
| 8 | 4.540e-4 | 2.880e-3 |
| 12 | 2.444e-4 | 1.965e-3 |
| 16 | 1.485e-4 | 1.348e-3 |
| 24 | 6.934e-5 | 7.120e-4 |
| 32 | 3.914e-5 | 4.279e-4 |
| 40 | 2.484e-5 | 2.847e-4 |

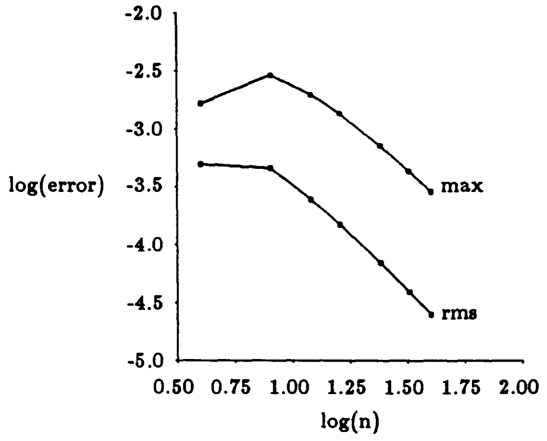


Figure 6.4: Log-log plot of the rms and maximum entropy errors as a function of grid size.

defined as

$$err_i = \frac{s}{R}|_i - \frac{s}{R}|_{inlet}, \tag{6.4}$$

$$\text{rms entropy error} = \left( \frac{\sum_{i=1}^{nsf} (err_i)^2}{nsf} \right)^{\frac{1}{2}}, \tag{6.5}$$

where $\frac{s}{R}$ is the entropy, and $nsf$ is the number of $S$ faces. The rms error and maximum entropy errors are tabulated in Table 6.2. These are plotted on a log-log scale in Fig. 6.4. The interval slopes of the log-log plot are tabulated in Table 6.3. The slopes approach two as the grid is refined demonstrating that the discretization is second-order accurate. Also, the overall level of error is low even for a coarse grid.

The convergence rate presented in Table 6.1 was obtained by solving the matrix

Table 6.3: Slopes of a log-log plot of the errors as a function of grid size.

| interval n-n | slope of rms error | slope of max error |
|---|---|---|
| 4-8 | 0.110 | -0.804 |
| 8-12 | 1.528 | 0.943 |
| 12-16 | 1.729 | 1.305 |
| 16-24 | 1.880 | 1.579 |
| 24-32 | 1.985 | 1.769 |
| 32-40 | 2.043 | 1.826 |

each iteration, but only factoring the matrix once (at the first iteration). This matrix factor was then used as a preconditioner in GMRES during each of the other iterations. Table 6.4 and Table 6.5 show for $n = 16$, the matrix factorization time and the number of GMRES iterations required during each iteration and the corresponding CPU time for a CRAY XMP 2/16 and a VAX 8700. Notice that on both the CRAY and VAX, the amount of CPU time used by GMRES during an iteration is less than that required to factor the matrix. More time is spent in GMRES during the second iteration because many of the equations are satisfied early in the iteration history, and the effective dimensionality of the problem is reduced as the solution converges. On the CRAY, the solid state disk (SSD) option has been utilized, so there is only a small memory penalty for using GMRES. On the VAX, the virtual memory is used to store the vector of solutions (basically one for every GMRES iteration). On the CRAY, more GMRES iterations are needed. The most robust tolerance has been used in GMRES (i.e. keep iterating until there is no improvement in the solution). Since the matrix is solved to machine accuracy, more GMRES iterations are required on the CRAY, a higher precision machine.

Table 6.6 and Table 6.7 show the factorization time, the number of GMRES iterations and CPU time during iteration 2, the total GMRES time and the total solution time for the CRAY and VAX. The total solution times are given for reference. Little attention has been given to the speed of the code in which the matrix is set. It has been modularized so there are many subroutine calls, most of which were written using SMP. In a production code, the matrix set up time could be greatly reduced by writing

Table 6.4: Matrix factorization time and GMRES number of iterations and CPU time for 5 Newton iterations for n=16 on a CRAY XMP 2/16 (CPU time in seconds).

| iter | factorization time | No. GMRES iterations | GMRES time |
|---|---|---|---|
| 1 | 1.949 | 1 | 0.081 |
| 2 | – | 23 | 0.988 |
| 3 | – | 20 | 0.878 |
| 4 | – | 14 | 0.632 |
| 5 | – | 3 | 0.161 |
| total | 1.949 | 61 | 2.74 |

total time = 58.2 seconds for 5 iterations

matrix factored with nonsymmetric profile solver (SKYSOL) with natural ordering

Table 6.5: Matrix factorization time and GMRES number of iterations and CPU time for 5 Newton iterations for n=16 on a VAX 8700 (CPU time in seconds).

| iter | factorization time | No. GMRES iterations | GMRES time |
|---|---|---|---|
| 1 | 13.37 | 0 | 0.42 |
| 2 | – | 10 | 5.57 |
| 3 | – | 6 | 3.39 |
| 4 | – | 1 | 0.93 |
| 5 | – | 0 | 0.44 |
| total | 13.37 | 17 | 10.75 |

total time = 226.0 seconds for 5 iterations

matrix factored with nonsymmetric profile solver (SKYSOL) with natural ordering

the code more efficiently. The use of finite difference derivatives would also reduce the CPU time for some of the matrix components. The matrix set up time is also a linear function of the number of unknowns, as compared with the matrix solution time, and for larger problems constitutes a smaller percentage of the total time. For a more efficient matrix set up, the matrix factorization would probably be the most time consuming part of the solver, and reducing the number of factorizations could decrease the overall solution times dramatically.

The bump test case was also used to benchmark several matrix solution methods

Table 6.6: Matrix factorization times, GMRES statistics and total solution times for 5 Newton iterations on different grids using a CRAY XMP 2/16 (CPU time in seconds).

| n | iter=1 factor time | iter=2 No. GMRES iter | iter=2 GMRES time | total GMRES time | total time 5 iterations |
|---|---|---|---|---|---|
| 4 | 0.034 | 6 | 0.020 | 0.044 | 3.76 |
| 8 | 0.251 | 12 | 0.134 | 0.404 | 13.98 |
| 12 | 0.835 | 15 | 0.363 | 0.939 | 31.7 |
| 16 | 1.967 | 23 | 1.016 | 2.78 | 58.2 |
| 24 | 6.623 | 28 | 2.837 | 7.544 | 138.4 |
| 32 | 15.47 | 30 | 5.742 | 25.33 | 246.3 |
| 40 | 30.68 | 31 | 8.856 | 23.75 | 390.5 |

matrix factored with nonsymmetric profile solver (SKYSOL) with natural ordering

Table 6.7: Matrix factorization times, GMRES statistics and total solution times for 5 Newton iterations on different grids using a VAX 8700 (CPU time in seconds).

| n | iter=1 factor time | iter=2 No. GMRES iter | iter=2 GMRES time | total GMRES time | total time 5 iterations |
|---|---|---|---|---|---|
| 4 | 0.11 | 2 | 0.06 | 0.14 | 20.5 |
| 8 | 1.06 | 7 | 0.68 | 1.34 | 57.2 |
| 12 | 4.61 | 6 | 1.66 | 2.98 | 123.5 |
| 16 | 13.46 | 10 | 5.54 | 10.19 | 226.0 |
| 24 | 62.97 | 12 | 19.02 | 36.6 | 548.9 |
| 32 | 191.53 | 12 | 40.72 | 81.9 | 1090.5 |
| 40 | 459.07 | 12 | 76.76 | 159.5 | 1903.7 |

matrix factored with nonsymmetric profile solver (SKYSOL) with natural ordering including all those available in SPARSPAK. These include:

1. the nonsymmetric profile storage solver (SKYSOL) with a natural ordering,
2. the nonsymmetric profile storage solver (SKYSOL) with Reverse Cuthill-Mckee (RCM) ordering,
3. SPARSPAK with RCM ordering,
4. SPARSPAK with Minimum Degree ordering,
5. SPARSPAK with Nested Dissection ordering,
6. SPARSPAK with One-Way Dissection ordering, and
7. SPARSPAK with Refined Quotient Tree ordering.

The SPARSPAK license was only available on a Micro VAX II, so all runs were made on this machine. Grids with $n = 4, 8, 12, 16, 24$ and 32 were run. The Micro VAX II did not have a large enough page size to run the $n = 40$ grid, and with several methods, the matrix for $n = 32$ could not be factored. The results are shown in Table 6.8 and Table 6.9. These tables provide information about the relative merits of these factorization methods which are summarized below:

- The number of nonzero elements is larger when symmetry is required compared to when it is not (NZs in Table 6.8 compared with NZs in Table 6.9). This demonstrates how the structure of the matrix is far from symmetric.
- Comparing the two ordering schemes using SKYSOL in Table 6.8, it can be seen that the factor time and storage are less with the RCM ordering. However it does take longer to order the matrix using RCM.
- As the grid is resolved, the minimum degree ordering using SPARSPAK is the most efficient as shown in Table 6.9.
- The work (factor time) and storage approach the theoretical values set out in the previous chapter. That is, for a 2D grid, with $ni^2$ vertices:

- the computer work is $O(ni^4)$ and storage is $O(ni^3)$ for banded or profile elimination such as with SKYSOL or RCM ordering with SPARSPAK.
- the computer work is $O(ni^3)$ and storage is $O(ni^2 \log ni)$ for sparse elimination such as with minimum degree and nested dissection.

- SPARSPAK preserves a symmetric storage whereas SKYSOL allows for a nonsymmetric structure. The ordering is identical for RCM using SKYSOL or SPARSPAK. There is improvement in storage and factor time using SKYSOL where the nonsymmetric structure is exploited.

The CPU time and storage to solve a matrix directly is greatly dependent on the solver and ordering used. For large grids, the general sparse methods are very promising as demonstrated by the minimum degree ordering.

If the leading edge is allowed to move, then this additional variable has a significant impact on the ordering. Table 6.10 shows a comparison between the matrix solutions obtained with and without the leading edge movement for a $106 \times 21$ incompressible case similar to that shown in the next section. Natural and RCM orderings are used with SKYSOL as well as the RCM ordering with SPARSPAK. Notice how the RCM ordering does not work well with both SKYSOL and SPARSPAK when the leading edge is allowed to move, whereas there is little difference for the natural ordering. In ISES [25] [27] [16], this leading edge variable is one of the global variables which is partitioned from the matrix. This difference in ordering would not be as dramatic if the global variable approach had been used.

Table 6.8: Comparison of different orderings using a nonsymmetric profile storage matrix solver (SKYSOL) for matrices obtained on several grids. Solved on a Micro VAX II (CPU time in seconds).

| | | n | | | | | |
|---|---|---|---|---|---|---|---|
| Ordering | Description | 4 | 8 | 12 | 16 | 24 | 32 |
| | NEQ (a) | 217 | 817 | 1801 | 3169 | 7057 | 12481 |
| | NZs (b) | 1793 | 8065 | 18839 | 34112 | 78233 | 140331 |
| Natural | Tot Stor (c) | 6449 | 45039 | 144573 | 333851 | 1096839 | 2564403 |
| | NZs in **L** (d) | 3248 | 24064 | 78576 | 182912 | 605568 | 1421056 |
| | NZs in **U** (e) | 1899 | 16073 | 55191 | 131925 | 448929 | 1068461 |
| | Order Time (f) | 0.01 | 0.01 | 0.05 | 0.07 | 0.14 | 0.25 |
| | Factor Time (g) | 0.59 | 6.89 | 30.58 | 89.88 | 420.0 | 1295. |
| | Solve Time (h) | 0.04 | 0.29 | 0.87 | 2.00 | 6.44 | 25.69 |
| RCM | Tot Stor (c) | 5826 | 39641 | 128332 | 295718 | 970600 | 2267666 |
| | NZs in **L** (d) | 2561 | 18228 | 58107 | 134163 | 440115 | 1028232 |
| | NZs in **U** (e) | 1963 | 16511 | 59419 | 142541 | 488143 | 1164548 |
| | Order Time (f) | 0.59 | 3.16 | 7.71 | 14.84 | 36.49 | 68.84 |
| | Factor Time (g) | 0.49 | 5.98 | 26.21 | 75.70 | 347.8 | 1050. |
| | Solve Time (h) | 0.03 | 0.30 | 0.95 | 2.19 | 7.27 | 25.07 |

(a) Dimension of matrix.
(b) Number of nonzeros in matrix.
(c) Total storage: matrix factor, permutation, inverse permutation, right hand side, solution and temporary vectors.
(d) Number of nonzeros in lower matrix factor.
(e) Number of nonzeros in upper matrix factor.
(f) Time to order matrix (i.e. set up the permutation vector).
(g) Time to factor the matrix.
(h) Time to do forward and back-substitution to solve for right hand side.

Table 6.9: Comparison of different orderings using SPARSPAK for matrices obtained on several grids. Solved on a Micro VAX II (CPU time in seconds).

| | | n | | | | | |
|---|---|---|---|---|---|---|---|
| Ordering | Description | 4 | 8 | 12 | 16 | 24 | 32 |
| | NEQ (a) | 217 | 817 | 1801 | 3169 | 7057 | 12481 |
| | NZs (b) | 2915 | 12899 | 29979 | 54135 | 123815 | 221779 |
| RCM | Tot Stor (c) | 8667 | 60213 | 196673 | 454489 | 1494973 | 3496367 |
| | Order Time (d) | 0.12 | 0.43 | 1.03 | 1.94 | 4.46 | – |
| | Factor Time (e) | 0.62 | 7.84 | 37.56 | 113.2 | 548.4 | – |
| | Solve Time (f) | 0.07 | 0.47 | 1.52 | 3.52 | 12.45 | – |
| Minimum Degree | Tot Stor (c) | 8529 | 45950 | 125857 | 246576 | 637952 | 1254524 |
| | Order Time (d) | 0.66 | 2.75 | 6.40 | 11.58 | 26.88 | 49.15 |
| | Factor Time (e) | 0.82 | 9.53 | 47.36 | 115.4 | 435.6 | 1155. |
| | Solve Time (f) | 0.07 | 0.41 | 1.17 | 2.34 | 6.19 | 12.64 |
| Nested Dissection | Tot Stor (c) | 9738 | 63464 | 165944 | 374983 | 1029673 | 2323222 |
| | Order Time (d) | 0.24 | 1.30 | 3.86 | 7.50 | 20.82 | 41.65 |
| | Factor Time (e) | 1.20 | 22.42 | 82.95 | 288.9 | 1287. | 4138. |
| | Solve Time (f) | 0.08 | 0.58 | 1.57 | 3.55 | 10.05 | 30.93 |
| One-Way Dissection | Tot Stor (c) | 9107 | 61339 | 202627 | 464891 | 1550601 | 3623861 |
| | Order Time (d) | 0.16 | 0.78 | 1.81 | 3.33 | 7.90 | – |
| | Factor Time (e) | 0.59 | 20.30 | 113.1 | 376.3 | 1906. | – |
| | Solve Time (f) | 0.07 | 0.83 | 2.89 | 6.93 | 25.39 | – |
| Refined Quotient Tree | Tot Stor (c) | 7411 | 42465 | 123989 | 270385 | 831593 | 1873413 |
| | Order Time (d) | 0.12 | 0.45 | 1.00 | 1.76 | 4.04 | 7.23 |
| | Factor Time (e) | 0.90 | 10.57 | 42.25 | 120.0 | 517.1 | 1490. |
| | Solve Time (f) | 0.09 | 0.55 | 1.59 | 3.53 | 10.57 | 26.98 |

(a) Dimension of matrix.
(b) Number of nonzeros in matrix. Symmetry is preserved in SPARSPAK.
(c) Total storage.
(d) Time to order matrix (i.e. set up the permutation vector).
(e) Time to factor the matrix.
(f) Time to do forward and back-substitution to solve for right hand side.

Table 6.10: Comparison of different orderings using SKYSOL and SPARSPAK for a 106 × 21 case with and without leading edge movement. Solved on a Micro VAX II (CPU time in seconds).

| Method and Ordering | Description | with LE movement | without LE movement |
|---|---|---|---|
| | NEQ (a) | 8566 | 8566 |
| | NZs (b) | 153834 | 150721 |
| | NZs Sym (c) | 254902 | 248702 |
| SKYSOL Natural | Tot Stor (d) | 1,161,082 | 1,150,675 |
| | NZs in L (e) | 647068 | 640771 |
| | NZs in U (f) | 462618 | 458508 |
| | Order Time (g) | 0.08 | 0.08 |
| | Factor Time (h) | 402.67 | 395.78 |
| | Solve Time (i) | 7.47 | 7.41 |
| SKYSOL RCM | Tot Stor (d) | 13,572,650 | 1,056,017 |
| | NZs in L (e) | 2,958,185 | 511528 |
| | NZs in U (f) | 10,563,069 | 493093 |
| | Order Time (g) | 108.24 | 69.91 |
| | Factor Time (h) | - | 346.82 |
| | Solve Time (i) | - | 7.46 |
| SPARSPAK RCM | Tot Stor (d) | >10,000,000 (j) | 1,645,573 |
| | Order Time (g) | 25.29 | 7.82 |
| | Factor Time (h) | - | 516.93 |
| | Solve Time (i) | - | 13.9 |

(a) Dimension of matrix.
(b) Number of nonzeros in matrix.
(c) Number of nonzeros in matrix if symmetry is preserved.
(d) Total storage: matrix factor, permutation, inverse permutation, right hand side, solution and temporary vectors.
(e) Number of nonzeros in lower matrix factor.
(f) Number of nonzeros in upper matrix factor.
(g) Time to order matrix (i.e. set up the permutation vector).
(h) Time to factor the matrix.
(i) Time to do forward and back-substitution to solve for right hand side.
(j) The print field limits this to less than 10,000,000 in SPARSPAK.

## 6.2 Incompressible Gostelow Cascade

This test case geometry was derived by Gostelow [28] using an analytic conformal transformation method. It is an incompressible cascade with a stagger angle of 37.5°, and an inlet angle of 53.5° (relative to axial). The pitch to chord ratio is given as 0.9901573 so the pitch to axial chord ratio is 1.24807.

The incompressible solution was obtained on a $22 \times 129$ grid shown in Fig. 6.5. A blowup of the grid showing the leading edge with the stagnation streamline is shown in Fig. 6.6. Recall that the only grid line which is actually a streamline is the stagnation streamline grid line. Because an elliptic grid generator is used and the grid has been readjusted during the iteration history, the grid lines are very close to being streamlines for this incompressible flow solution.

Table 6.11 shows the convergence history for this run. Because the flow is incompressible, the total pressure is uniform throughout the flowfield. Therefore, the entropy convection equation has been used instead of the $S$ momentum equation to ensure a uniform total pressure. Notice that the line search algorithm automatically damped the solution during the first two iterations. Also notice that the matrix was factored after the fourth iteration, and GMRES used the factor from the fourth iteration to obtain the matrix solution for iterations 5-10. The solution converged to machine tolerance after 8 iterations. At the tenth iteration, the line search algorithm could no longer decrease the residual norm so the program stopped. This solution took a total of 447.3 CRAY seconds.

Figure 6.7 shows the calculated pressure coefficient distribution along the blade surface and compares this with the analytic results presented in reference [28]. The comparison is excellent. A contour plot of the pressure coefficient is shown in Fig. 6.8.

The exit flow angle is established by the imposed Kutta condition. Using the present method, it has been calculated to be 30.089°. In reference [27], a value of 30.06° was calculated using ISES, and this compares to the analytic value of 30.025°.

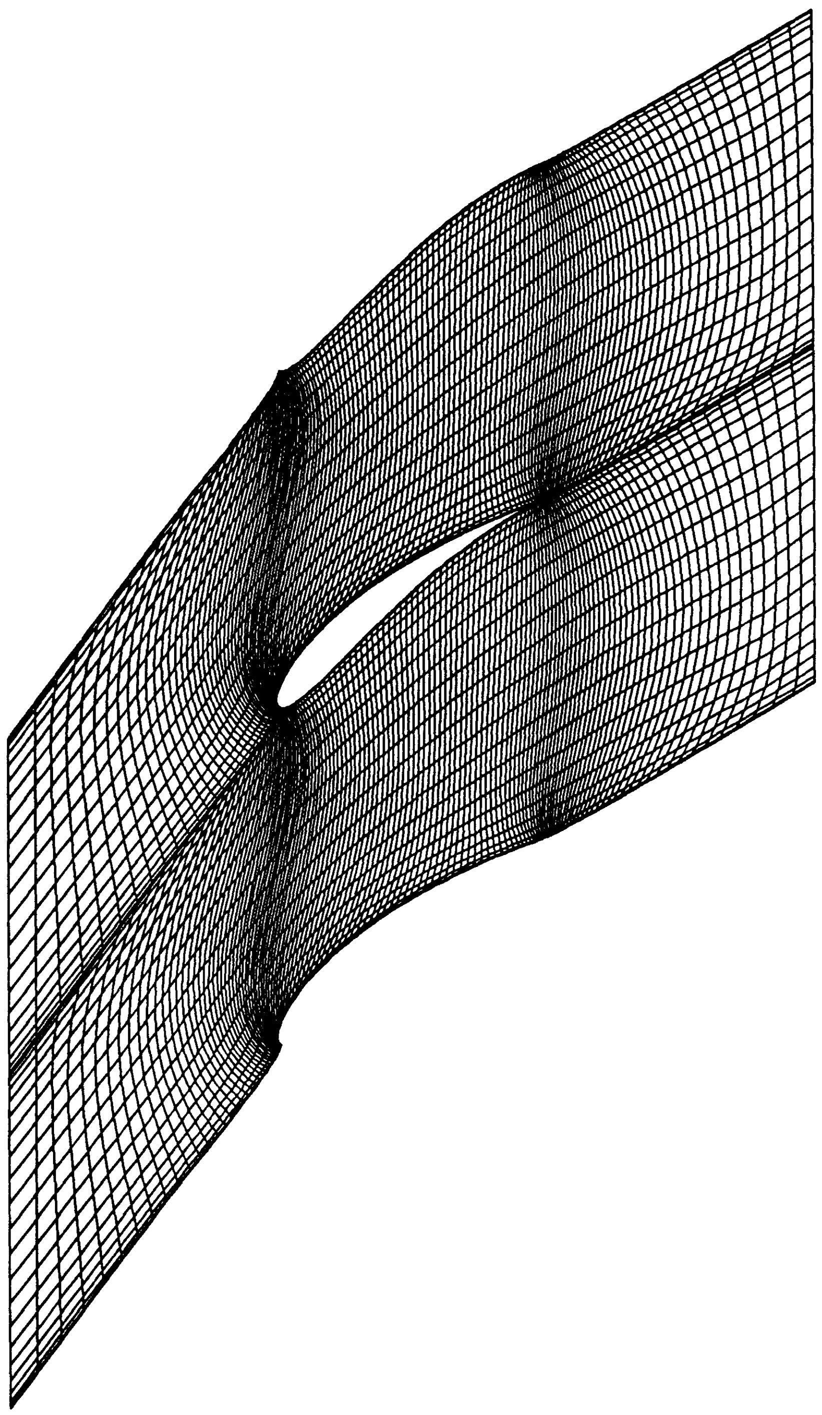


Figure 6.5: Grid for the Gostelow test case. ni = 22, nk = 129.

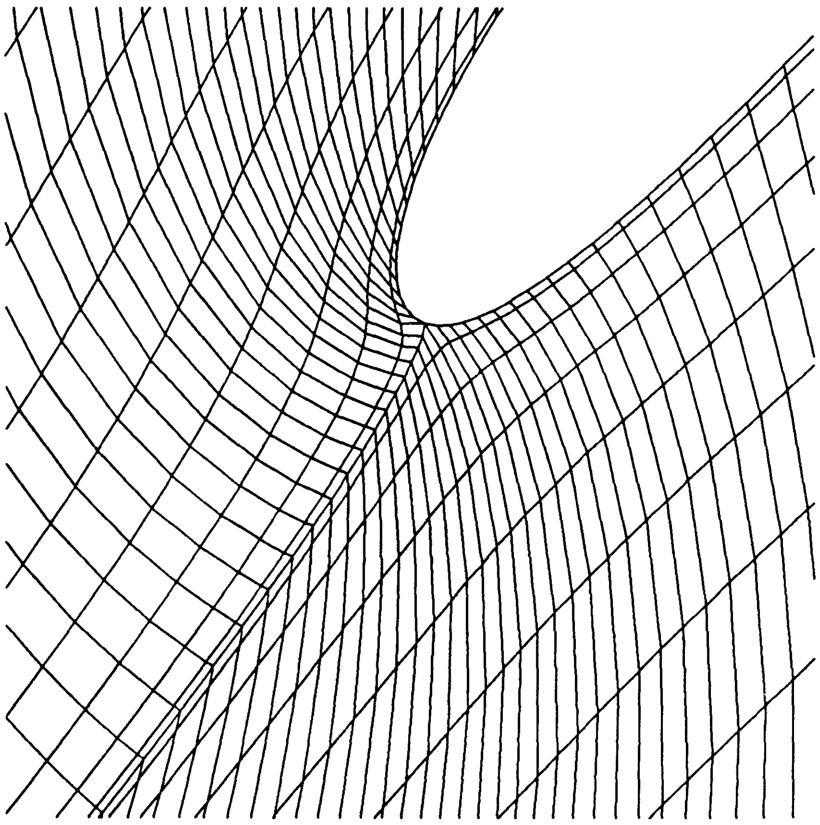


Figure 6.6: Blowup of grid about leading edge for the Gostelow test case. ni = 22, nk = 129.

Table 6.11: Convergence history for the Gostelow test case. Incompressible flow, entropy convection equation applied, ni=22, nk=129, GMRES max cycles = 15, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 1.14e-4 | - | |
| 1 | 7.29e-5 | 0.354 | matrix factored |
| 2 | 4.61e-5 | 0.418 | matrix factored |
| 2 | 7.54e-4 | - | grid readjusted |
| 3 | 1.40e-4 | 1.0 | matrix factored |
| 3 | 4.31e-4 | - | grid readjusted |
| 4 | 4.62e-5 | 1.0 | matrix factored |
| 4 | 1.83e-4 | - | grid readjusted |
| 5 | 2.14e-5 | 1.0 | matrix sol. in 12 GMRES cycles |
| 6 | 1.14e-7 | 1.0 | matrix sol. in 13 GMRES cycles |
| 7 | 9.61e-12 | 1.0 | matrix sol. in 11 GMRES cycles |
| 8 | 5.42e-15 | 1.0 | matrix sol. in 7 GMRES cycles |
| 9 | 4.47e-15 | 1.0 | matrix sol. in 4 GMRES cycles |
| 10 | 4.47e-15 | 0. | matrix sol. in 4 GMRES cycles<br>line search could not decrease residual norm |

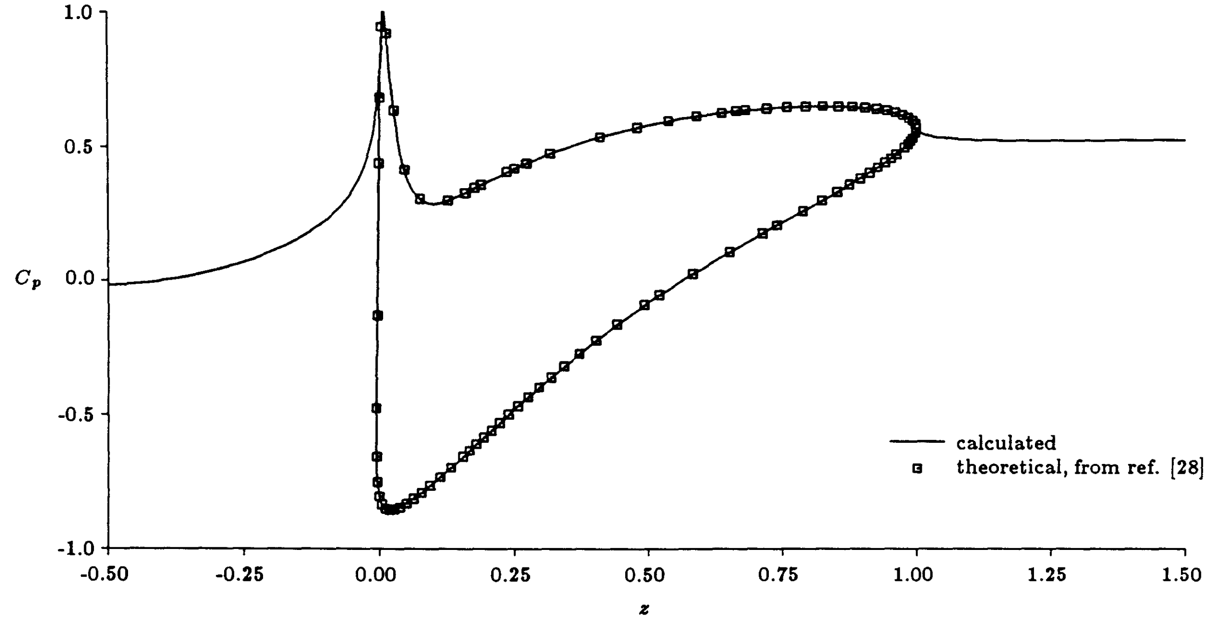


Figure 6.7: Pressure coefficient distribution on the Gostelow blade surfaces for ni = 22, nk = 129.

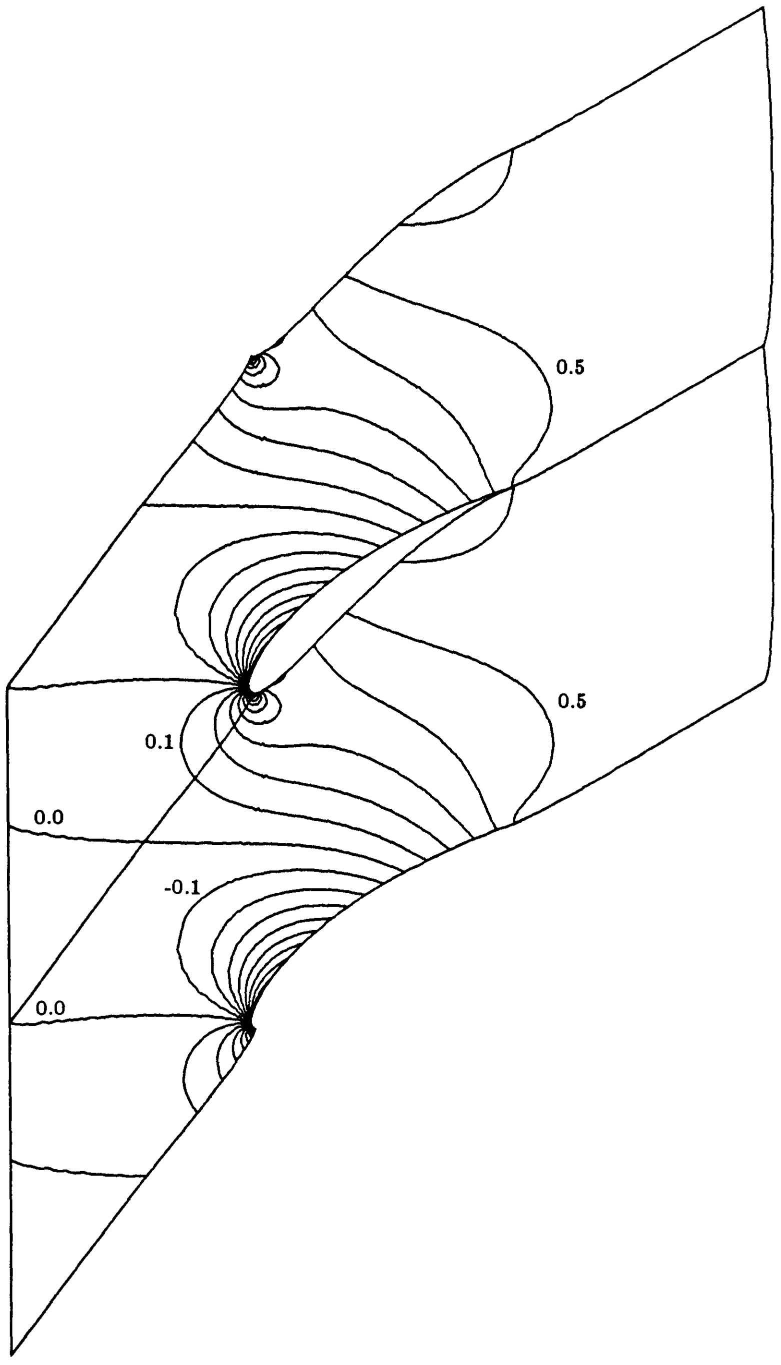


Figure 6.8: Contour plot of the pressure coefficient for the Gostelow test case. Contour interval = 0.1.

## 6.3 T7 Turbine Cascade

The T7 turbine nozzle is a high turning subsonic linear cascade designed by Rolls-Royce. Experimental surface pressure measurements have been made for this case. These data are compared with numerical procedures in references [27] and [59]. This case has an inlet angle of $52.8°$ and a pitch to axial chord ratio of 0.940094. The mass flow and total pressure are specified so the incoming Mach number is 0.28. The exit flow angle is approximately 70 degrees so this is a useful test case to determine the effect of grid shear on the algorithm.

Figure 6.9 shows the $21 \times 106$ grid used. The leading edge is shown in more detail in Fig. 6.10. The iteration history is tabulated in Table 6.12. For this case, the leading edge was not allowed to move until after the error norm was below a certain tolerance which happened after the first iteration. The solution converges until the line search algorithm causes heavy damping at iteration 6 and the program stops after the seventh iteration. The error norm is at $3.72 \times 10^{-8}$ which is machine accuracy on a 32 bit machine, but not on the CRAY. A coarse grid ($11 \times 52$) was also run and its solution history is in Table 6.13. It converges to 64 bit machine accuracy in 7 iterations. This shows that the problem is more than likely not a bug in the solver. This lack of convergence could be due to several things:

- The blade is defined as a cubic spline which may not be smooth enough locally to allow for realistic derivatives of the leading edge movement.
- The flow solution near the leading edge is highly non-linear. There may be local minima which prevent Newton's method from converging to the solution.

The solution is converged to better than engineering accuracy, so the results are meaningful. The $21 \times 106$ grid solution took 293 CRAY seconds and the coarse grid solution took 62 CRAY seconds.

Figure 6.11 shows the calculate Mach number distribution compared with the experimental results. The comparison is quite good. Fig. 6.12 shows the same plot but

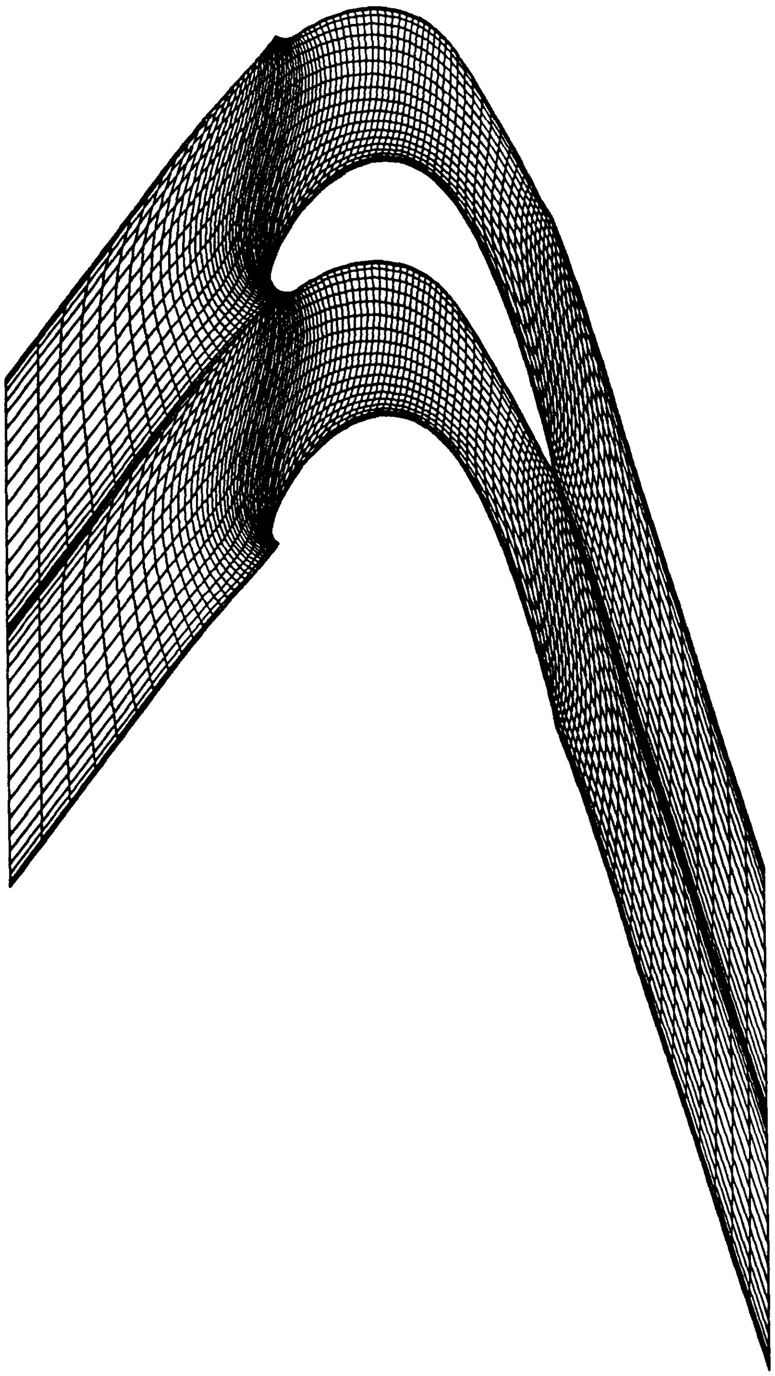


Figure 6.9: Grid for the T7 turbine cascade. ni = 21, nk = 106.

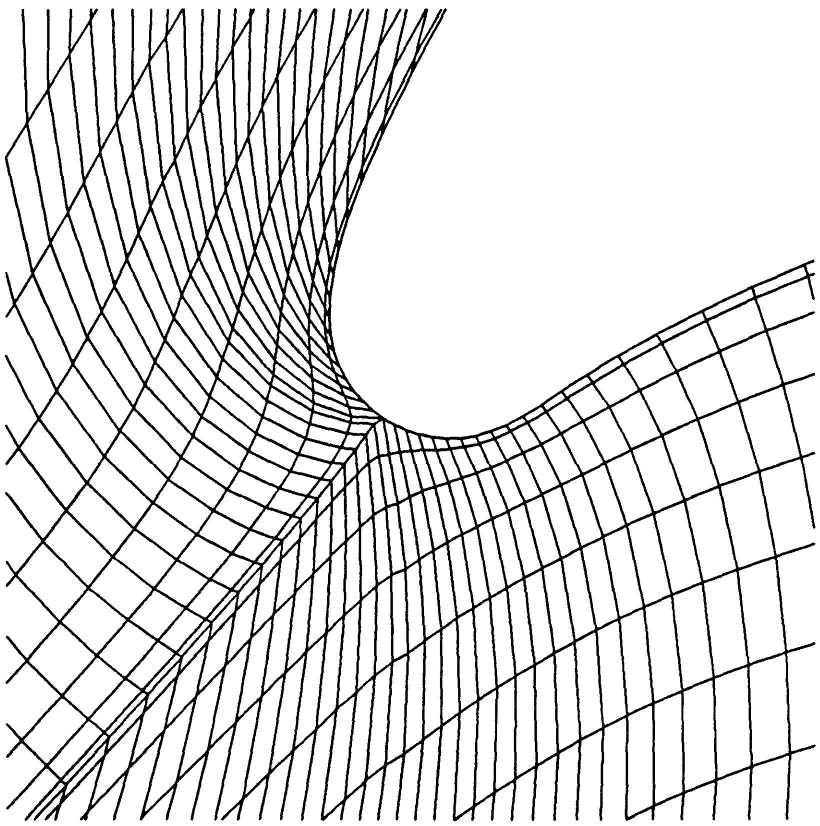


Figure 6.10: Blowup of grid about leading edge for the T7 turbine cascade. ni = 21, nk = 106.

Table 6.12: Convergence history for the T7 turbine nozzle. Compressible flow, $S$ momentum equation applied, ni=21, nk=106, GMRES max cycles = 15, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 9.94e-4 | - | |
| 1 | 7.47e-6 | 1.0 | matrix factored |
| 1 | 2.23e-5 | - | leading edge allowed to move |
| 2 | 1.97e-5 | 1.0 | matrix factored |
| 3 | 9.16e-6 | 1.0 | matrix factored |
| 4 | 7.50e-7 | 1.0 | matrix factored |
| 5 | 3.76e-8 | 1.0 | matrix factored |
| 6 | 3.72e-8 | 0.05 | matrix sol. in 13 GMRES cycles |
| 7 | 3.72e-8 | 0.0 | matrix sol. in 14 GMRES cycles |
| | | | line search could not decrease residual norm |

Table 6.13: Convergence history for the coarse grid T7 turbine nozzle. Compressible flow, $S$ momentum equation applied, ni=11, nk=52, GMRES max cycles = 15, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 1.57e-3 | - | |
| 1 | 2.83e-5 | 1.0 | matrix factored |
| 2 | 5.09e-6 | 1.0 | matrix factored |
| 2 | 1.69e-5 | - | leading edge allowed to move |
| 3 | 2.95e-7 | 1.0 | matrix factored |
| 4 | 3.66e-9 | 1.0 | matrix sol. in 9 GMRES cycles |
| 5 | 4.11e-11 | 1.0 | matrix sol. in 8 GMRES cycles |
| 6 | 2.92e-13 | 1.0 | matrix sol. in 6 GMRES cycles |
| 7 | 2.71e-15 | 1.0 | matrix sol. in 5 GMRES cycles |
| 8 | 1.66e-15 | 1.0 | matrix sol. in 2 GMRES cycles |

with the coarse grid results. The Mach number contours are shown in Fig. 6.13. The $S$ momentum equation was satisfied, so a measure of the accuracy of the calculation is to determine the total pressure errors. Contours of total pressure are shown in Fig. 6.14. Those contours less than the inlet value are plotted in the upper two passages and those contours greater than the inlet are plotted in the lower two passages. These errors are small, but do demonstrate the numerical dissipation for this algorithm. It also shows that as the grid becomes coarser due to the stretching, the errors get worse. In ISES, this error is minimized because there are no fluxes across the stream tubes. The discretization for an $A$ face is not second-order accurate for non-uniform grids. These total pressure errors show the effect this has on the solution. For 2D, the discretization could be made second-order accurate on any grid which might improve these errors. Again, these errors are small, and the overall results are good.

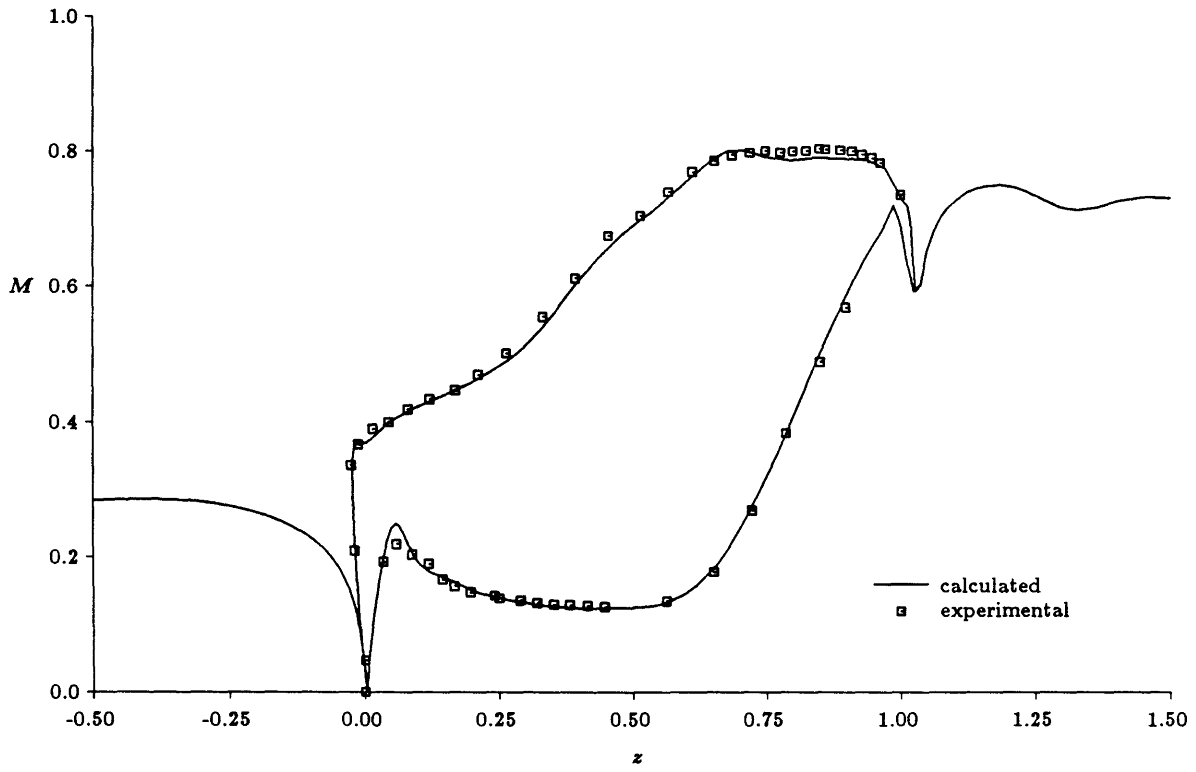


Figure 6.11: Comparison between the calculated and experimental Mach number distributions for the T7 turbine nozzle. The Mach number is calculated from the wall static pressures and the upstream total pressure. ni = 21, nk = 106.

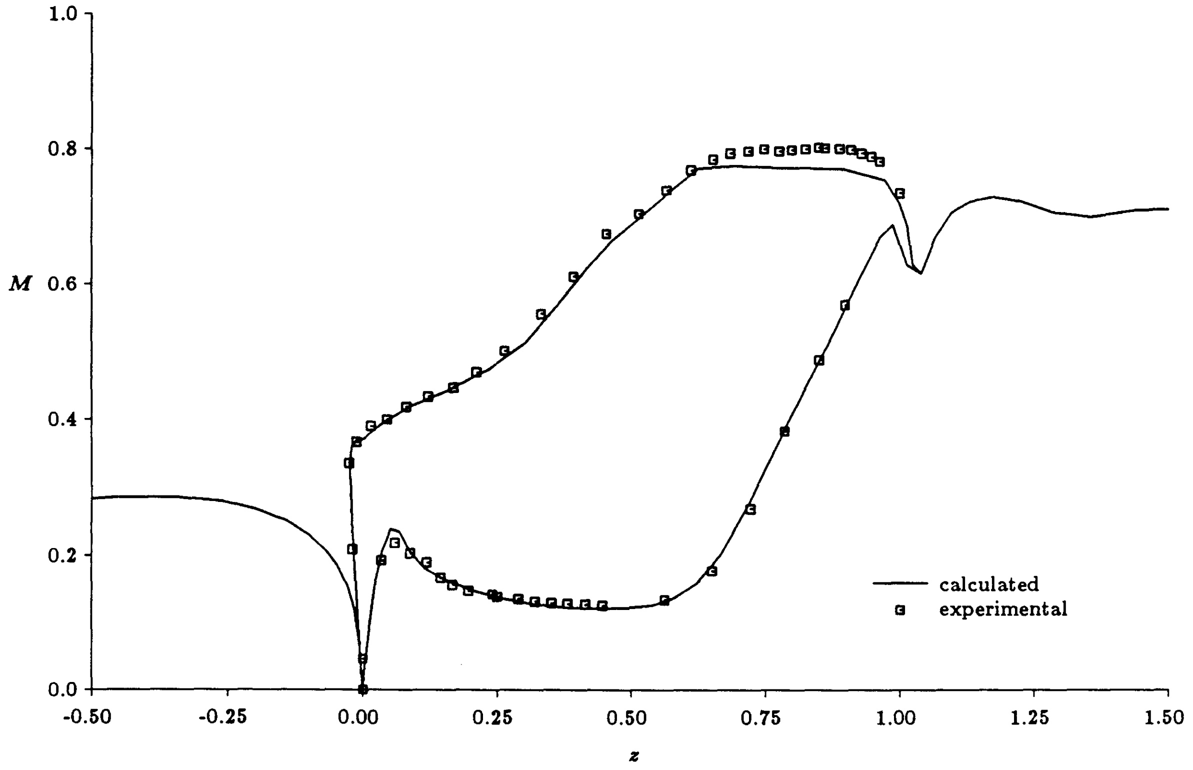


Figure 6.12: Comparison between the coarse grid calculated and experimental Mach number distributions for the T7 turbine nozzle. The Mach number is calculated from the wall static pressures and the upstream total pressure. ni = 11, nk = 52.

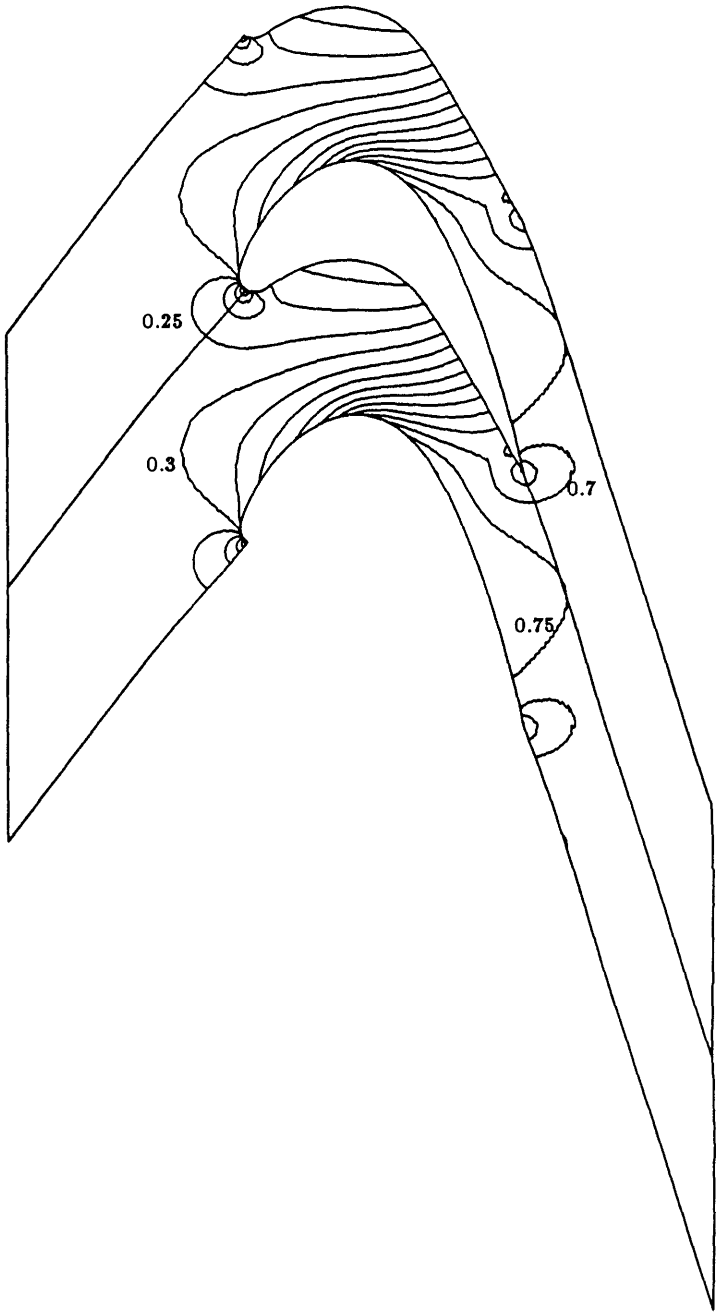


Figure 6.13: Contour plot of Mach number for the T7 turbine cascade. Contour interval = 0.05. The Mach number is calculated from the wall static pressures and the upstream total pressure. ni = 21, nk = 106.

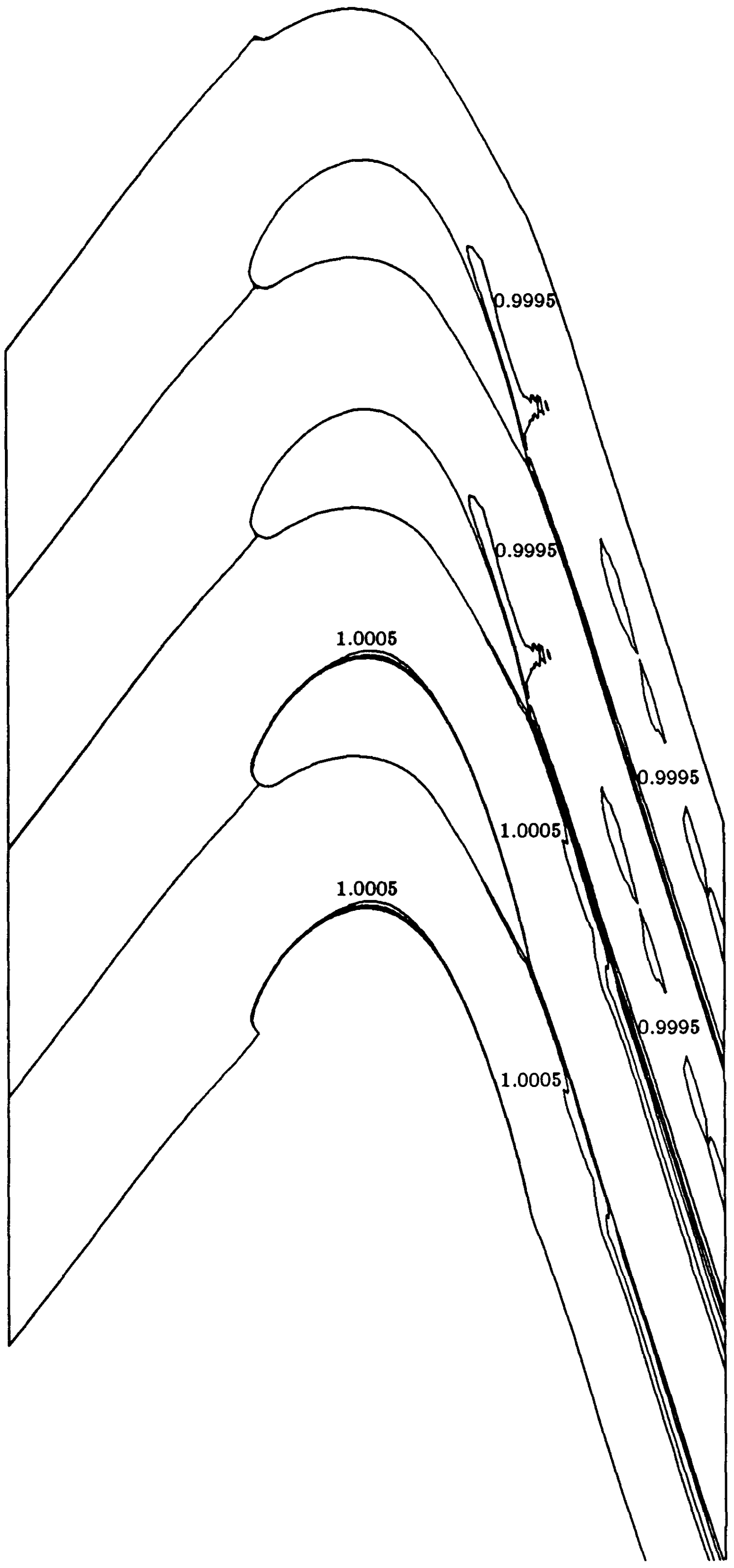


Figure 6.14: Contour plot of $P_T/P_T|_{ideal}$ for the T7 turbine nozzle. The upper two passages contain contours less than 1, and the lower two passages contain contours greater than 1. The contour interval = 0.0005.

Table 6.14: Convergence history for the Garabedian transonic compressor cascade. Compressible flow, $M_c^2 = 0.9$, $S$ momentum equation applied, ni=26, nk=89, GMRES max cycles = 15, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 1.35e-3 | - | |
| 1 | 3.59e-5 | 1.0 | matrix factored |
| 1 | 2.64e-4 | - | grid readjusted |
| 2 | 3.47e-5 | 1.0 | matrix factored |
| 3 | 9.78e-6 | 1.0 | matrix factored |
| 3 | 1.20e-4 | - | grid readjusted |
| 4 | 1.08e-5 | 1.0 | matrix factored |
| 5 | 4.45e-7 | 1.0 | matrix factored |
| 6 | 9.95e-9 | 1.0 | matrix sol. in 11 GMRES cycles |
| 7 | 6.33e-12 | 1.0 | matrix sol. in 12 GMRES cycles |
| 8 | 2.40e-15 | 1.0 | matrix sol. in 8 GMRES cycles |
| 9 | 2.40e-15 | 0.0 | matrix sol. in 4 GMRES cycles<br>line search could not decrease residual norm |

## 6.4 Transonic Garabedian Compressor Cascade

This test case is a supercritical compressor cascade designed by Garabedian using a numerical Hodograph method with a coupled boundary layer [5]. The inlet flow angle is 45.92° and the inlet Mach number is 0.72. The pitch to chord ratio is 1.201 and the pitch to axial chord ratio is 1.3868 with a 30 degree stagger angle. The blade plus displacement thickness is used as the blade shape. The trailing edge, therefore, is not closed and this gap continues in the wake region.

The $26 \times 89$ grid is shown in Fig. 6.15 with the leading edge detail shown in Fig. 6.16. The iteration history is in Table 6.14. Machine accuracy is achieved in 8 iterations. Only the first order pressure upwinding has been used with $M_c^2 = 0.9$. The total solution time is 361 CRAY seconds.

The Mach number contour plot is shown in Fig. 6.17 and the surface Mach number distribution is in Fig. 6.18 with a comparison to the Hodograph method. The agreement is good except in the supersonic region where a weak shock is predicted with a 1 % total pressure loss. This loss convects downstream as shown in Fig. 6.19. This is similar to the ISES prediction [27] using first order artificial compressibility.

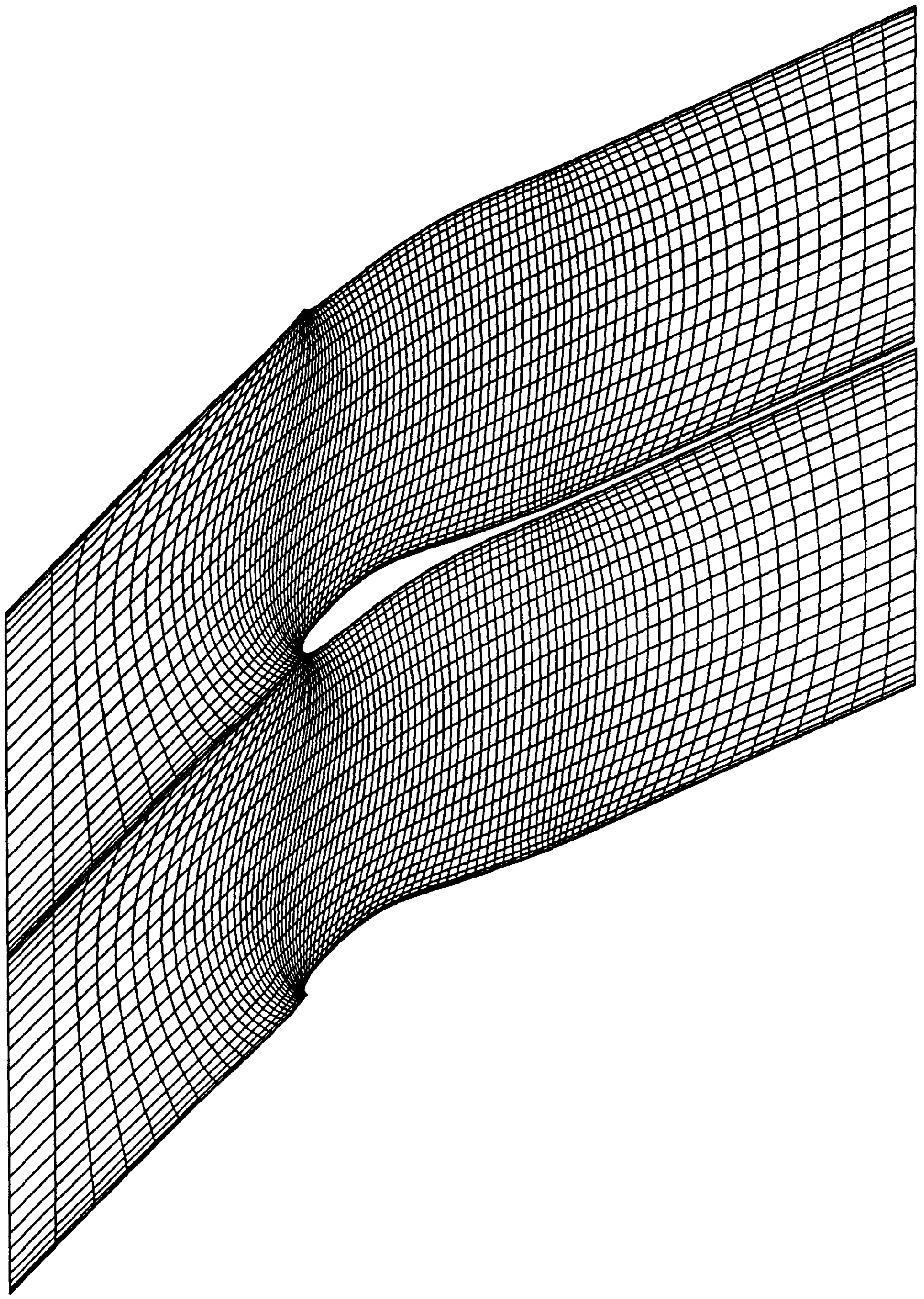


Figure 6.15: Grid for the Garabedian cascade. ni = 26, nk = 89.

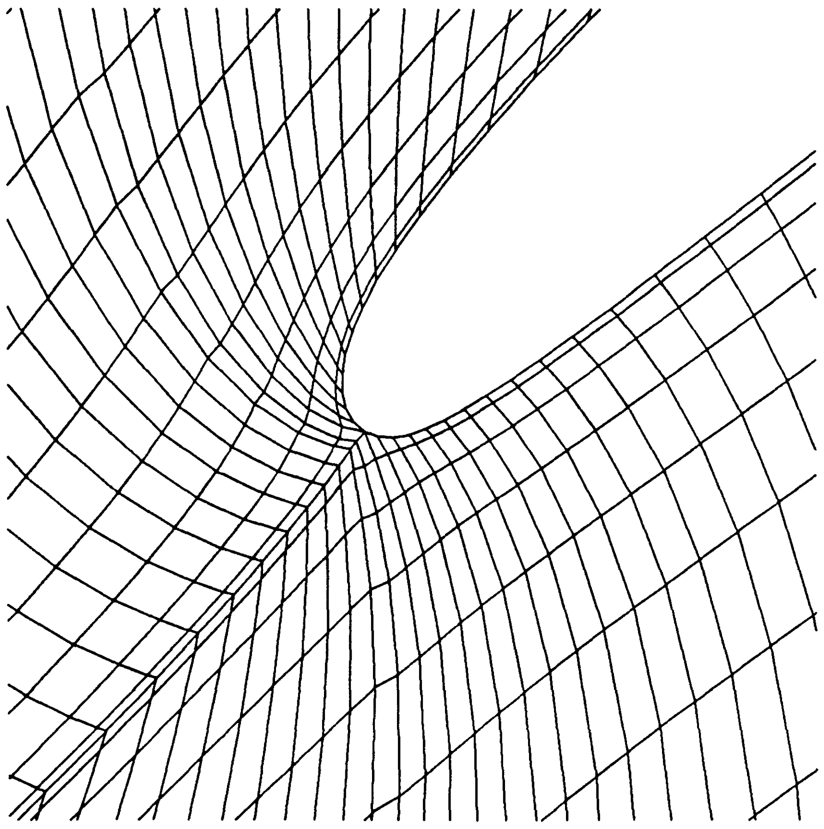


Figure 6.16: Blowup of grid about leading edge for the Garabedian cascade. ni = 26, nk = 89.

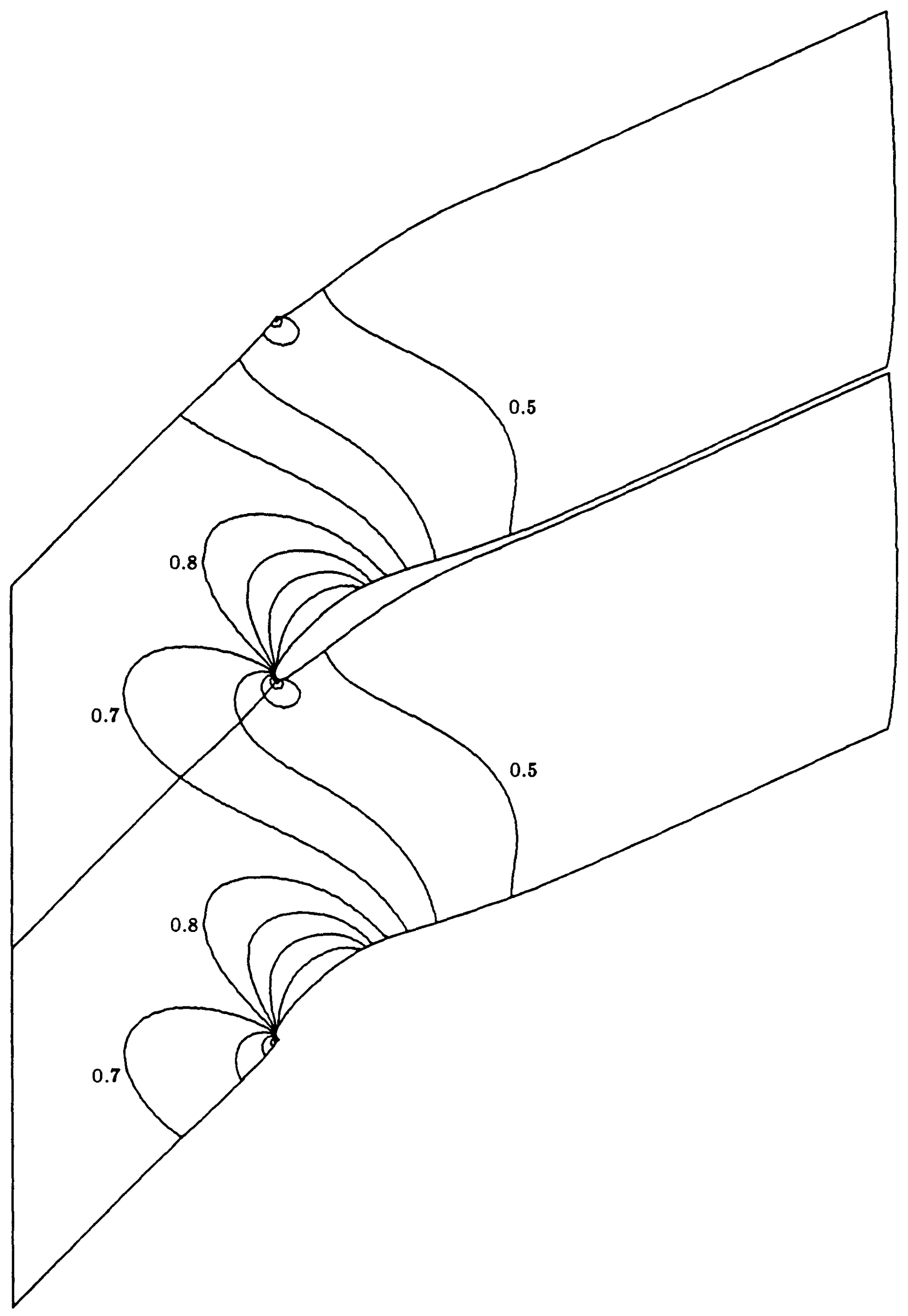


Figure 6.17: Contour plot of Mach number for the Garabedian cascade. Contour interval = 0.1. The Mach number is calculated from the wall static pressures and the upstream total pressure.

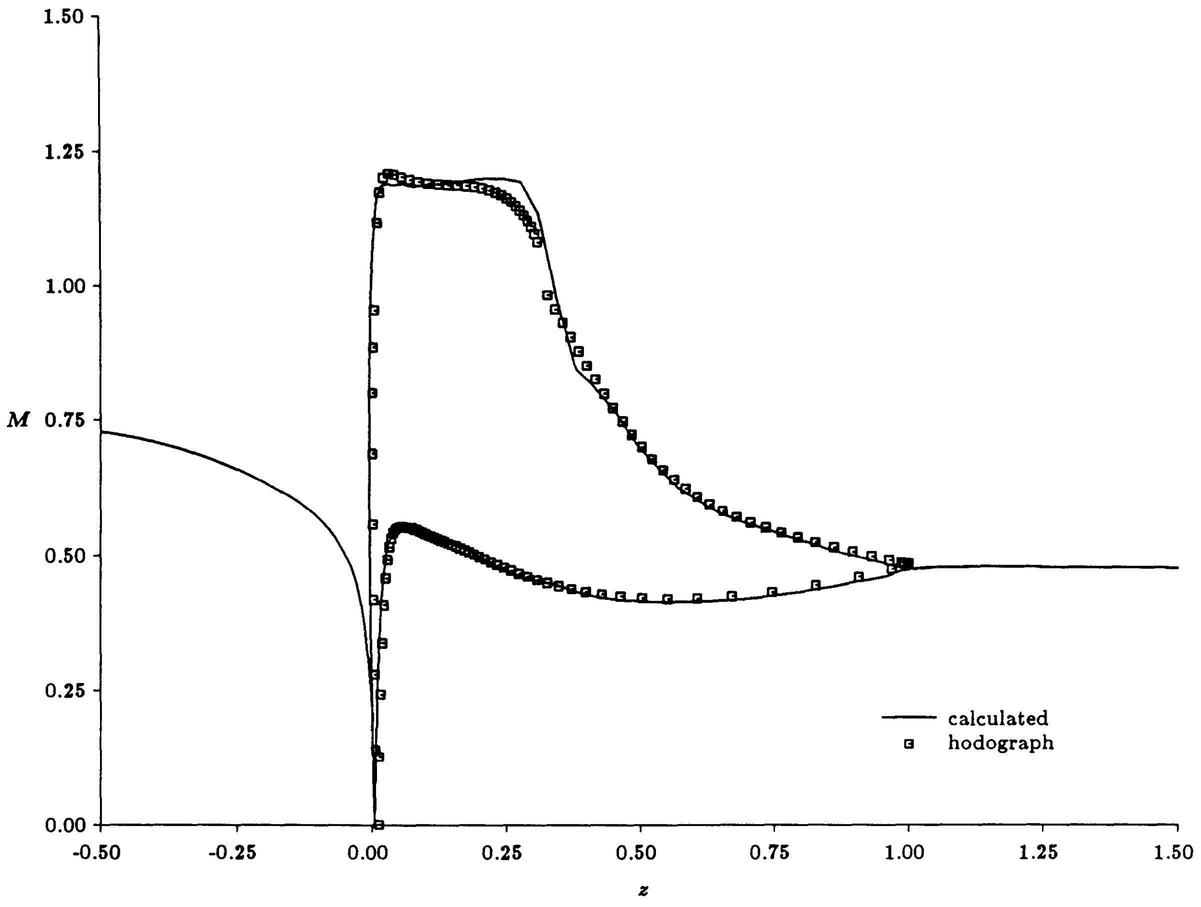


Figure 6.18: Comparison of the Mach number distribution calculated and the Hodograph solution for the Garabedian cascade. The Mach number is calculated from the wall static pressures and the upstream total pressure.

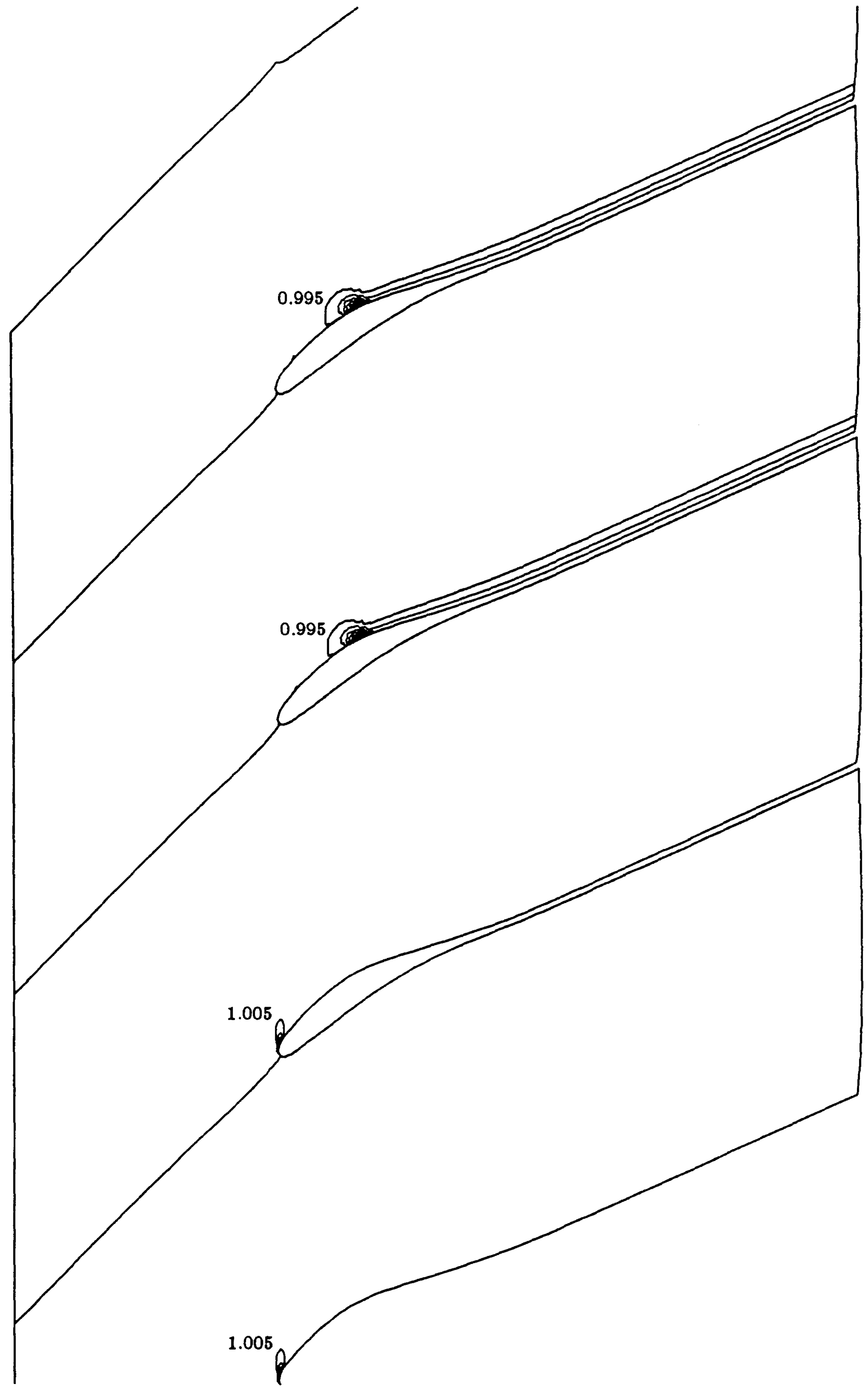


Figure 6.19: Contour plot of $P_T/P_T|_{upstream}$ for the Garabedian cascade. The upper two passages contain contours less than 1, and the lower two passages contain contours greater than 1. The contour interval = 0.005.

## 6.5 2D Inverse Design

This case demonstrates the design option capability in 2D. The blade shape is obtained from a thickness distribution and loading distribution ($\Delta P$) similar to that employed by Novak [46]. A NACA 0012 thickness distribution [1] is specified as well as a 30.790° inlet angle[1]. The mass flux and total pressure are the same as for the $\sin^2(\pi z)$ bump ($P_T = 0.69469$). The pressure loading is specified as:

$$\Delta P = 0.1 \sin(\pi(1-z)^2), \tag{6.6}$$

where $z = 0$ at the leading edge and $z = 1$ at the trailing edge. This loading distribution is plotted in Fig. 6.20.

The initial $19 \times 99$ grid showing the NACA 0012 thickness distribution is shown in Fig. 6.21. The solution history is in Table 6.15. Because the leading edge is not allowed to move for a design option case, it was felt that the more accurate entropy convection equation should be applied to prevent total pressure errors near the leading edge. Applying this equation from the very start caused the line search algorithm to damp the solution initially which was not really required. This did not occur with the use of the $S$ momentum equation, so this equation was used during the first two iterations. After that the entropy convection equation was used. The first three iterations were run on a VAX 8800 for 487 seconds and the other 6 iterations took 158 seconds to solve on the CRAY. Figure 6.22 shows the final grid. The elliptic grid generator was only run on the initial grid. Notice the waviness of the grid lines which are produced. In Fig. 6.23, three blade shapes and the stagnation streamlines are overlaid. These represent the converged solution and the solutions after the first and second iterations. Notice how rapidly the blade obtains its converged shape after starting from the grid in Fig. 6.21.

The surface Mach number distribution is in Fig. 6.24 and the Mach number contours are in Fig. 6.25.

To demonstrate the accuracy of the method with only one volume blade-to-blade

[1]This odd number is the result of an initial bad input which produced a reasonable blade shape. The value was then retained.

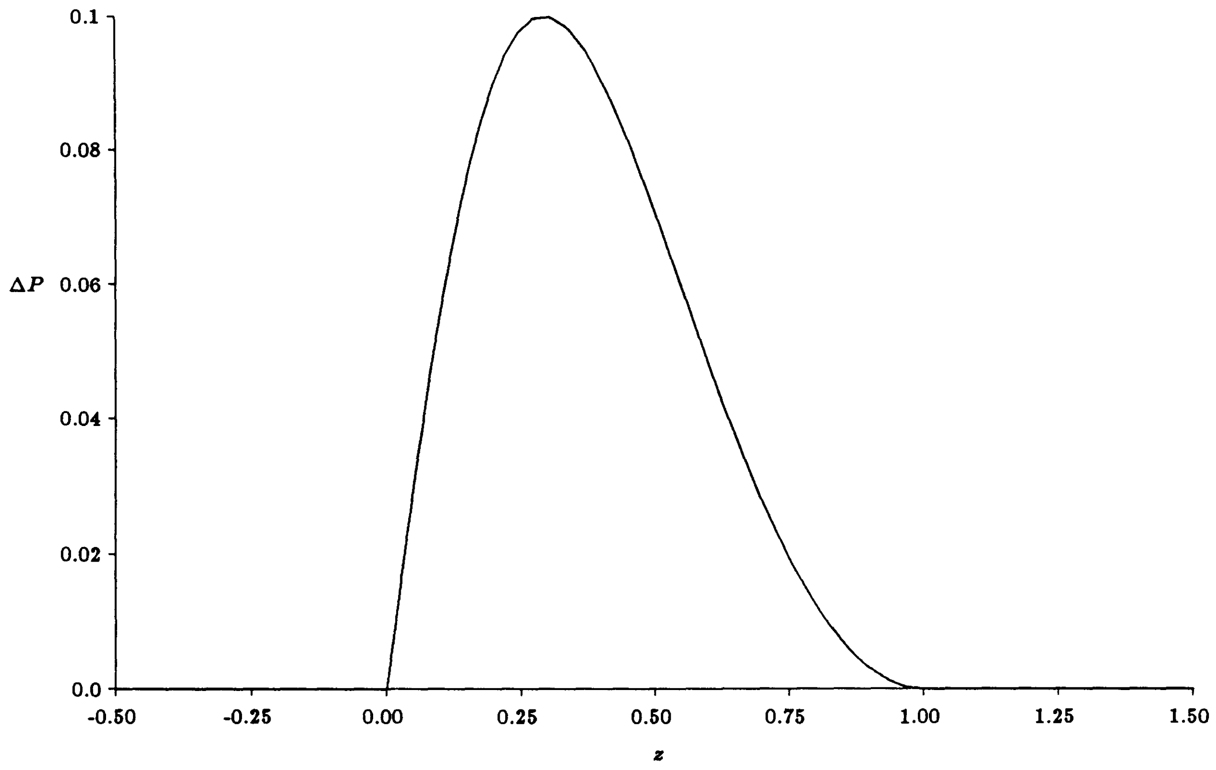


Figure 6.20: Loading distribution for the 2D design option case. ni = 19, nk = 99.

Table 6.15: Convergence history for the 2D design option case. Compressible flow, $S$ momentum equation applied during iteration 1 and 2, entropy convection equation applied after the second iteration, ni=19, nk=99, GMRES max cycles = 20, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 2.41e-3 | - | |
| 1 | 2.10e-3 | 1.0 | matrix factored |
| 2 | 7.81e-6 | 1.0 | matrix factored |
| 2 | 1.32e-2 | - | switch to the entropy convection equation |
| 3 | 4.44e-3 | 1.0 | matrix factored |
| 4 | 4.81e-4 | 1.0 | matrix factored |
| 5 | 8.01e-6 | 1.0 | matrix factored |
| 6 | 3.00e-12 | 1.0 | matrix sol. in 6 GMRES cycles |
| 7 | 1.89e-15 | 1.0 | matrix sol. in 3 GMRES cycles |
| 8 | 1.73e-15 | 1.0 | matrix sol. in 1 GMRES cycles |
| 9 | 1.73e-15 | 0.0 | matrix sol. in 1 GMRES cycles |
| | | | line search could not decrease residual norm |

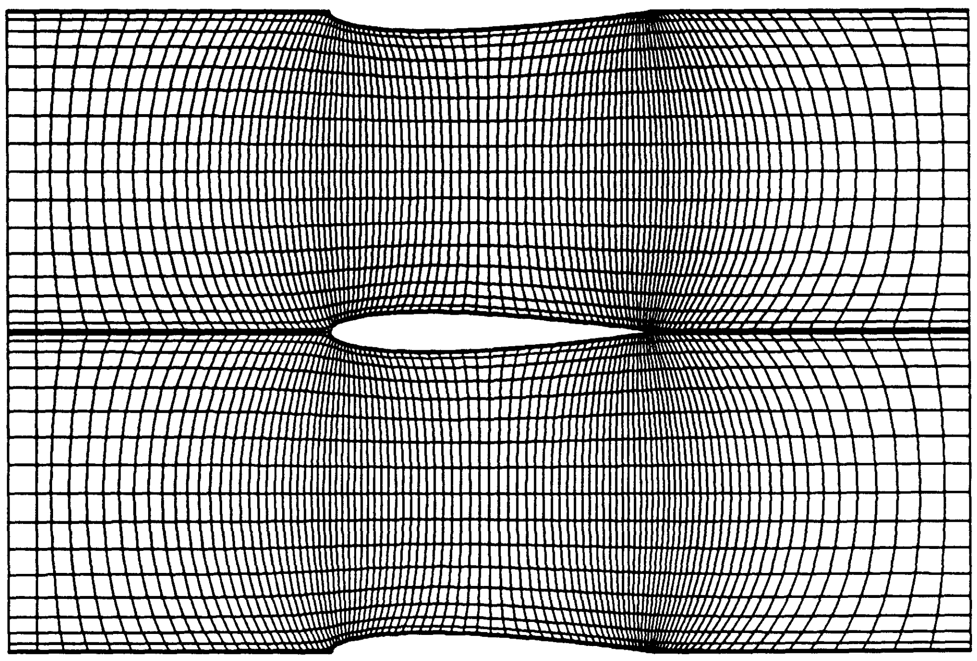


Figure 6.21: Initial grid for the 2D design option case. ni = 19, nk = 99.

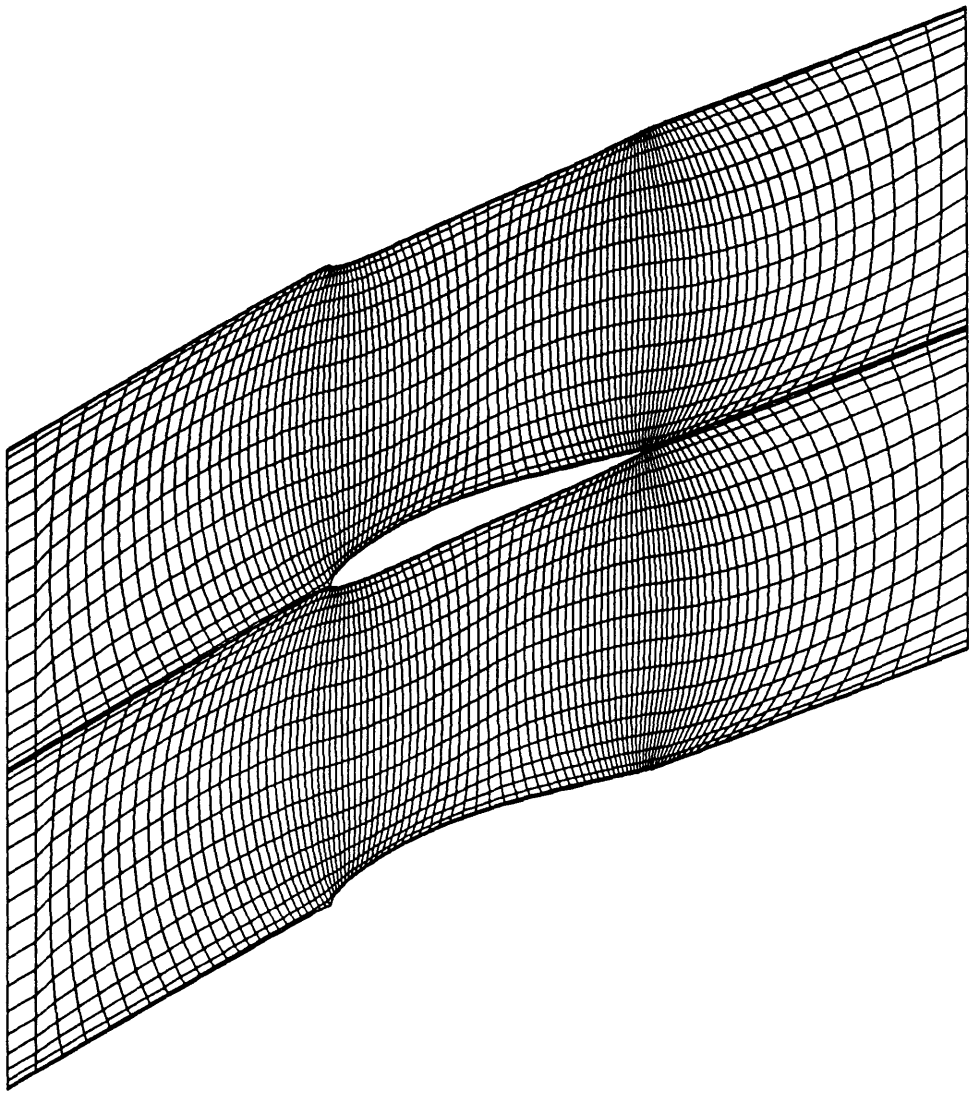


Figure 6.22: Final grid for the 2D design option case. ni = 19, nk = 99.

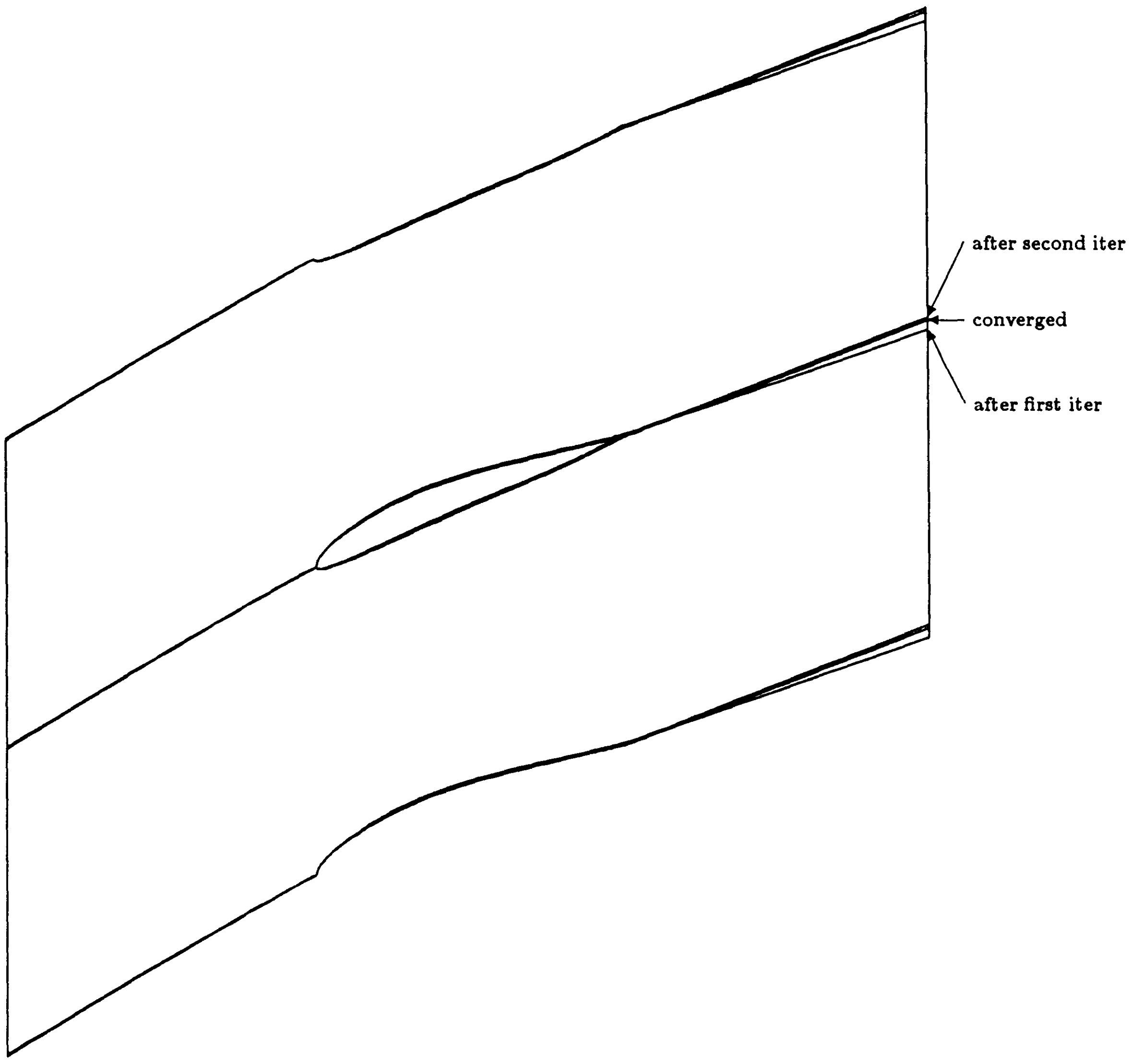


Figure 6.23: Comparison of the blade shape and stagnation streamlines for the converged solution and after the first and second iteration for the 2D design option case.

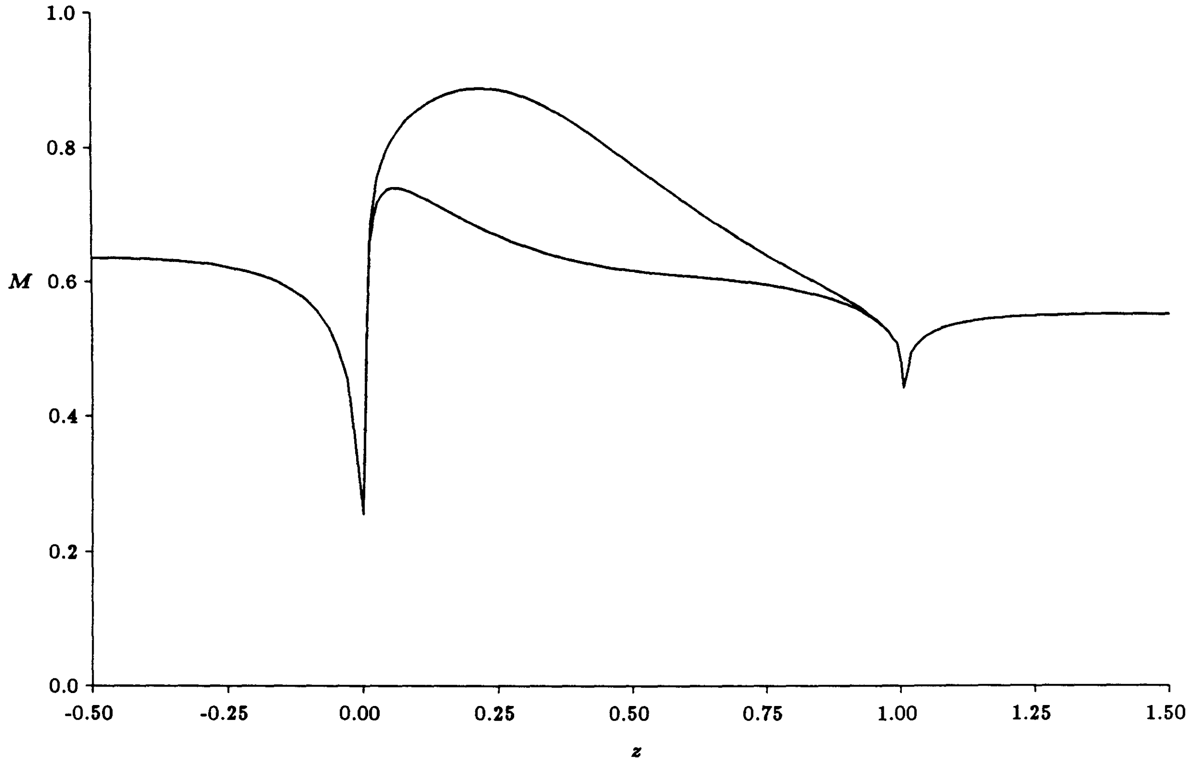


Figure 6.24: Surface Mach number distribution for the 2D design option case.

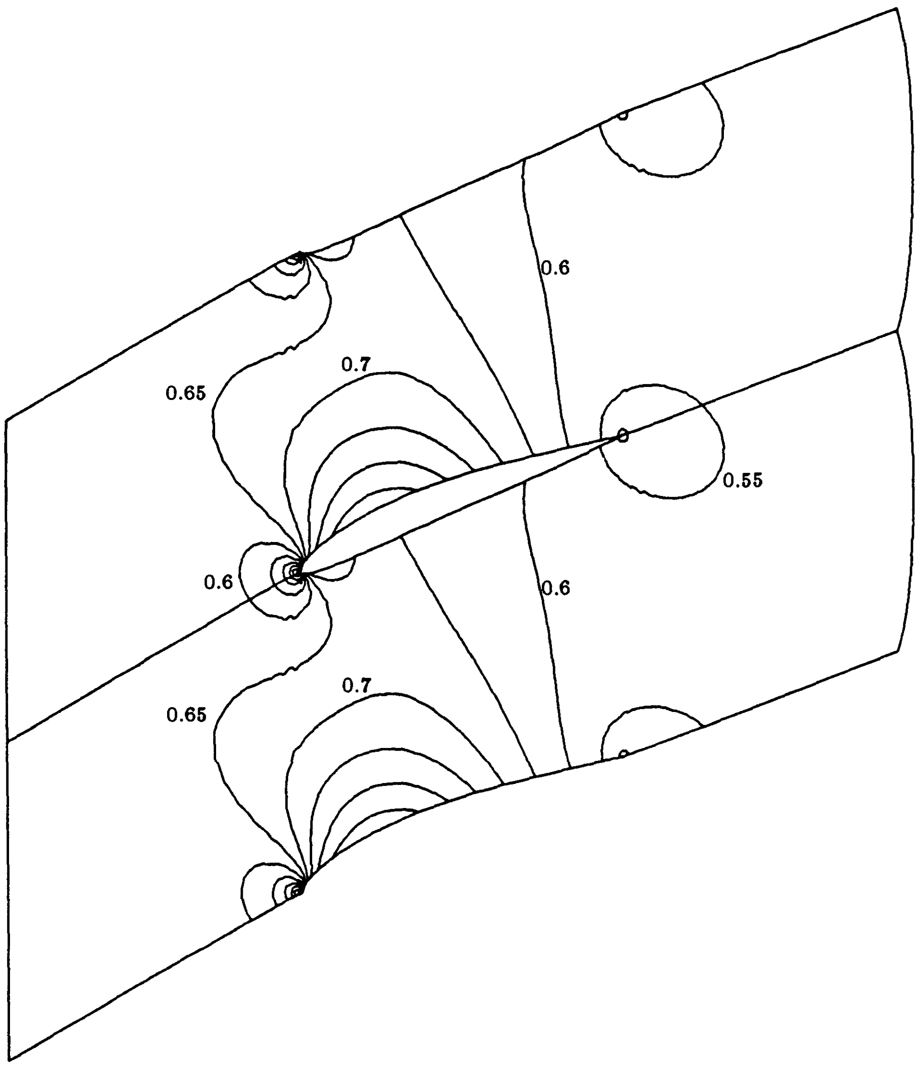


Figure 6.25: Contour plot of Mach number for the 2D design option case.

Table 6.16: Convergence history for the 2D design option case with one volume blade-to-blade. Compressible flow, entropy convection equation applied, ni=2, nk=99, GMRES max cycles = 20, no line search algorithm used, 7 iterations specified.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 8.51e-3 | - | |
| 1 | 2.55e-2 | 1.0 | matrix factored |
| 2 | 9.40e-5 | 1.0 | matrix factored |
| 3 | 2.16e-11 | 1.0 | matrix sol. in 5 GMRES cycles |
| 4 | 1.61e-14 | 1.0 | matrix sol. in 3 GMRES cycles |
| 5 | 8.71e-15 | 1.0 | matrix sol. in 1 GMRES cycles |
| 6 | 1.20e-14 | 1.0 | matrix sol. in 1 GMRES cycles |
| 7 | 8.79e-15 | 1.0 | matrix sol. in 1 GMRES cycles |

(these type of grids are used in many of the 3D calculations), this case was rerun with the same axial grid, and ni=2. The iteration history is in Table 6.16. No line search was performed because the initial residual norm was low due to the starting grid. This solution took 7.5 seconds to run on the CRAY.

The final grid is shown in Fig. 6.26. A comparison of the blade shapes produced using the ni=19 grid and the ni=2 grid is in Fig. 6.27. Notice how close the blade shapes are except near the leading edge where the implied pressure distribution with ni=2 is wrong. The Mach number distribution for this grid is in Fig. 6.28. It is not the same as Fig. 6.28, but the shape is very similar. All quantities are tangentially uniform upstream of the leading edge and downstream of the trailing edge. With the auxiliary pressure equation, the static pressure has a linear distribution within the blade passage. The exit angles match very well: $\alpha_{\text{exit}} = 20.987°$ for ni=19 and $\alpha_{\text{exit}} = 20.988°$ for ni=2.

Novak [46] uses the $rV_\theta$ derivative distribution to define the blade loading.They are related through the tangential momentum equation, and in 2D, this equation is:

$$-\Delta P = \frac{\dot{m}}{b}\frac{dV_x}{dz}. \tag{6.7}$$

where $b$ is a stream tube height in the y dimension. Integrating this equation from inlet to exit:

$$\int_{\text{inlet}}^{\text{exit}} -\Delta P\, dz = \frac{\dot{m}}{b} \int_{\text{inlet}}^{\text{exit}} dV_x. \tag{6.8}$$

where

$$\dot{m} = (\rho V_z)_{\text{inlet}} \Delta x\, b, \tag{6.9}$$

and $\Delta x$ is the pitch. The loading is only from $0 \leq z \leq 1$ , so Eq. (6.8) becomes

$$\int_0^1 \Delta P\, dz = (\rho V_z)_{\text{inlet}} \Delta x (V_x|_{\text{exit}} - V_x|_{\text{inlet}}). \tag{6.10}$$

This equation becomes a check on whether the global tangential momentum equation is satisfied. The given loading distribution in Eq. (6.6) has been integrated numerically using SMP:

$$-\int_0^1 0.1 \sin(\pi(1-z)^2)\, dz = -0.0504855. \tag{6.11}$$

From the flow solution:

$$(\rho V_z)_{\text{inlet}} = 0.4201;\ V_x|_{\text{inlet}} = 0.31223;\ V_x|_{\text{exit}} = 0.19215;\ \Delta x = 1. \tag{6.12}$$

The right side of Eq. (6.10) becomes

$$(0.42001)(1)(0.19215 - 0.31223) = -0.050434 \tag{6.13}$$

which is very close to the loading integral and demonstrates that the global tangential momentum equation is satisfied.

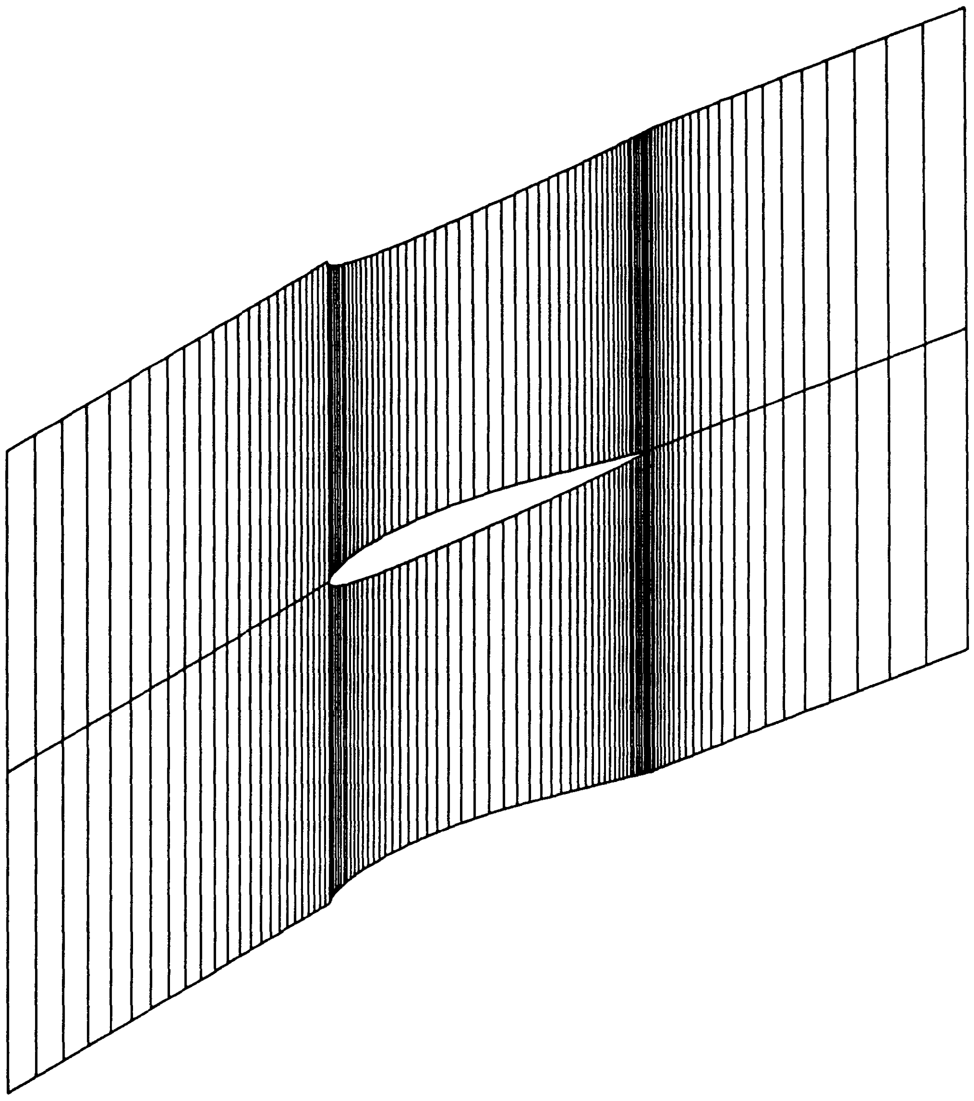


Figure 6.26: Final grid for the 2D design option case with one volume blade-to-blade. ni = 2, nk = 99.

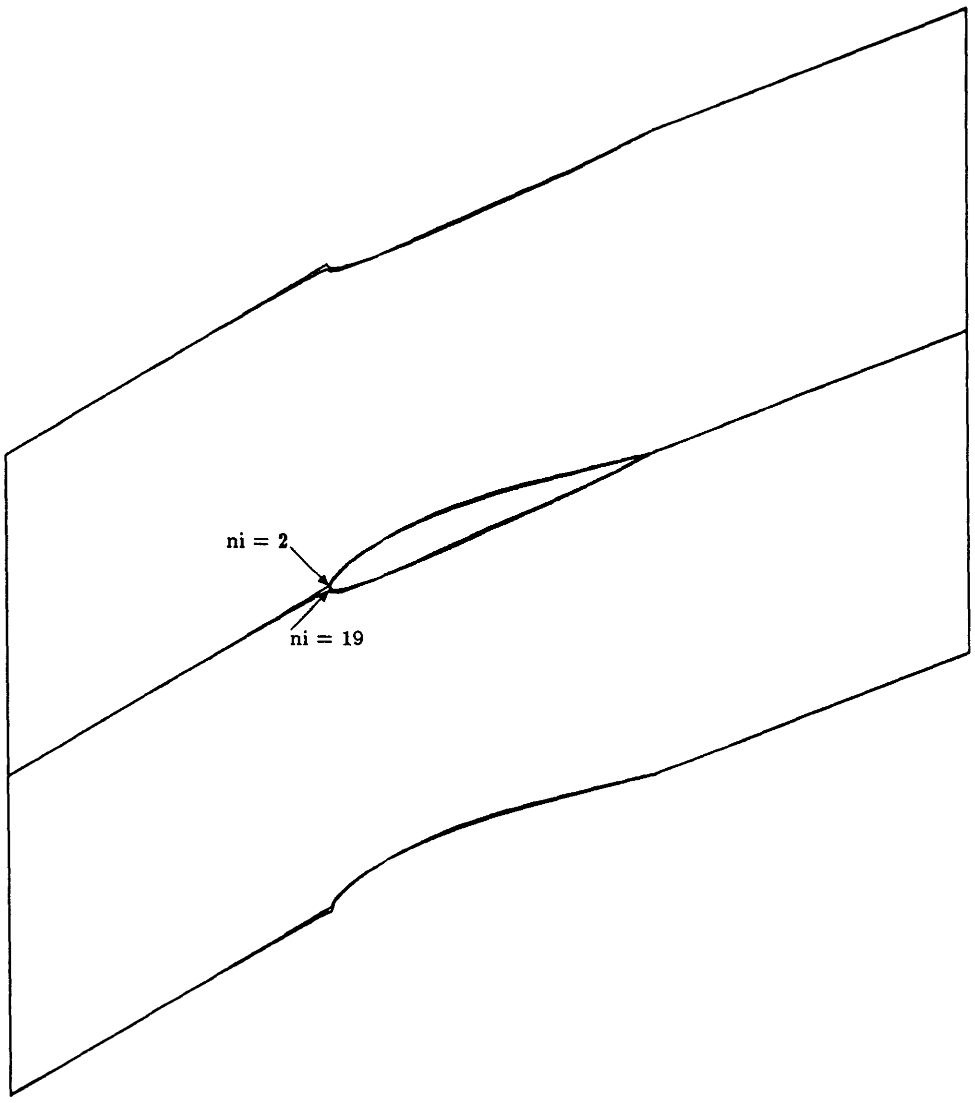


Figure 6.27: Comparison of the converged blade shape and stagnation streamlines for the 2D design option case with ni=2 and ni=19.

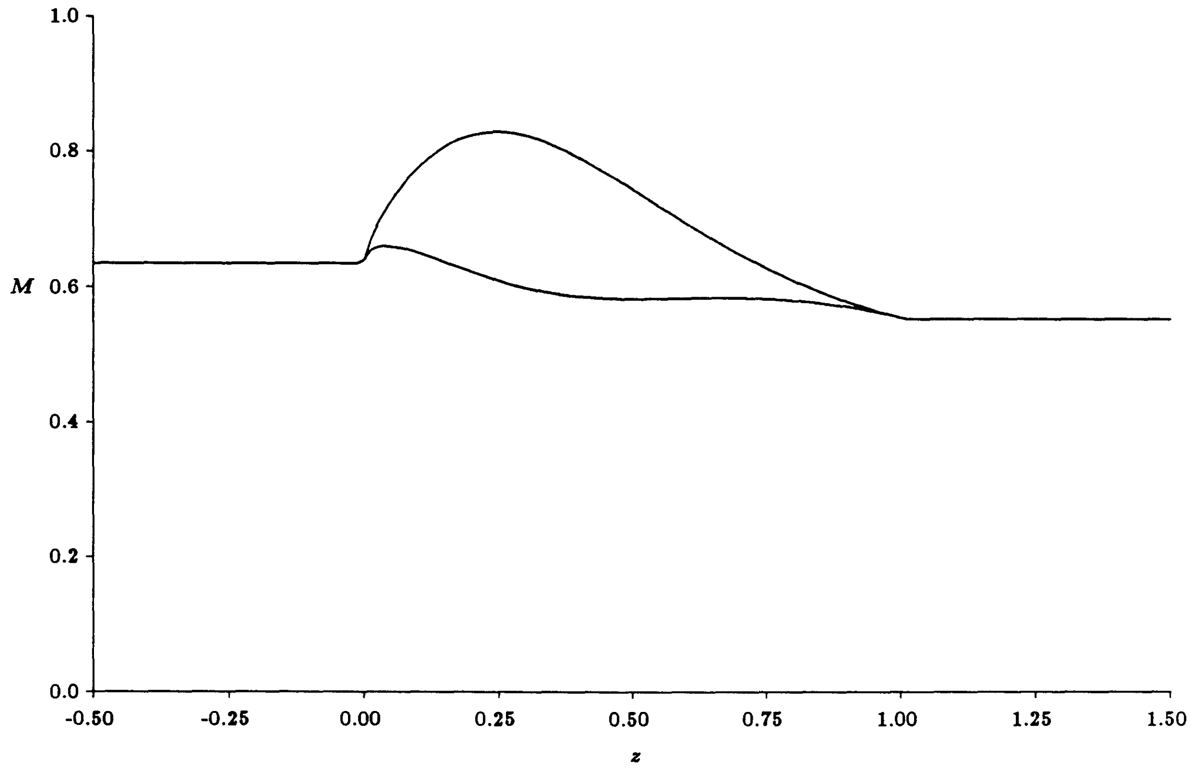


Figure 6.28: Surface Mach number distribution for the 2D design option case with one volume blade-to-blade.

# Chapter 7
# 3D Results

## 7.1 Stream Surface Warping in a Uniform Vorticity Duct

In order to demonstrate the use and limitations of the two stream function approach, the stream functions will be calculated for an infinite square duct with uniform streamwise vorticity. A special-purpose code has been used to make this calculation, but many of the subroutines of the flow solver have been utilized. This has allowed several issues about the two stream function formulation to be looked into.

The vorticity transport equation for incompressible flow is given in [4] as

$$\frac{D\overline{\omega}}{Dt} = \overline{\omega} \cdot \nabla \overline{V} + \nu \nabla^2 \overline{\omega}. \tag{7.1}$$

For steady, inviscid flow in a square duct which is infinite in the $z$ direction, one particular solution corresponds to

$$V_x = V_x(x, y), \tag{7.2a}$$

$$V_y = V_y(x, y), \tag{7.2b}$$

$$V_z = \text{constant}, \tag{7.2c}$$

$$\omega_x = \omega_y = 0, \tag{7.2d}$$

$$\omega_z = \text{constant}. \tag{7.2e}$$

This is only a 2D problem in some respects, however the stream surfaces will warp due to the secondary velocity field. Assuming a uniform distribution of the two stream functions at a given $z$ location (this is valid because the mass flow is uniform), the two stream functions can be determined at a $z$ station downstream. This is done by solving the following equations for $\psi_1$ and $\psi_2$ knowing $V_x$ and $V_y$. Solve

$$\rho V_x = \frac{\partial \psi_1}{\partial y}\frac{\partial \psi_2}{\partial z} - \frac{\partial \psi_1}{\partial z}\frac{\partial \psi_2}{\partial y} \tag{7.3}$$

at a $\psi_1$ location and solve

$$\rho V_y = \frac{\partial \psi_1}{\partial z}\frac{\partial \psi_2}{\partial x} - \frac{\partial \psi_1}{\partial x}\frac{\partial \psi_2}{\partial z} \tag{7.4}$$

at a $\psi_2$ location. At the boundaries, $\psi_1$ and $\psi_2$ are specified. Essentially Eqs. (7.3) and (7.4) replace the $A$ and $B$ momentum equations and many parts of the solution algorithm are used to solve this system. The discretizations are also consistent with the solver.

To obtain the cross stream velocity field, the streamwise vorticity definition is used:

$$\frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y} = \omega_z. \tag{7.5}$$

Introducing the 2D stream function $\psi$,

$$V_y = \frac{\partial \psi}{\partial x},\ V_x = -\frac{\partial \psi}{\partial y}, \tag{7.6}$$

Eq. (7.5) reduces to Poisson's equation

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \omega_z. \tag{7.7}$$

This is solved using a finite difference approach on a grid twice as fine as $\psi_1$ and $\psi_2$ are solved on.

For a 5 unit square duct with $\psi_1|\text{max} = 5$, $\psi_2|\text{max} = 5$, $\omega_z = 1$, $\Delta z = 1/4.9$ and ni=nj=10, the results are in Fig. 7.1 and Fig. 7.2.

Figure 7.1 shows the secondary velocity field for the given vorticity field, the uniform $\psi_1$, $\psi_2$ distribution specified at $z = 0$, and the $\psi_1$, $\psi_2$ distribution at the first station downstream at $z = 1/4.9$. For this uniform mass flux case, notice how the initially square stream tubes deform, but that the area remains the same.

Figure 7.2 shows the $\psi_1$ distribution ($\psi_2$ is the same distribution rotated 90°) as the stream surfaces progress downstream. The algorithm diverges after the station in

(d) $z = 15/4.9$. For finer grids, the algorithm diverges even sooner. The corner is a streamline so one family of surfaces cannot go past the corner. Theoretically, the stream surfaces would eventually wind up into spiral-like layers which are infinitesimally close to the wall, and then would be distributed about the passage. This problem which has no gradients of vorticity or axial velocity can produce very strong gradients of the stream functions. Also the equations become more nonlinear as the stream surfaces warp.

If an inlet total enthalpy profile, $H$, exists at $z = 0$ and corresponds to the $\psi_1$ distribution, then assuming inviscid flow, the enthalpy distribution would be the same as the $\psi_1$ distribution. This implies that the gradients of $H$ would be very large near the corners just as the gradients of $\psi_1$ are. An Euler solver would have trouble resolving these gradients also, and would most likely smear them out which would not represent an inviscid solution.

This test case is not exactly like the solver because in this example the velocities are used directly to determine the stream function which is marched downstream. In the solver, the discrete conservation equations are used to determine the stream function distribution and cannot be marched because the equations are elliptic. This square duct can be used to understand what happens to the matrix in the solver as the stream surfaces warp.

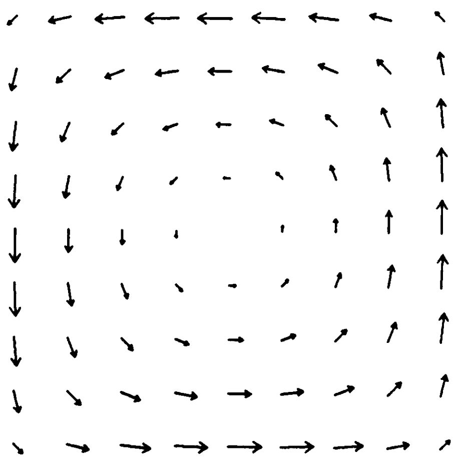
Consider the $A$ momentum equation which is the $x$ momentum equation for this regular grid. Essentially, the pressure force on a control volume in the $x$ direction is balanced by the net $x$ momentum leaving the control volume. In differential form, the $x$ momentum equation is:

$$\rho V_x \frac{\partial V_x}{\partial x} + \rho V_y \frac{\partial V_x}{\partial y} + \rho V_z \frac{\partial V_x}{\partial z} = -\frac{\partial P}{\partial x}. \tag{7.8}$$

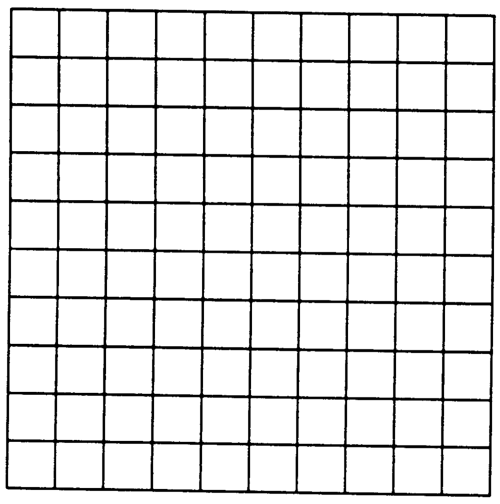
The third term in this equation can be examined by expanding the derivative of $V_x$ with respect to $z$ using Eq. (7.3):

$$\rho \frac{\partial V_x}{\partial z} = \frac{\partial \psi_1}{\partial y}\frac{\partial^2 \psi_2}{\partial z^2} + \frac{\partial^2 \psi_1}{\partial y \partial z}\frac{\partial \psi_2}{\partial z} - \frac{\partial \psi_1}{\partial z}\frac{\partial^2 \psi_2}{\partial y \partial z} - \frac{\partial^2 \psi_1}{\partial z^2}\frac{\partial \psi_2}{\partial y}, \tag{7.9}$$

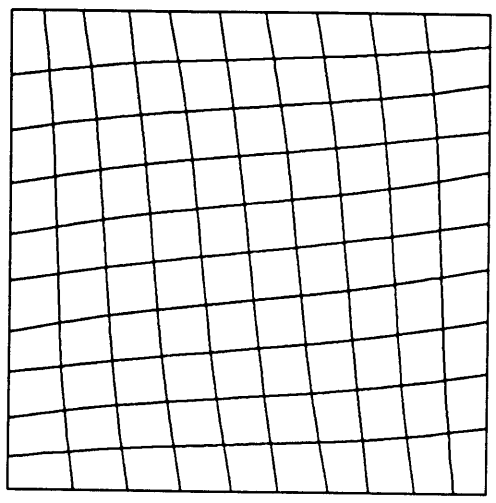
where density is constant. A discrete form of this expression is obtained when the



a) Velocity vectors.

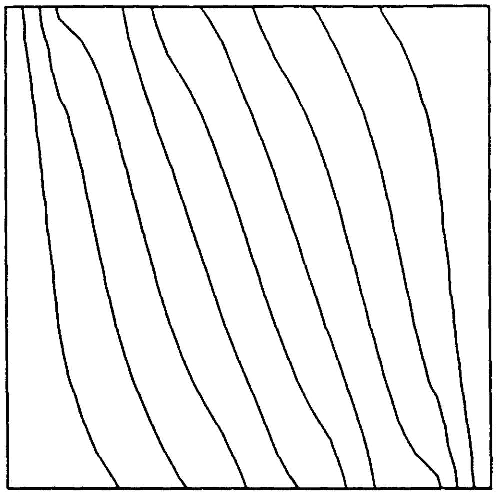


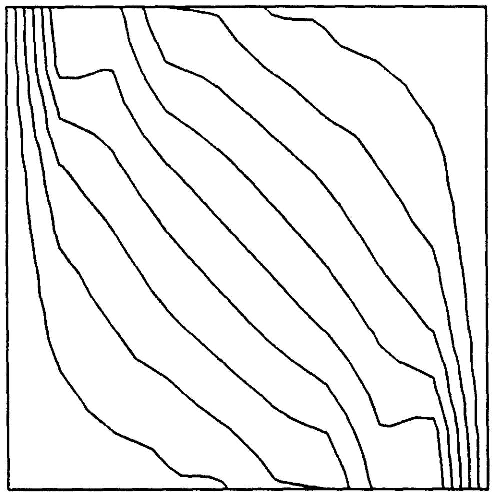
b) $\psi_1$, $\psi_2$ distribution at $z = 0$.

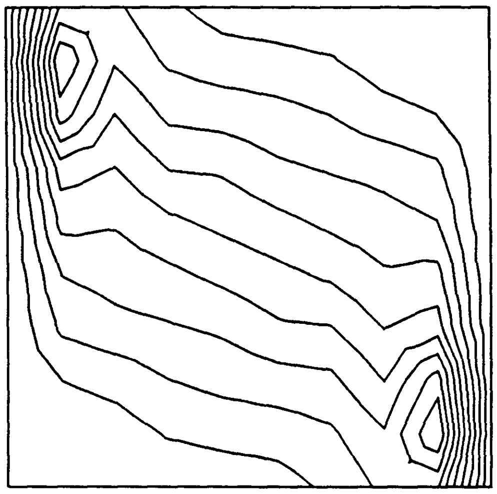


c) $\psi_1$, $\psi_2$ distribution at $z = 1/4.9$.

Figure 7.1: Secondary velocity field and initial and first station $\psi_1$, $\psi_2$ distributions for the uniform vorticity duct.


a) $z = 3/4.9$.


b) $z = 7/4.9$.


c) $z = 11/4.9$.

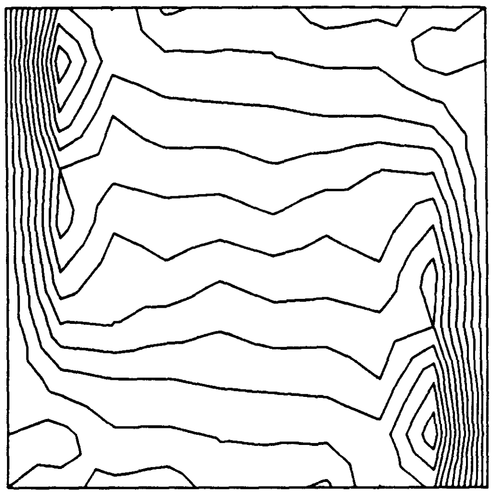

d) $z = 15/4.9$.

Figure 7.2: The $\psi_1$ distributions at four axial stations for the uniform vorticity duct.

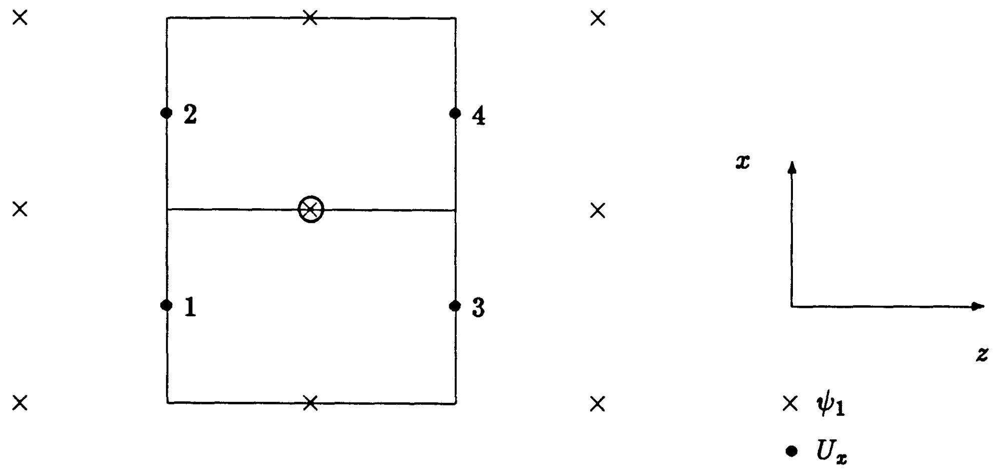


Figure 7.3: Projection of the reduced $A$ momentum control volumes in the $x-z$ plane showing the $\psi_1$ stencil.

momentum flux through the $S$ faces of a control volume are subtracted. When the equations are combined as discussed in Section 5.2.2, a stencil centered about a $\psi_1$ value is obtained. This is shown in Fig. 7.3 which is a projection of the two control volumes shown in Fig. 5.1 in the $(\xi^1,\xi^3)$ or $(x,z)$ plane. The $\psi_1$ stencil is also shown. Basically, for a uniform mass flux, the term corresponding to Eq. (7.9) is

$$\frac{\rho}{2\Delta z}\left[(V_x|_4+V_x|_3)-(V_x|_2+V_x|_1)\right]. \tag{7.10}$$

For a uniform grid, the coefficients associated with the $\frac{\partial^2\psi_1}{\partial z^2}$ term are shown in Fig. 7.4. This term has $\frac{\partial\psi_2}{\partial y}$ multiplied by it. As the stream surfaces rotate 90 degrees, this term goes to zero, and the diagonal of this equation in the matrix becomes small. As $\frac{\partial\psi_2}{\partial y}$ gets small, $\frac{\partial\psi_1}{\partial y}$ gets larger. However, the term $\frac{\partial^2\psi_2}{\partial z^2}$ which $\frac{\partial\psi_1}{\partial y}$ is multiplied by does not produce one large coefficient because of the way the equations are added. The central $\psi_1$ value is also on the diagonal of the matrix. As the stream surface rotates, the matrix becomes less well conditioned due to the assumed direction of the gradients.

Three problems have been addressed with this test case when the stream surfaces warp significantly. These are:

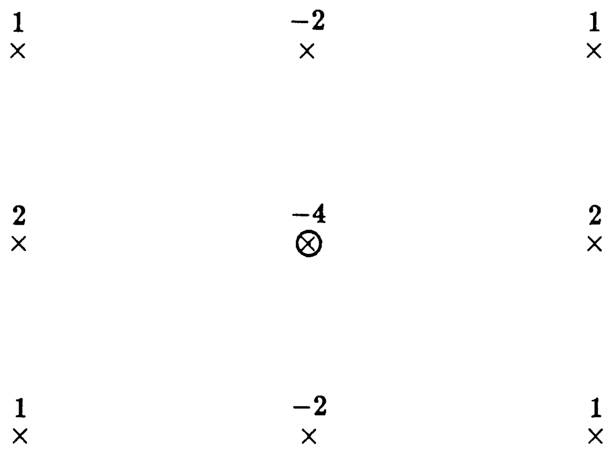


Figure 7.4: $\psi_1$ stencil for the $\frac{\partial^2 \psi_1}{\partial x^2}$ term for a uniform grid. This term gets multiplied by $\frac{\partial \psi_2}{\partial y}$.

1. The gradients of one stream function gets large as the gradient of the other stream function gets small. This requires a fine grid to resolve.
2. The equations become more nonlinear.
3. The discrete equations assume the gradients are predominantly in one direction, and if this is violated, the matrix becomes ill-conditioned.

The first two problems are associated with the two stream function approach in general. The third problem is strictly due to the present discretization approach. It was formulated to have many of the properties of ISES, and this assumption was necessary to do that. Another discretization approach, such as a finite element or spectral element method which could produce a positive definite matrix may not have this problem. However, these methods have other problems in dealing with nonlinearities, matrix size, or handling transonic flows.

## 7.2 GE $E^3$ Low Pressure Turbine Nozzle

NASA sponsored the Energy Efficient Engine ($E^3$) program between the late 70's and early 80's. General Electric had one contract to design, build, and test one of the demonstrator engines. This included the design and testing of the low pressure turbine which is described in reference [7]. The stage 1 nozzle was tested under two configurations. The Block II vane configuration with 72 blades has been used as a benchmark test case for several 3D analysis codes developed at GE. These include a time marching cell-centered Euler solver written by Holmes [34], and a pressure correction Navier-Stokes solver written by Hah and Leylek [31]. It is this configuration which has been analyzed with the current method and the solution is compared to the test and other analysis results.

Figure 7.5 shows the flow path of the nozzle as tested along with some boundary layer rake data obtained just upstream of the nozzle leading edge. This incoming hub and tip boundary layer and its development in the nozzle significantly affects the flowfield in the nozzle. An incoming shear layer is therefore prescribed at the upstream boundary of the flowfield for this inviscid analysis. It is the same one that was imposed for the analysis using the cell-centered Euler solver, and is shown in Fig. 7.6. To account for the boundary layer growth within the blade, the imposed shear layer is more pronounced than that measured at the leading edge. The measured total pressure profile at station 50 (see Fig. 7.5) just downstream of the trailing edge is in Fig. 7.7. A vortex core separates at the tip to produce this total pressure profile. The current inviscid code cannot predict this, but should be able to determine the overall global features of the flow.

Because of this shear layer, and the limitations of the two stream function method discussed in the previous section, the analysis was performed with one volume blade-to-blade as was shown in 2D for the design option case in the previous chapter. This implies that the stream function is axisymmetric, and the static pressure is linear across the blade passage and axisymmetric upstream of the leading edge and downstream of the trailing edge. The grid used in the $r$-$z$ plane is shown in Fig. 7.8 (ni=2, nj=33,

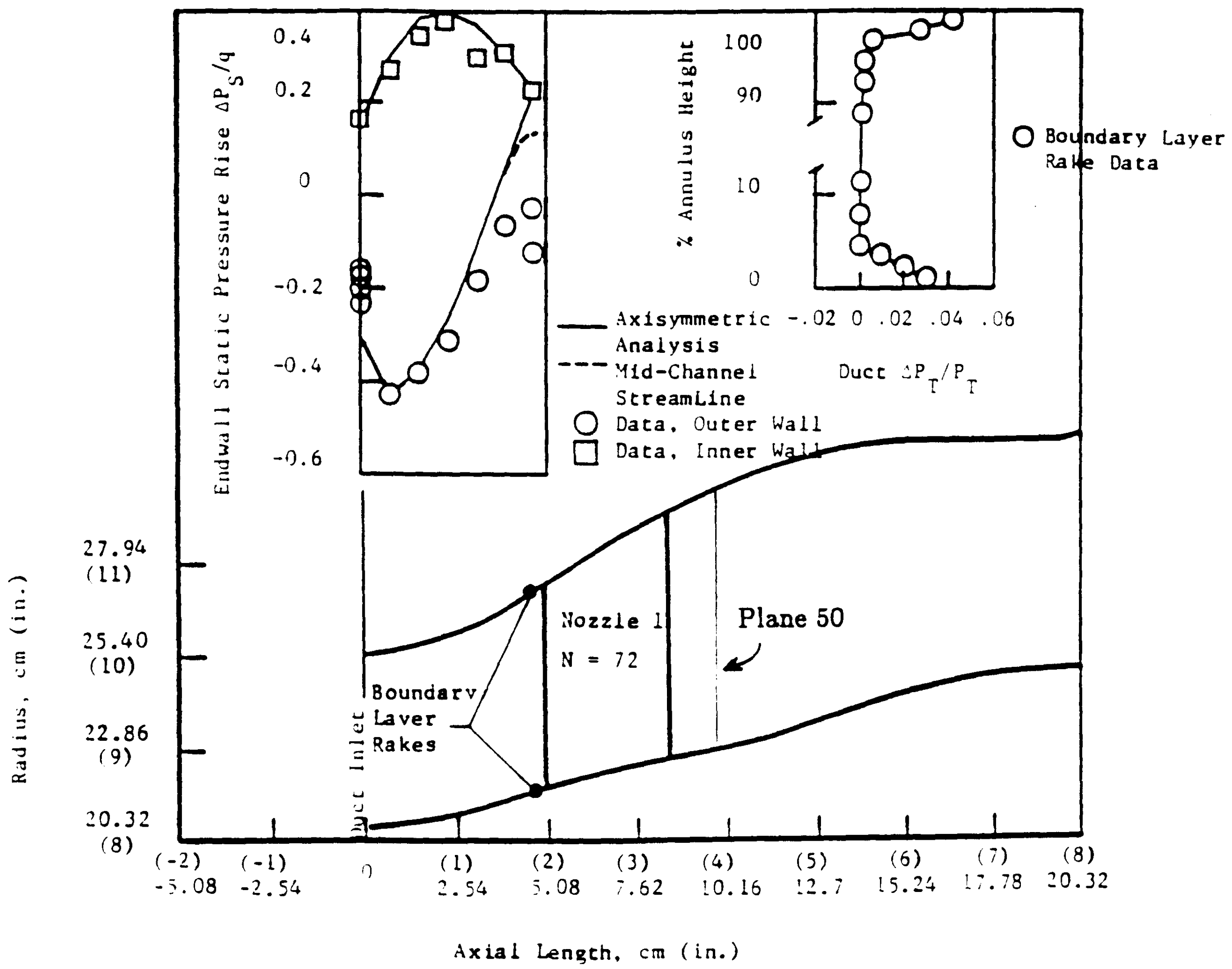


Figure 7.5: Flow path and boundary layer rake data for the E$^3$ nozzle test. From Fig. 85 in [7]. The inlet is plane 42.

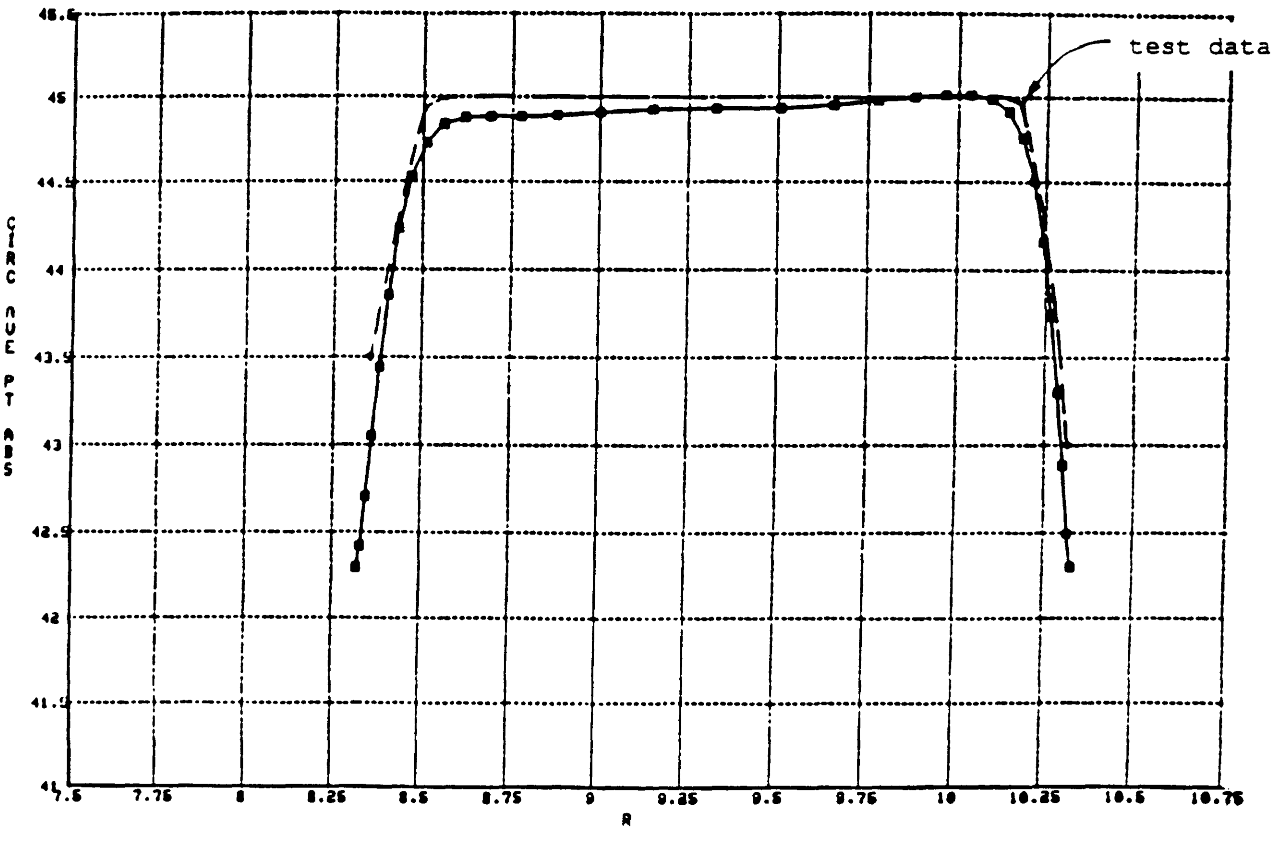


Figure 7.6: Incoming shear layer imposed at the upstream boundary for the $E^3$ test case.

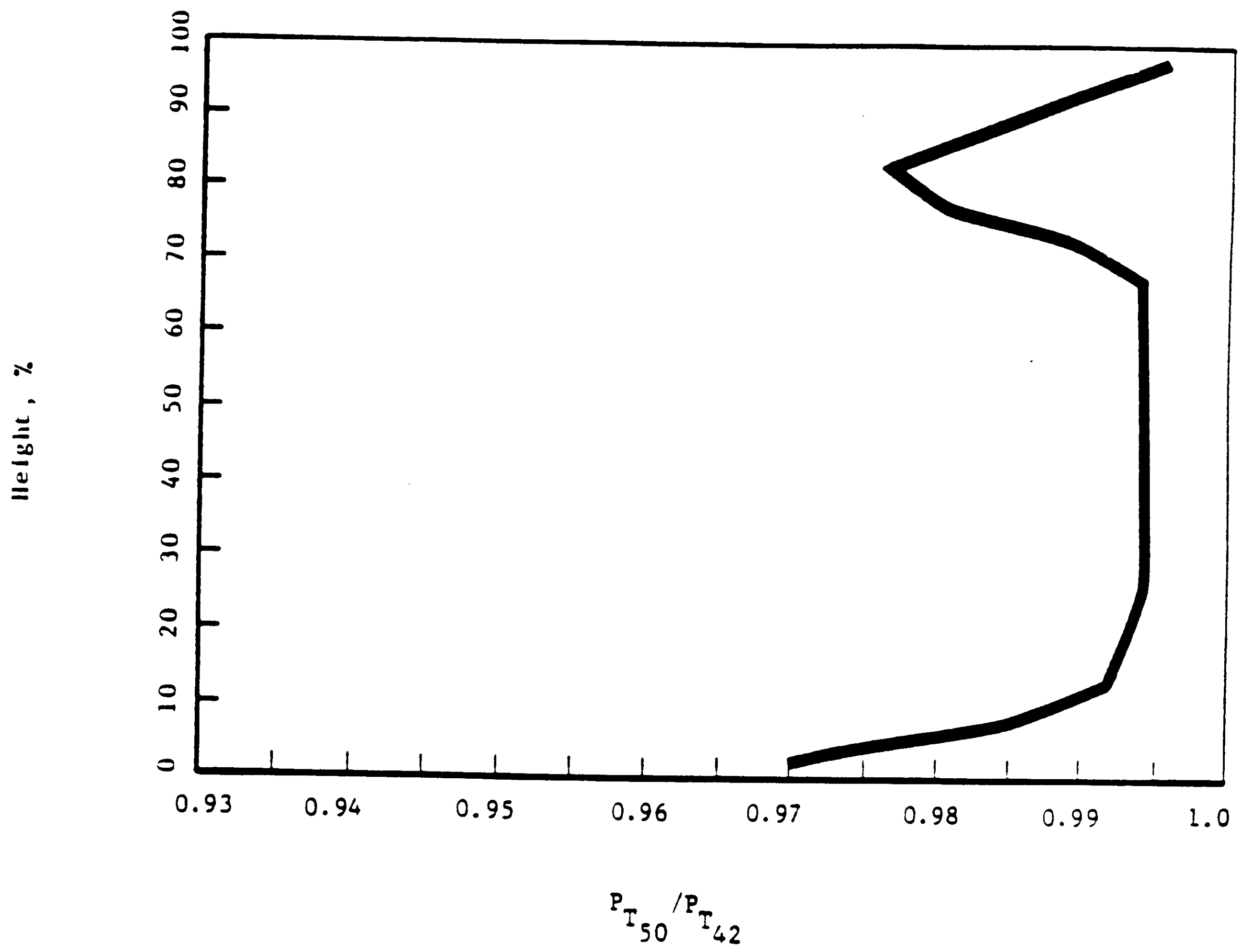


Figure 7.7: Experimental total pressure profile measured at plane 50. From Fig. 87 in [7]. Plane 42 is the inlet plane.

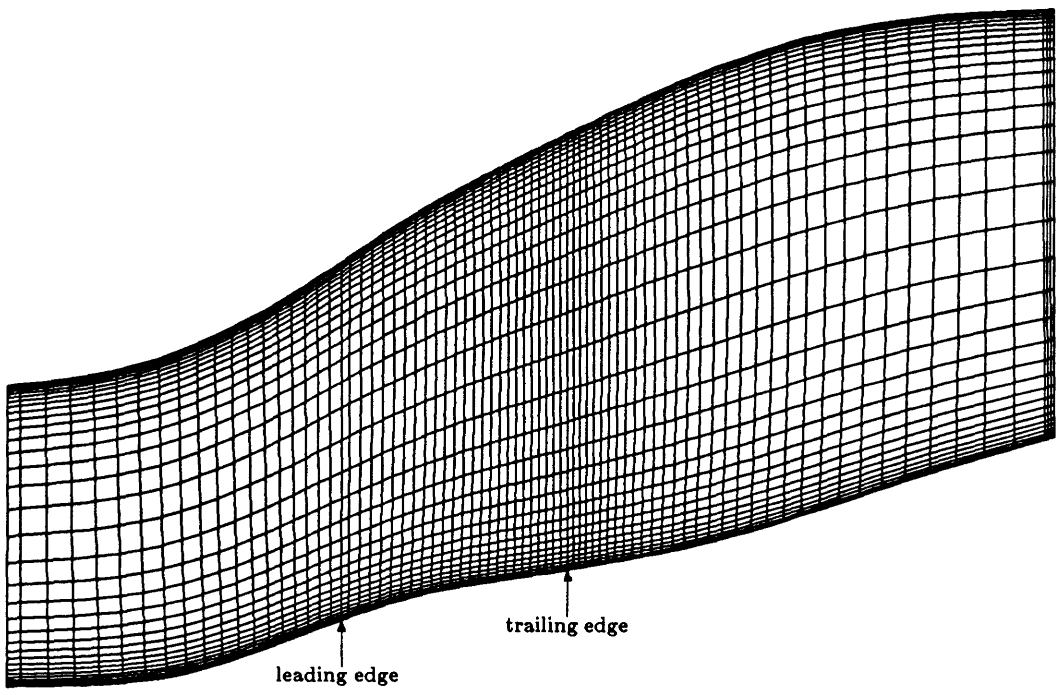


Figure 7.8: Grid in the $r$-$z$ plane used to analyze the $E^3$ LPT nozzle with incoming shear layer. ni=2, nj=33, nk=80.

nk=80). To enable the extrapolated pressure gradient boundary condition to be as realistic as possible, the grid is clustered near the exit plane. The grid at the pitch (j=17) is shown in Fig. 7.9.

The convergence history is in Table 7.1. To get this case to converge, a solution was obtained after four iterations for a duct with constant total pressure. The initial conditions are that $\psi_2$ is constant along the axial grid lines. By starting as a duct, the $\psi_2$ distribution is established. After the fourth iteration, the stagnation streamlines and leading edge were allowed to move. This calculation continued for three more iterations. The $\psi_2$ distribution was then imposed on an otherwise initial solution which had the variable upstream total pressure. After 15 total iterations, the error norm is $5.93 \times 10^{-8}$ which represents a converged solution, but not to machine accuracy. The maximum errors occur near the leading edge and the reasons for this convergence problem are probably similar to those discussed in Section 6.3. The total CPU time was 904 CRAY

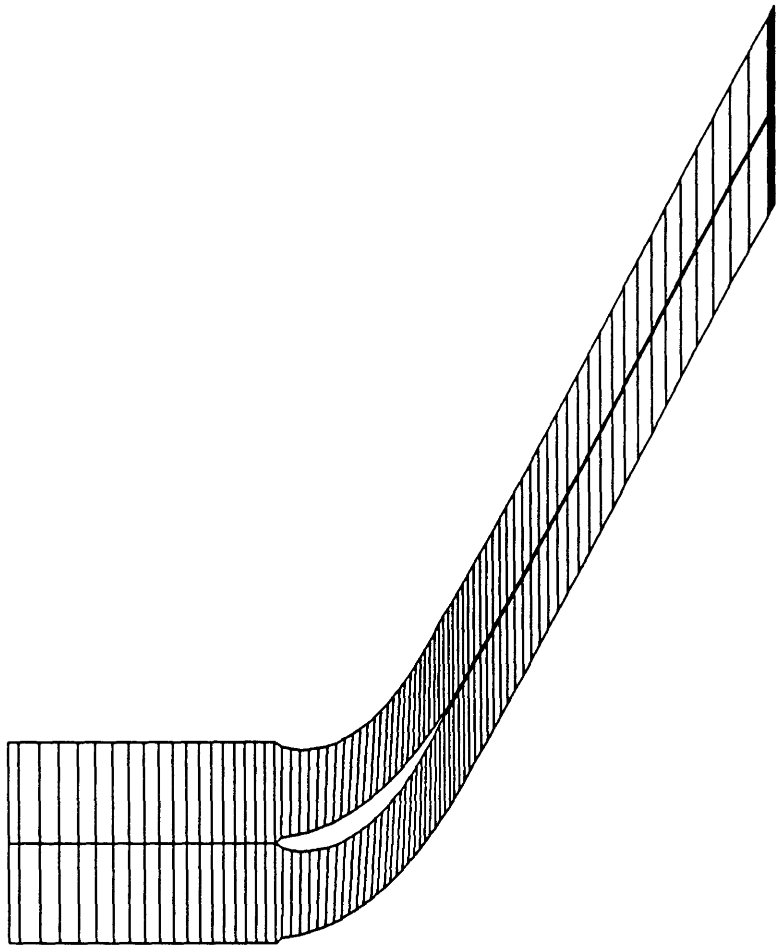


Figure 7.9: Grid at the pitch (j=17) in the $m'$-$\theta$ plane used to analyze the E$^3$ LPT nozzle with incoming shear layer. ni=2, nj=33, nk=80.

Table 7.1: Convergence history for the $E^3$ case with incoming shear layer.

Compressible flow, $S$ momentum equation applied, solution started as duct with uniform total pressure, ni=2, nj=33, nk=80, GMRES max cycles = 10, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 1.30e-3 | - | |
| 1 | 7.07e-4 | 1.0 | matrix factored |
| 2 | 8.28e-5 | 1.0 | matrix factored |
| 3 | 1.07e-6 | 1.0 | matrix factored |
| 4 | 1.05e-10 | 1.0 | matrix sol. in 8 GMRES cycles |
| 4 | 3.32e-2 | - | stagnation grid line and leading edge now allowed to move, still uniform entropy |
| 5 | 7.47e-6 | 1.0 | matrix factored |
| 6 | 6.45e-7 | 1.0 | matrix factored |
| 7 | 7.30e-8 | 1.0 | matrix factored |
| 0 | 8.71e-4 | - | variable entropy case started using $\psi_2$ from previous calculation, leading edged constrained initially |
| 1 | 1.62e-4 | 1.0 | matrix factored |
| 2 | 8.77e-6 | 1.0 | matrix factored |
| 2 | 6.50e-3 | - | leading edge now allowed to move |
| 3 | 2.36e-6 | 1.0 | matrix factored |
| 4 | 1.10e-7 | 1.0 | matrix factored |
| 5 | 8.27e-8 | 1.0 | matrix sol. in 10 GMRES cycles |
| 6 | 7.04e-8 | 1.0 | matrix sol. in 10 GMRES cycles |
| 7 | 6.36e-8 | 1.0 | matrix sol. in 10 GMRES cycles |
| 8 | 5.93e-8 | 1.0 | matrix sol. in 10 GMRES cycles |

seconds.

Contours of $\psi_2$, which represent the axisymmetric streamlines calculated, are shown in Fig. 7.10, and the pitch line Mach numbers are in Fig. 7.11. Fig. 7.12 shows contours of the entropy variable $\hat{s}$ which for this stationary case represents a normalized total presure. The reference pressure and temperature are defined at the hub, so $\hat{s} = \exp(-s/R) = 1.0$ at the hub. The shear layer convects along streamlines as it should under this "axisymmetric" assumption. This assumption does not preclude one mechanism for the secondary flow development, namely, how the secondary velocities must adjust so the total pressure is convected and continuity and momentum are satisfied in

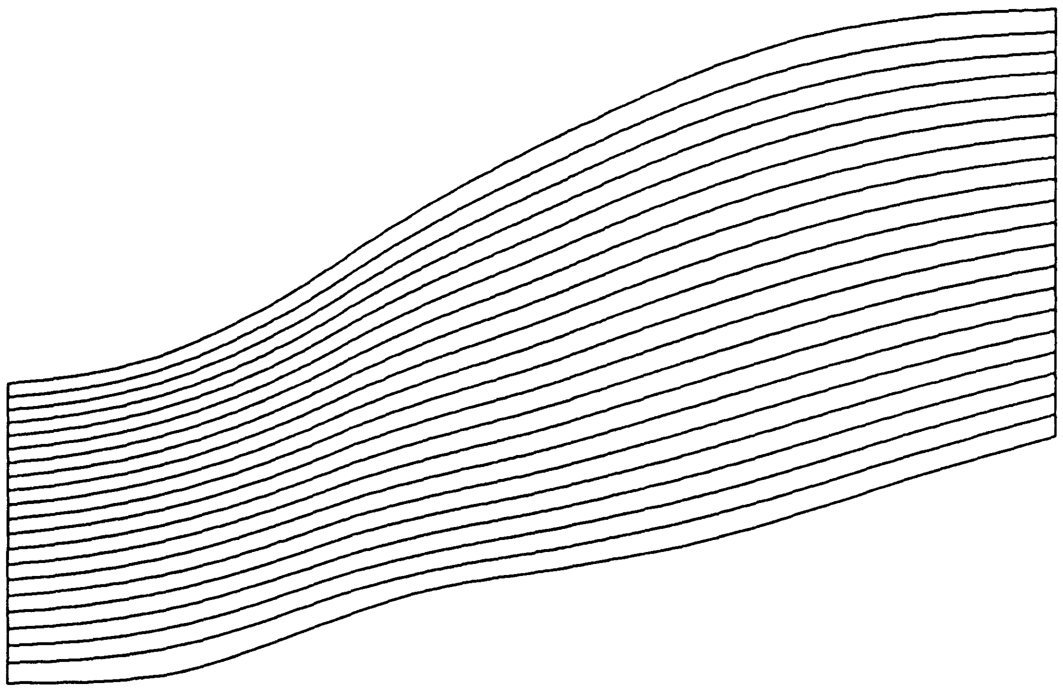


Figure 7.10: Contours of $\psi_2$ (streamlines) for the E$^3$ nozzle with incoming shear layer.

a channel with an area variation.

Figure 7.13 shows a comparison of the exit angle at plane 50 predicted using the two stream function solver, the viscous code in reference [31], results obtained using the Euler code described in reference [34], and an axisymmetric solution used to design the turbine. Also included on this plot is the measured exit angle distribution at plane 50. This exit angle distribution has been obtained from reference [7] and is based on a circumferential and radial traverse of a nulling probe. The relative angle measurements are quite accurate, but the absolute level is not. This level is usually determined by satisfying mass conservation, but the data was not published with this correction. The approximate offset is $-1.5$ degrees [11]. Much of the over-turning at the hub and tip is captured with the current approach. The effect of secondary flows due to the tip vortex are not captured so the overturning at the tip is under predicted as compared to the measurement, Euler solution and viscous flow solutions. The viscous code calculation used a $51 \times 50 \times 60$ grid, and the Euler calculation used a $25 \times 33 \times 81$ grid.

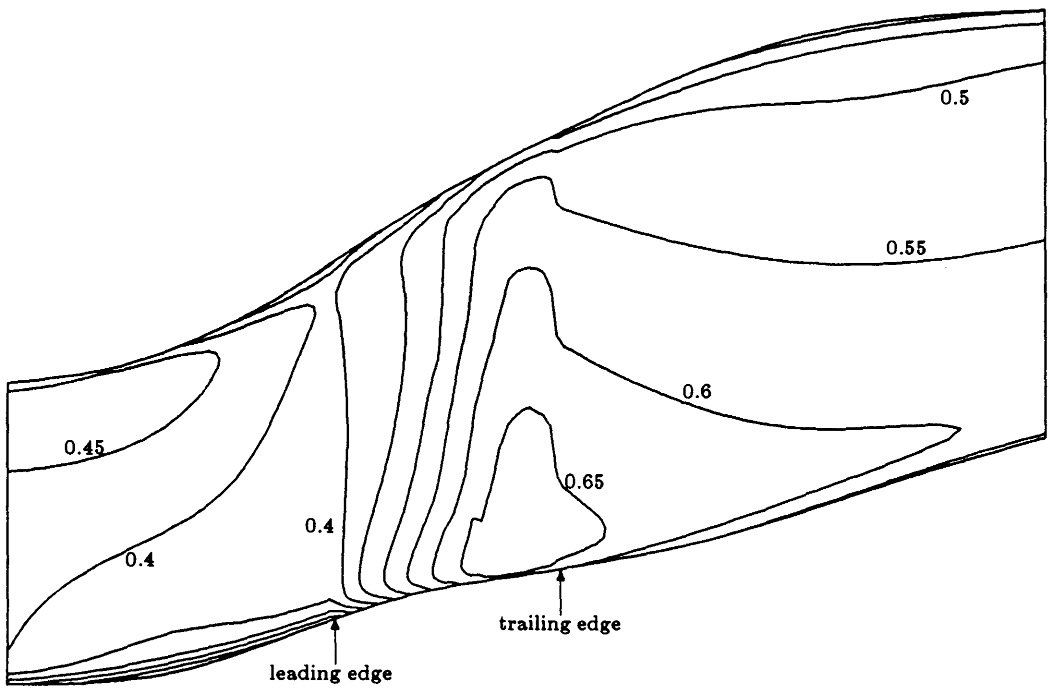


Figure 7.11: Contours of Mach number for the $E^3$ nozzle with incoming shear layer. Contour interval=0.05.

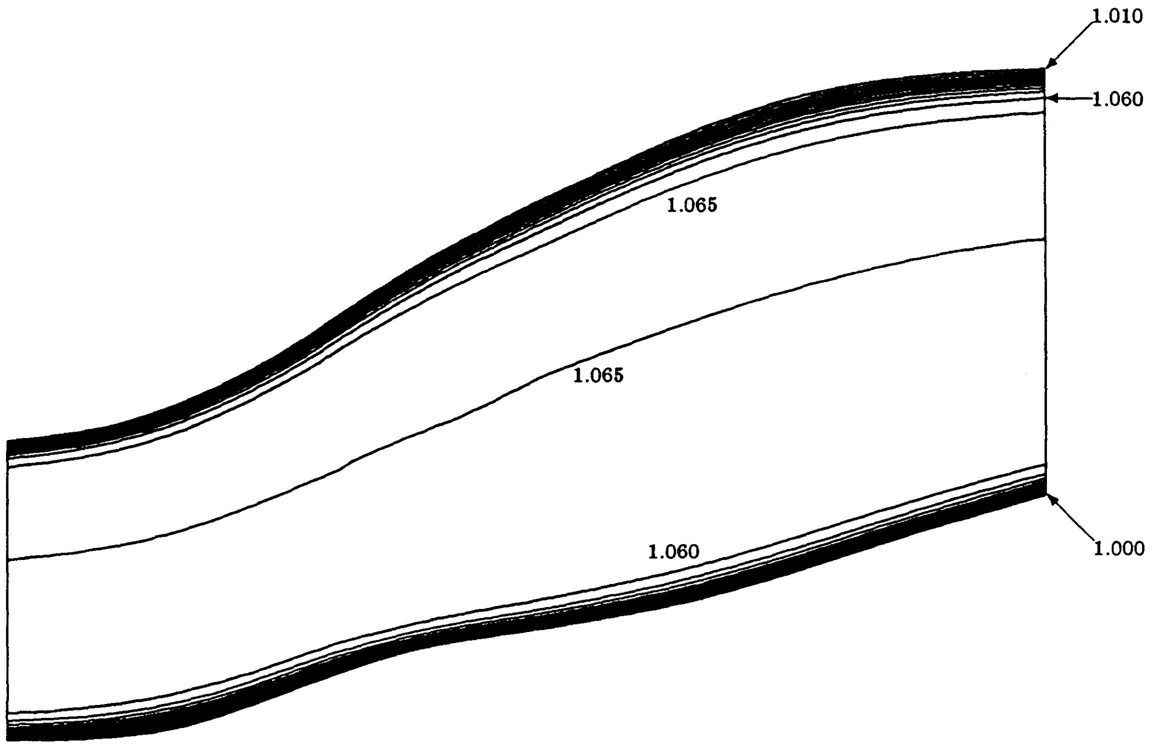


Figure 7.12: Contours of entropy variable $\hat{s}$ for the $E^3$ nozzle with incoming shear layer. Contour interval=0.005.

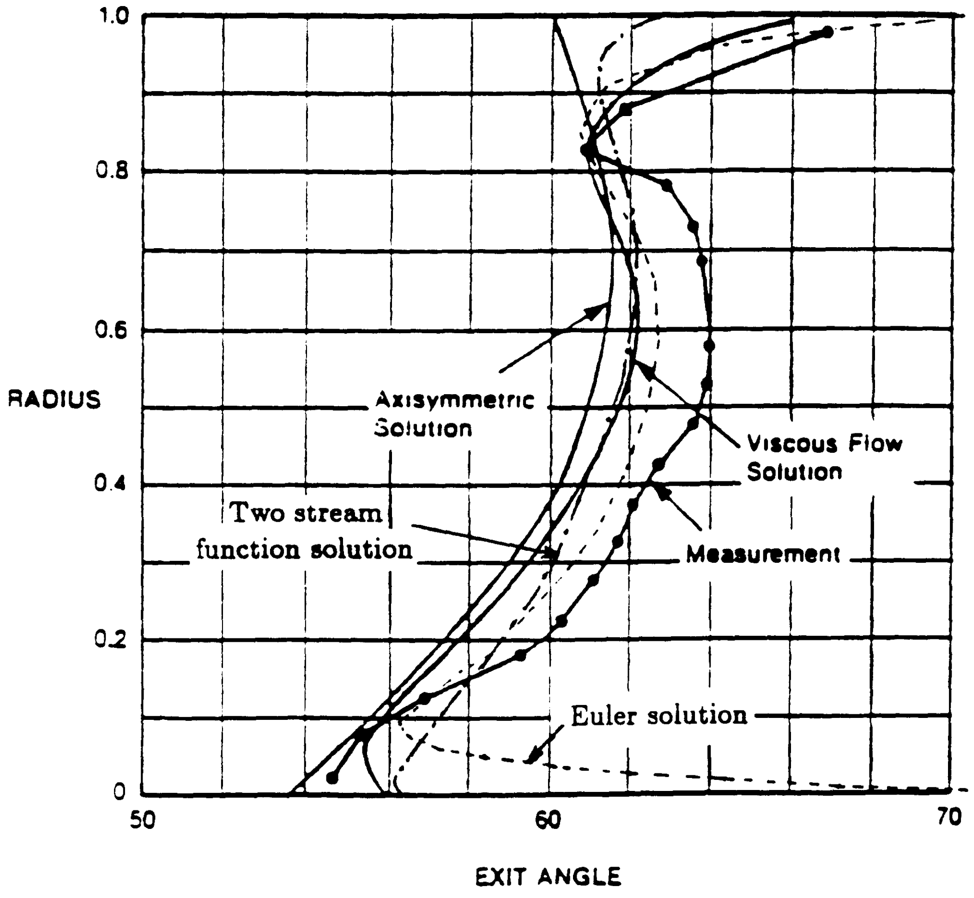
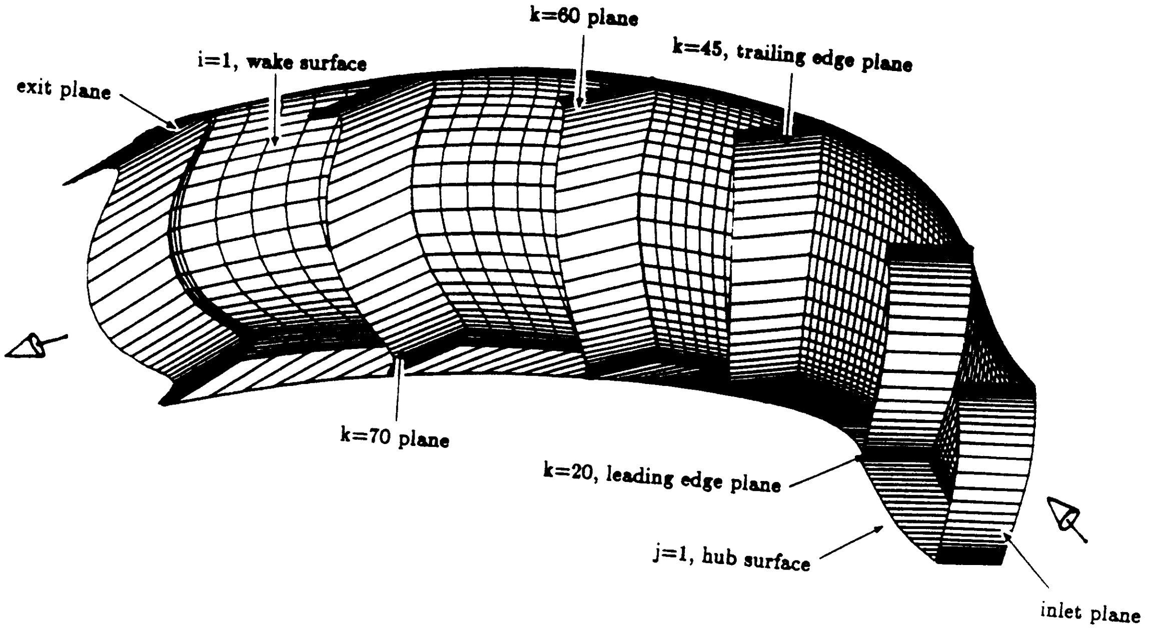


Figure 7.13: Comparison between experiment and several code predictions of the exit angle distribution at plane 50 for the $E^3$ nozzle. The measurement absolute level should be offset approximately $-1.5$ degrees [11].

The development of the secondary velocity field continues past the measurement plane. The code solves for the stagnation stream surfaces, and the axial progression of these stream surfaces can be visualized by plotting the grid in the cross stream planes. Figure 7.14 shows two 3D views of the $E^3$ grid which explains the orientation of the cross stream planes. In one of the views, the j=1 plane (hub surface) and the i=1 plane (upstream stagnation stream surface, blade pressure surface, and wake surface) are also shown. Figure 7.15 and Fig. 7.16 show the cross stream grids at several axial locations. The orientation of these grid surfaces is aft looking forward or downstream looking upstream, whereas in Fig. 7.14, the projection seen is the upstream-looking face.



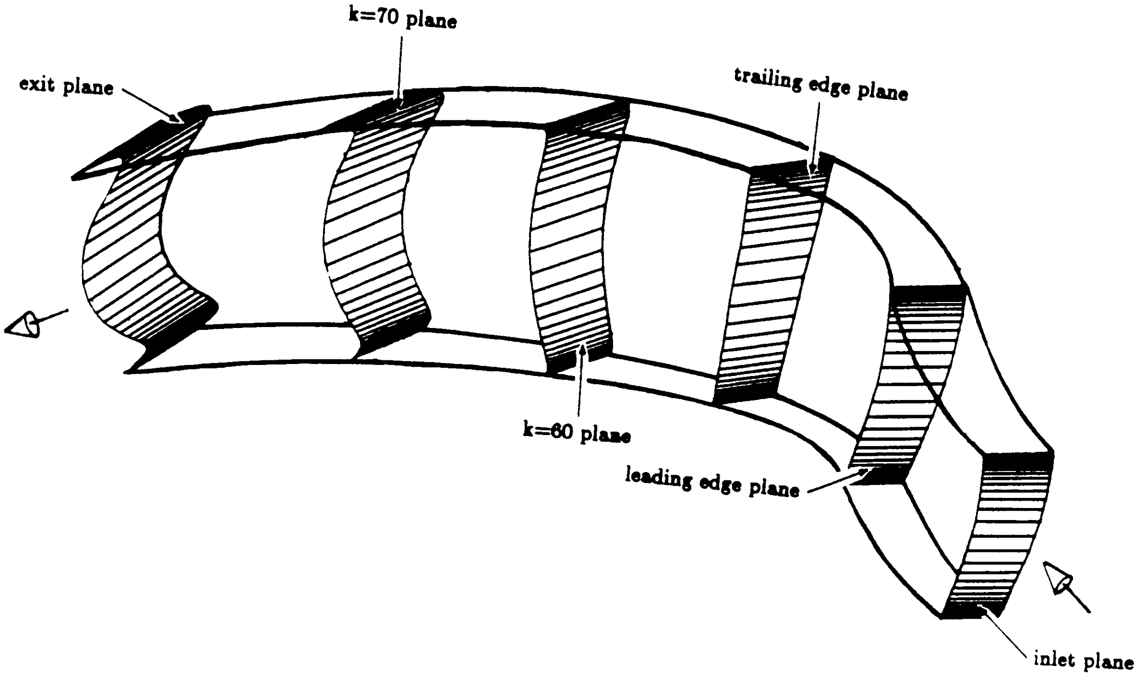


Figure 7.14: Three dimensional views of the $E^3$ grid showing the cross stream grid plane orientation. The upstream projections of these cross stream planes are shown. The j=1 plane (hub surface), and i=1 plane (upstream stagnation stream surface, blade pressure surface and wake) are also shown in the top figure.

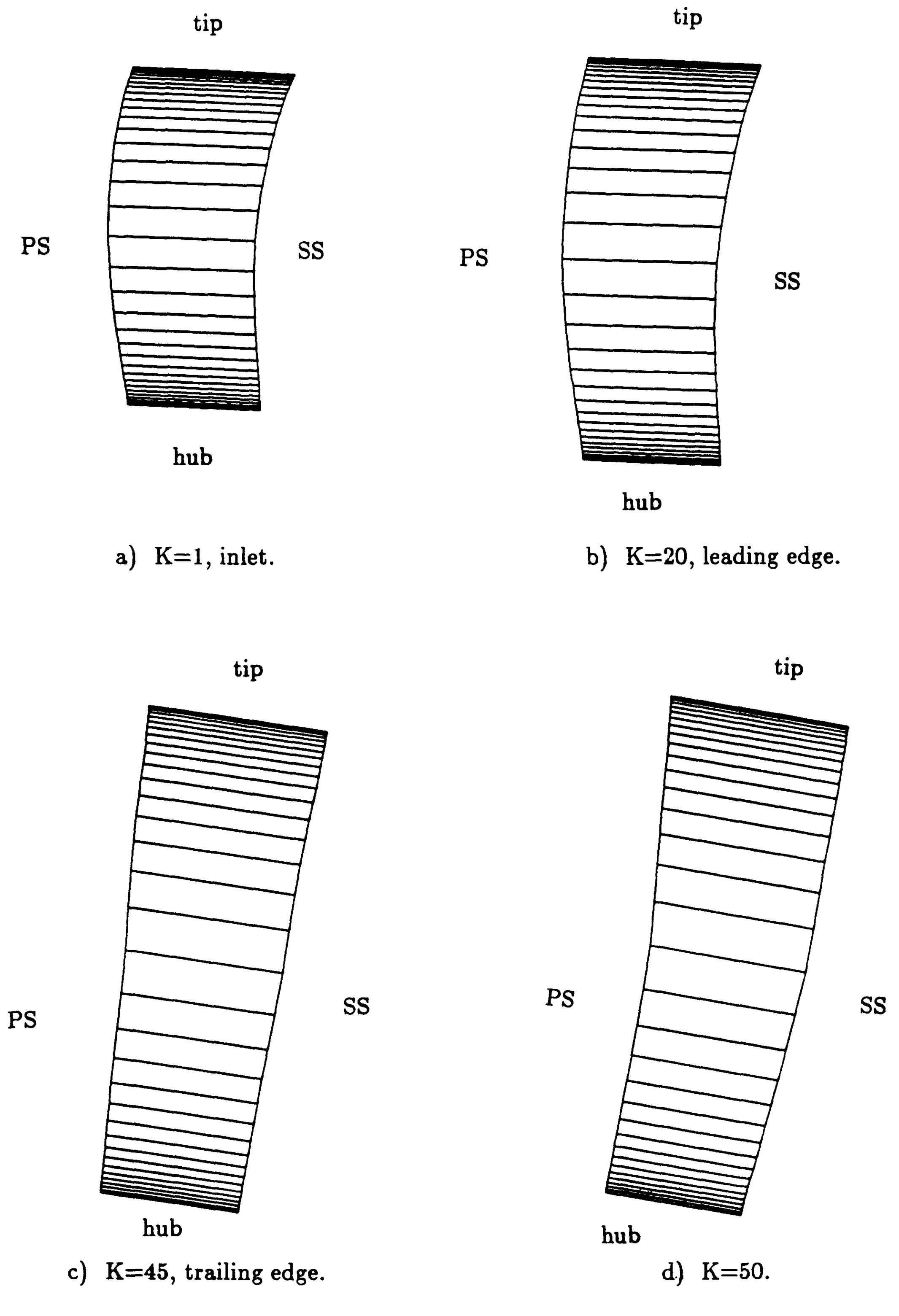
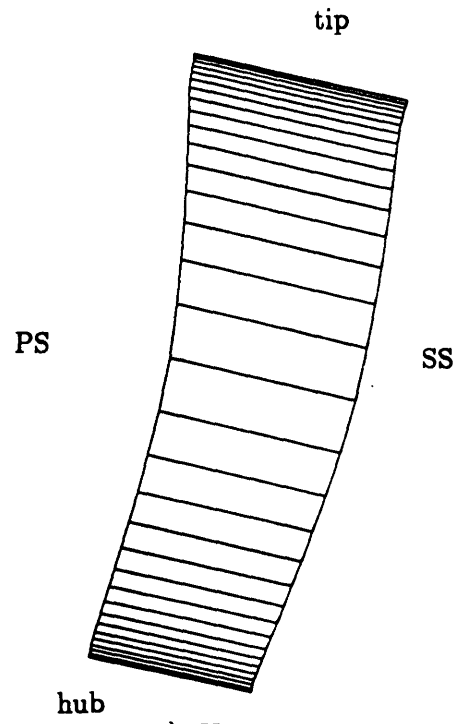
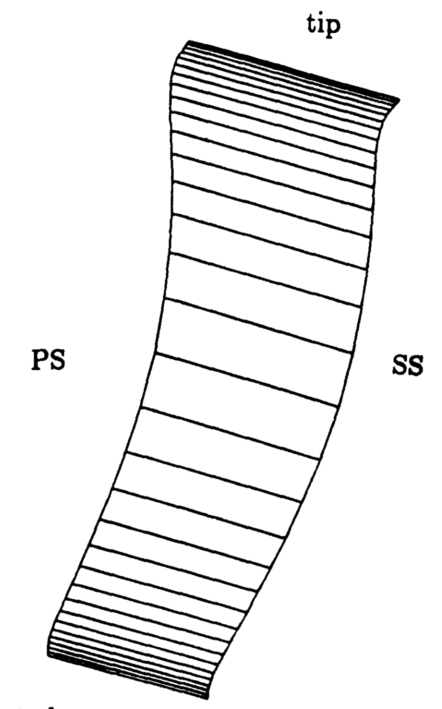


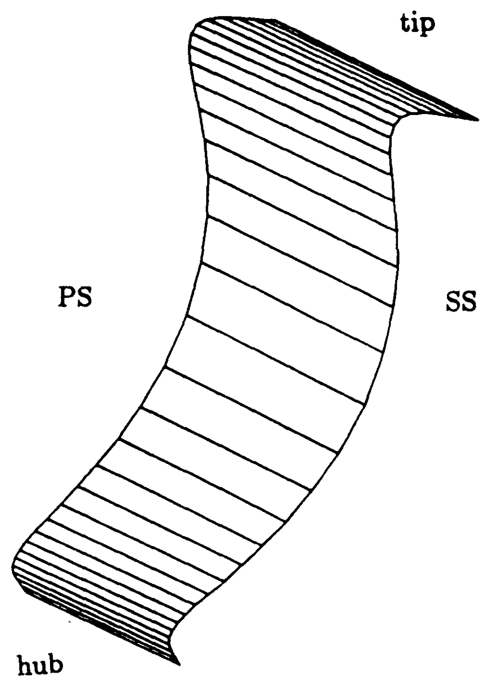
Figure 7.15: Cross stream grid lines at several axial locations showing the stagnation stream surfaces for the $E^3$ nozzle with incoming shear layer, I. Aft looking forward, PS and SS are the pressure and suction side respectively.



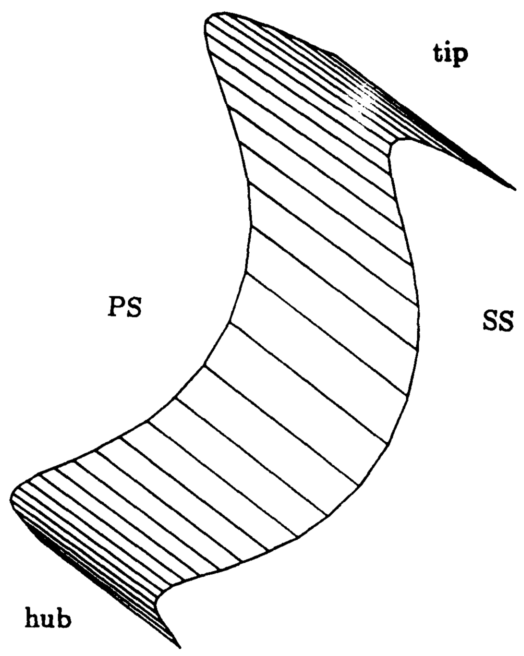
a) K=55.



b) K=60.



c) K=70.



d) K=80, exit.

Figure 7.16: Cross stream grid lines at several axial locations showing the stagnation stream surfaces for the $E^3$ nozzle with incoming shear layer, II. Aft looking forward, PS and SS are the pressure and suction side respectively.

### 7.2.1 Full 3D Calculation

The $E^3$ nozzle with an incoming shear layer could not be run with more than one volume blade-to-blade. To demonstrate the full 3D capability of the algorithm, the $E^3$ case has been run with a uniform total pressure. The flow rate was reduced to one tenth of the design value. A coarse grid was used due to memory restrictions of the direct solver. The grid has 6 nodes blade-to-blade, 7 nodes spanwise, and 34 nodes axially. The grid in the $r$-$z$ plane for i=1 is shown in Fig. 7.17. Both the inlet and exit boundaries are placed closer to the blade than for the previous solution so the grid size could be reduced. The grid in $m'$-$\theta$ at the hub is shown in Fig. 7.18.

The initial $\psi_2$ distribution was established using results obtained on the same axial and spanwise grid, but with ni=2. The convergence history for this "axisymmetric" calculation is in Table 7.2. Table 7.3 shows the convergence history for the full 3D calculation. The errors start out low due to the initialization of $\psi_2$ and because of the low flow rate. The solution is converged to an engineering accuracy. Convergence is limited, in this case, by the use of the extrapolated pressure gradient boundary condition which is applied in a region where radius and curvature are still changing. This is not strictly correct and is inconsistent with the entropy convection equation. The residual errors, therefore, cannot be reduced to machine zero. Table 7.4 shows the convergence history for the ni=2 case using the $S$ momentum equation. Notice that these errors are below those in Table 7.2. This is because the use of the $S$ momentum equation is not as constraining as the entropy convection equation. Convergence problems are still encountered, but near the leading edge as in the case with the incoming shear layer. The entropy convection equation was applied to the 3D case to ensure that no artificial streamwise vorticity was created which would cause the stream functions to warp severely and diverge. Other downstream boundary conditions have been attempted, as well as modifying the exit flow path to being cylindrical. These approaches were worse, and the current method seems to be the best overall approach tried, as long as the residuals are below an engineering tolerance. The solution time for this 3D calculation was 432 CRAY seconds.

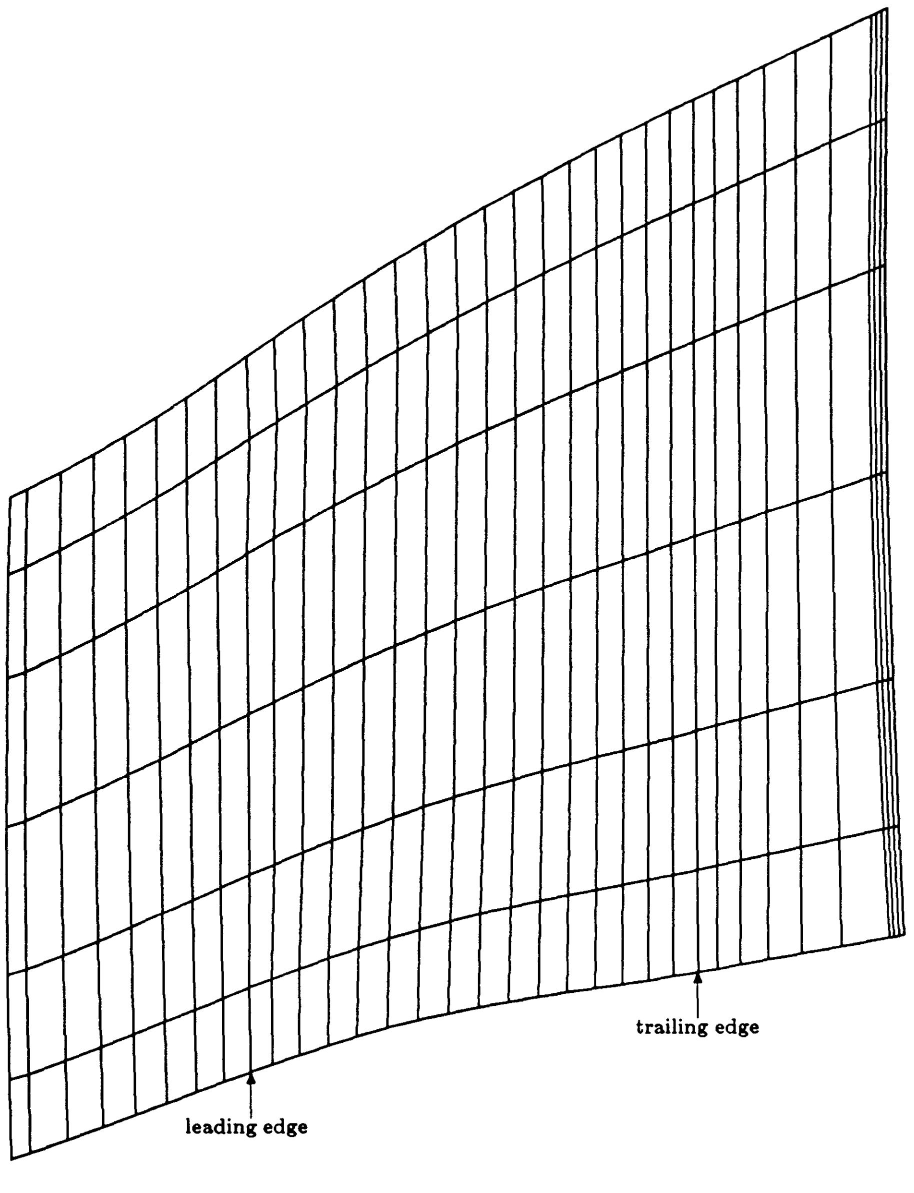


Figure 7.17: Grid in $r$-$z$ plane for i=2 for the full 3D calculation of the $E^3$ nozzle with uniform total pressure.

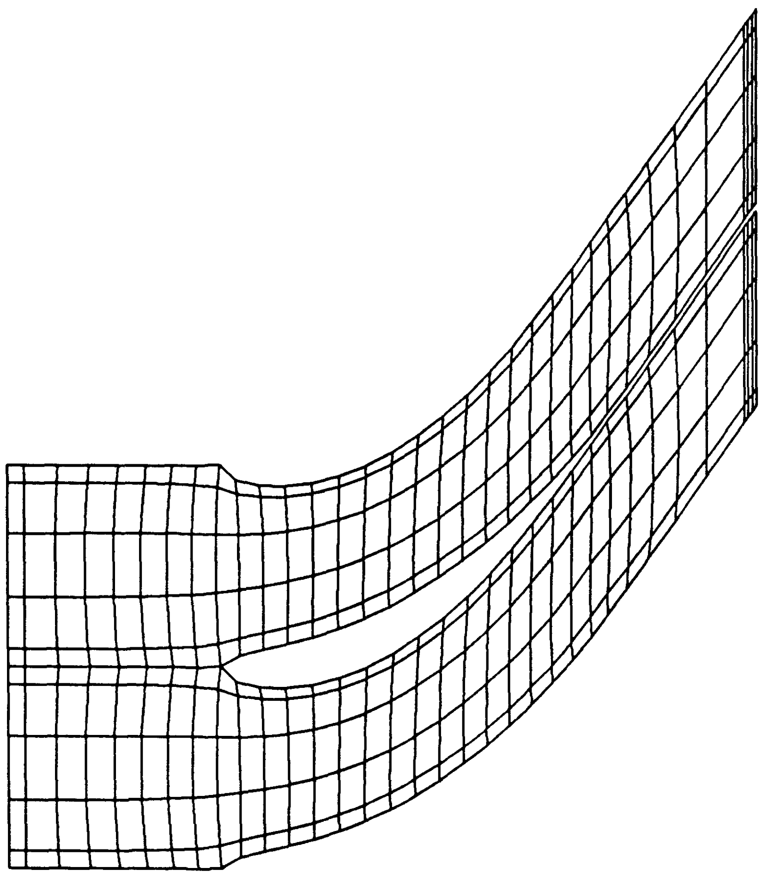


Figure 7.18: Grid in $m'$-$\theta$ plane at the hub for the full 3D calculation of the $E^3$ nozzle with uniform total pressure.

Table 7.2: Convergence history for the $E^3$ case with uniform total pressure, ni=2, entropy convection.

Compressible flow, entropy convection equation applied, ni=2, nj=7, nk=34, GMRES max cycles = 10, line search algorithm used initially, after that $\lambda = 0.8$ prescribed.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 2.91e-5 | - | |
| 1 | 1.33e-5 | 1.0 | matrix factored |
| 2 | 3.02e-6 | 1.0 | matrix factored |
| 3 | 1.40e-6 | 1.0 | matrix factored |
| 4 | 1.40e-6 | 0.0 | matrix factored |
| | | | line search could not reduce residual norm. |
| 4 | 1.40e-6 | - | solution restarted with no line search used |
| 5 | 1.45e-6 | 0.8 | matrix factored |
| 6 | 1.03e-6 | 0.8 | matrix sol. in 10 GMRES cycles |
| 7 | 2.07e-7 | 0.8 | matrix sol. in 10 GMRES cycles |
| | | | engineering tolerance achieved |

Table 7.3: Convergence history for the full 3D $E^3$ case with uniform total pressure.

Compressible flow, entropy convection equation applied, leading edge initially constrained, ni=6, nj=7, nk=34, GMRES max cycles = 10, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 3.21e-5 | - | |
| 1 | 2.37e-5 | 0.28 | matrix factored |
| 2 | 2.13e-5 | 1.0 | matrix factored |
| 3 | 4.16e-6 | 1.0 | matrix factored |
| 4 | 9.14e-7 | 1.0 | matrix factored |
| 5 | 7.58e-7 | 0.37 | matrix factored |
| 6 | 5.23e-7 | 0.49 | matrix sol. in 8 GMRES cycles |
| 6 | 4.75e-6 | - | leading edge now allowed tomove |
| 7 | 2.68e-6 | 1.0 | matrix factored |
| 8 | 1.04e-6 | 1.0 | matrix sol. in 10 GMRES cycles |
| 9 | 9.98e-7 | 0.1 | matrix sol. in 10 GMRES cycles |
| 10 | 9.98e-7 | 0.0 | matrix factored |
| | | | line search could not decrease residual norm |

Table 7.4: Convergence history for the $E^3$ case with uniform total pressure, ni=2 $S$ momentum equation applied.

Compressible flow, $S$ momentum equation applied, ni=2, nj=7, nk=34, GMRES max cycles = 10, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 2.91e-5 | - | |
| 1 | 1.49e-6 | 1.0 | matrix factored |
| 2 | 6.60e-8 | 1.0 | matrix factored |
| 3 | 3.44e-8 | 1.0 | matrix sol. in 10 GMRES cycles |
| 4 | 2.22e-8 | 1.0 | matrix sol. in 9 GMRES cycles |
| 5 | 1.63e-8 | 1.0 | matrix sol. in 10 GMRES cycles |
| 6 | 1.29e-8 | 1.0 | matrix sol. in 9 GMRES cycles |
| 7 | 1.08e-8 | 1.0 | matrix sol. in 10 GMRES cycles |
| 8 | 9.35e-9 | 1.0 | matrix sol. in 10 GMRES cycles |
| 9 | 8.24e-9 | 1.0 | matrix sol. in 10 GMRES cycles |
| 10 | 7.35e-9 | 1.0 | matrix sol. in 10 GMRES cycles |
| 11 | 6.60e-9 | 1.0 | matrix sol. in 10 GMRES cycles |
| 12 | 5.95e-9 | 1.0 | matrix sol. in 10 GMRES cycles |
| 13 | 5.37e-9 | 1.0 | matrix sol. in 10 GMRES cycles |
| 14 | 4.85e-9 | 1.0 | matrix sol. in 10 GMRES cycles |
| 15 | 4.38e-9 | 1.0 | matrix sol. in 10 GMRES cycles |

The convergence is very sensitive for a full 3D calculation. Any artificial vorticity layers seemed to cause the algorithm to diverge. Also the constants, $\kappa_A$ and $\kappa_B$, in the auxiliary pressure equations must be set to one to constrain the odd-even stream function modes. In 2D, a value of 0.05 for either $\kappa_A$ or $\kappa_B$ is satisfactory.

The Mach number contours in the $r$-$z$ plane for i=3 are shown in Fig. 7.19. Figure 7.20 shows the Mach number contours at mid span (j=4). Contours of $\psi_1$ and $\psi_2$ are shown in Fig. 7.21 (these are actual contours projected on the cross stream grid planes) which shows how the stream sheets warp even without any streamwise vorticity, and is due to the 3D loading variations. The $\psi_1$, $\psi_2$ distribution is essentially uniform downstream, and is partially specified and partially determined by the downstream extrapolated pressure gradient boundary condition (see Fig. 7.21 (d)). The warping can be seen to take place through the blade row going from the trailing edge to leading edge.

For comparison, Fig. 7.22 shows the Mach contours for the "axisymmetric" case with ni=2 which also used the entropy convection equation. It is very similar to Fig. 7.19, and the difference can be attributed to the blade-to-blade resolution.

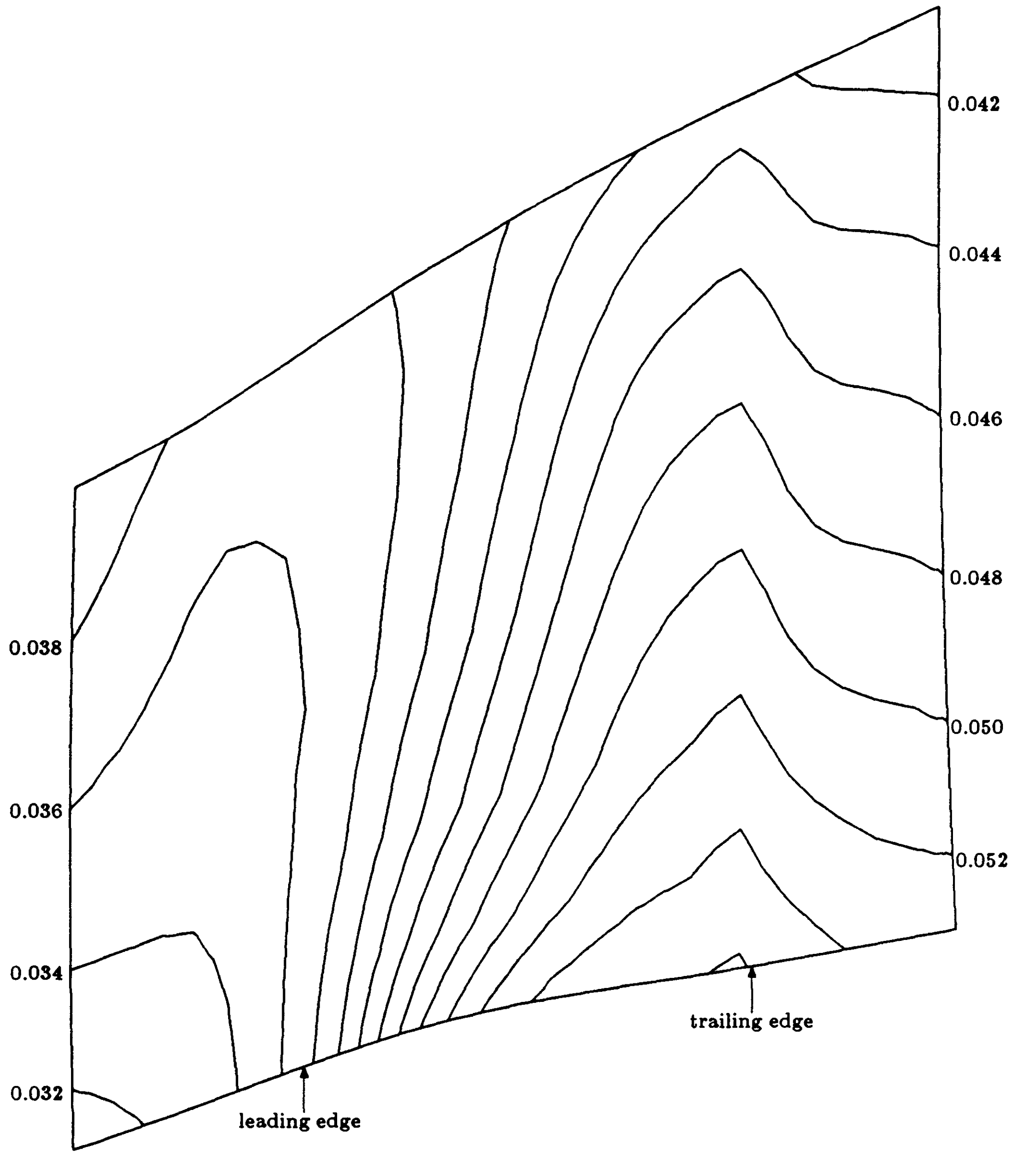


Figure 7.19: Mach number contours in the $r$-$z$ plane for i=3 for the full 3D calculation of the $E^3$ nozzle with uniform total pressure. Contour interval=0.002.

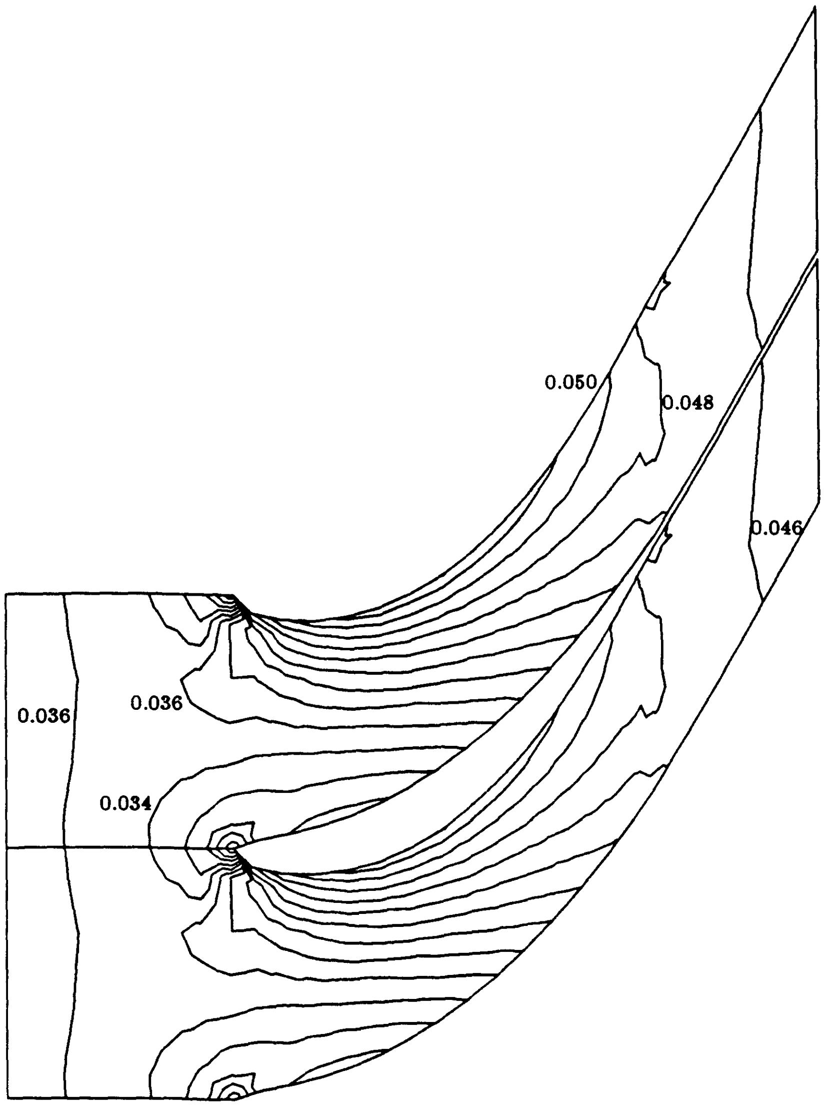


Figure 7.20: Mach number contours in the $m'$-$\theta$ plane at mid span (j=4) for the full 3D calculation of the $E^3$ nozzle with uniform total pressure. Contour interval = 0.002.

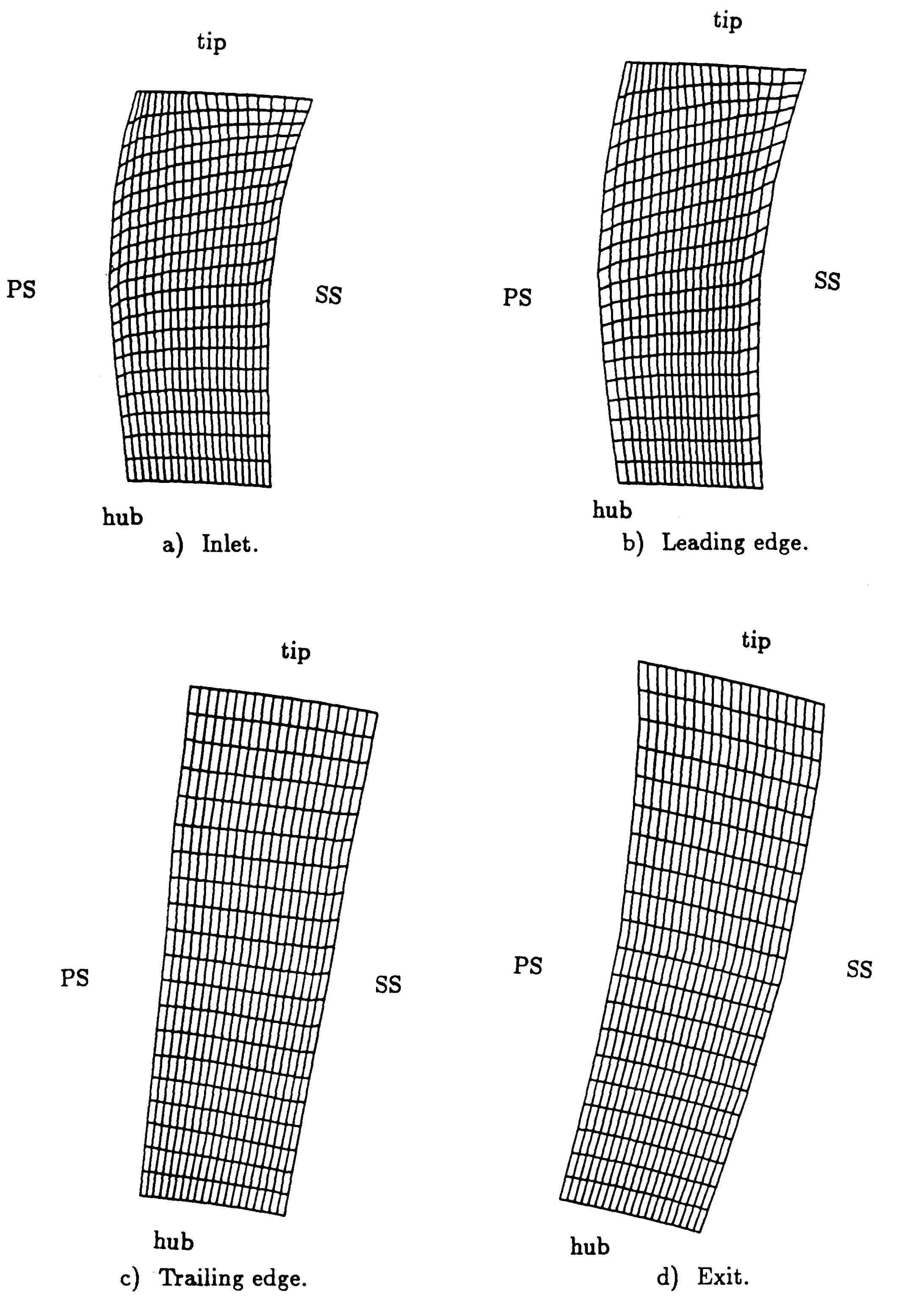


Figure 7.21: Contours of $\psi_1$ and $\psi_2$ representing stream surfaces at several axial locations for the full 3D calculation of the $E^3$ nozzle with uniform total pressure. Aft looking forward, PS and SS are the pressure and suction side respectively.

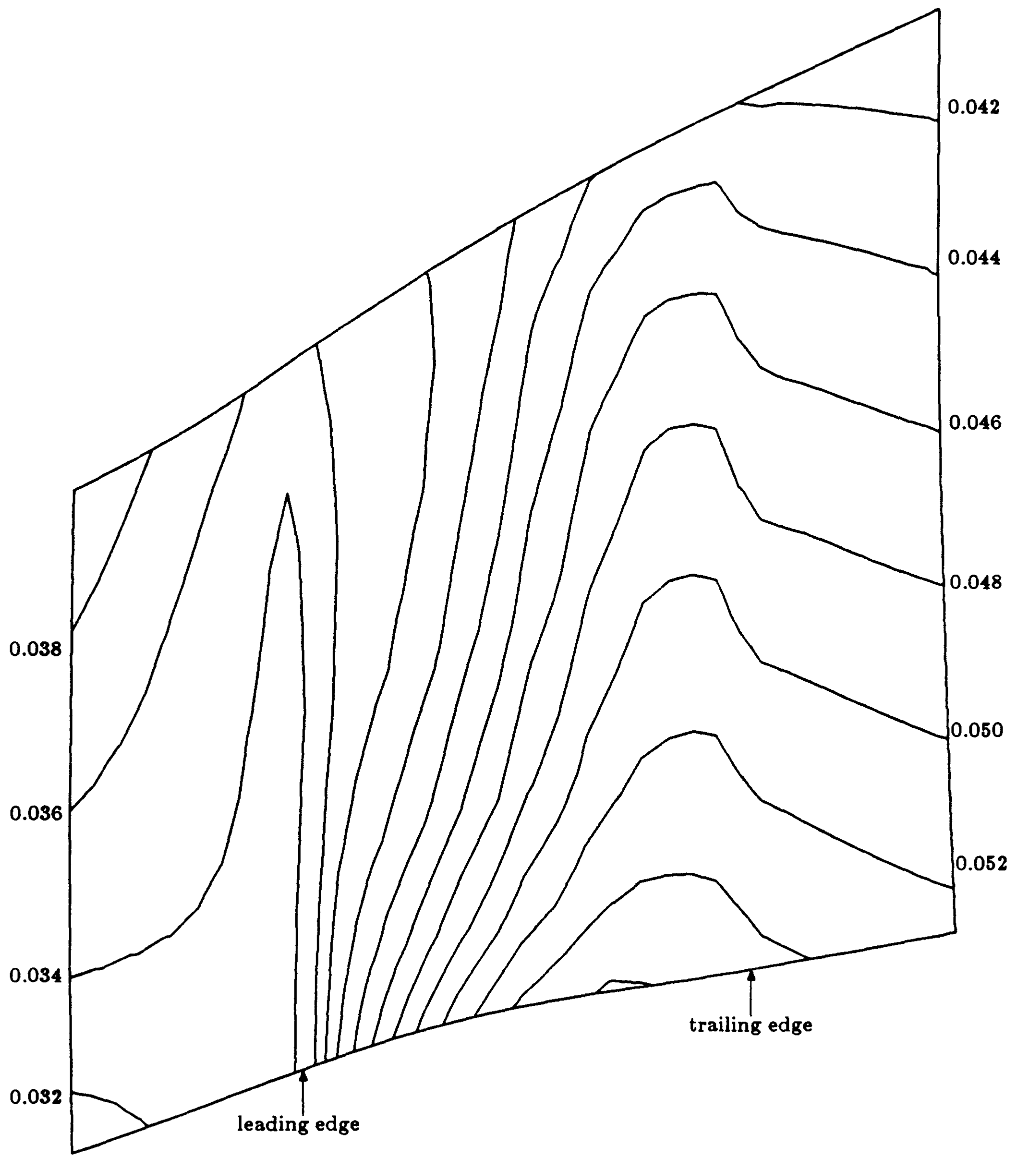


Figure 7.22: Mach number contours in the $r$-$z$ plane for for the one blade-to-blade volume calculation of the $E^3$ nozzle with uniform total pressure. Contour interval=0.002.

## 7.3 NASA 67 Transonic Fan

NASA and other research institutions have performed extensive rig tests of a transonic fan rotor (the NASA 67 rotor). The NASA testing is described by Strazisar in reference [58]. The meridional view of the test flow path is in Fig. 7.23. Numerical simulations have also been performed [49] [65] using an Euler solver with and without boundary layer coupling. Because of the quality of the experimental results, this rotor is being widely used as a benchmark test case to validate numerical solvers. This rotor will be analyzed using the two stream function solver in its "axisymmetric" mode (i.e. one volume in the blade-to-blade passage). Even though the inlet total pressure and rothalpy are uniform, the streamwise vorticity generated by the shocks prevents the solver from being run in the full 3D mode.

The peak efficiency flow condition will be used. The mass flow is 34.55 kg/s, and the design tip speed of 429 m/s at a diameter of 51.3 cm defines the wheel speed. The $2 \times 21 \times 54$ grid is shown in the $r$-$z$ plane in Fig. 7.24. Figure 7.25 shows the grid in the $m'$-$\theta$ plane for the pitch (j=11). The resulting stagnation streamlines are also shown in this plot.

The convergence history is in Table 7.5. Converging to machine zero was again a problem as it was for the other 3D cases and the T7 turbine cascade. In this case, a zero absolute tangential velocity ($V_\theta = 0$) boundary condition was applied upstream. The maximum residuals for the total pressure boundary condition would not be reduced where the relative Mach number went through unity. This is a highly nonlinear regime and can conceivably cause problems for a Newton solver. To demonstrate that this was the problem, the relative flow angle boundary condition was applied using the angles calculated with the solution at iteration 15. The residual errors in the total pressure equation reduced close to machine precision ($0.17 \times 10^{-12}$) at iteration 21 as shown in Table 7.6. The angle boundary condition did not present any problems even near the sonic line. However, the convergence was still a problem and the maximum residuals are near the leading edge as occurred with other solutions. If the periodic boundaries are replaced by solid walls to create a 3D duct, the solution converges quadratically to

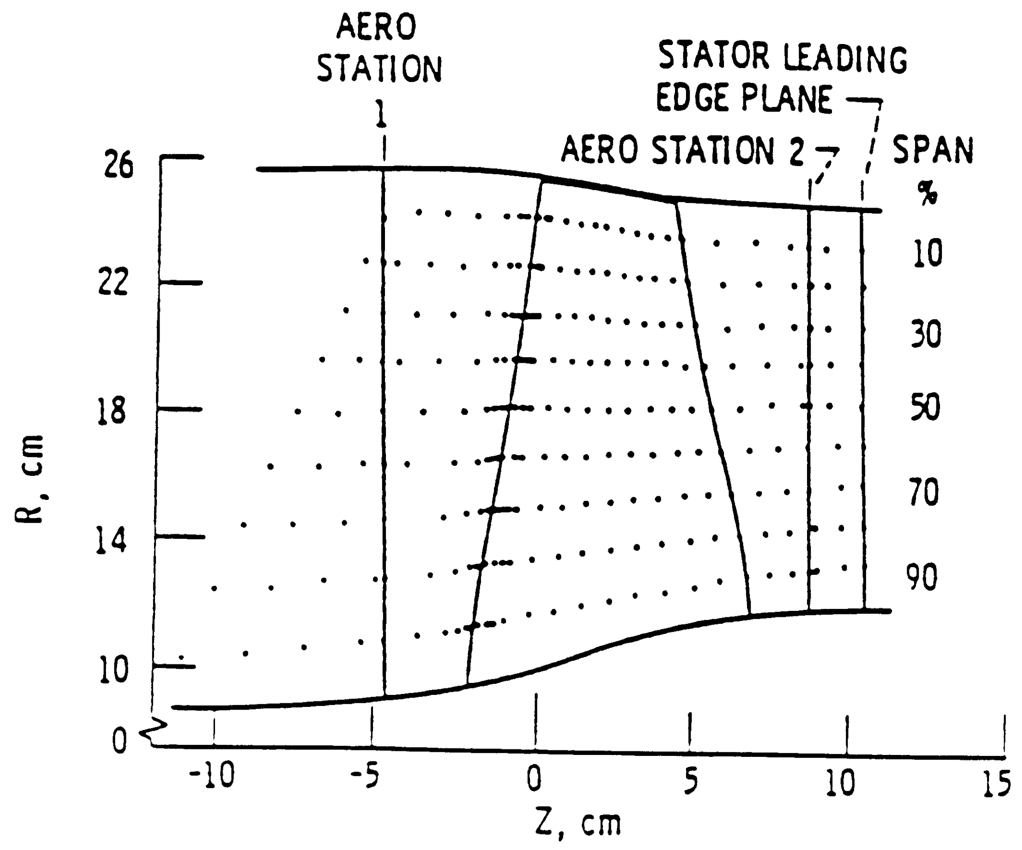


Figure 7.23: Meridional view of the NASA 67 test rig flow path showing survey locations. From Fig. 1 in reference [58] .

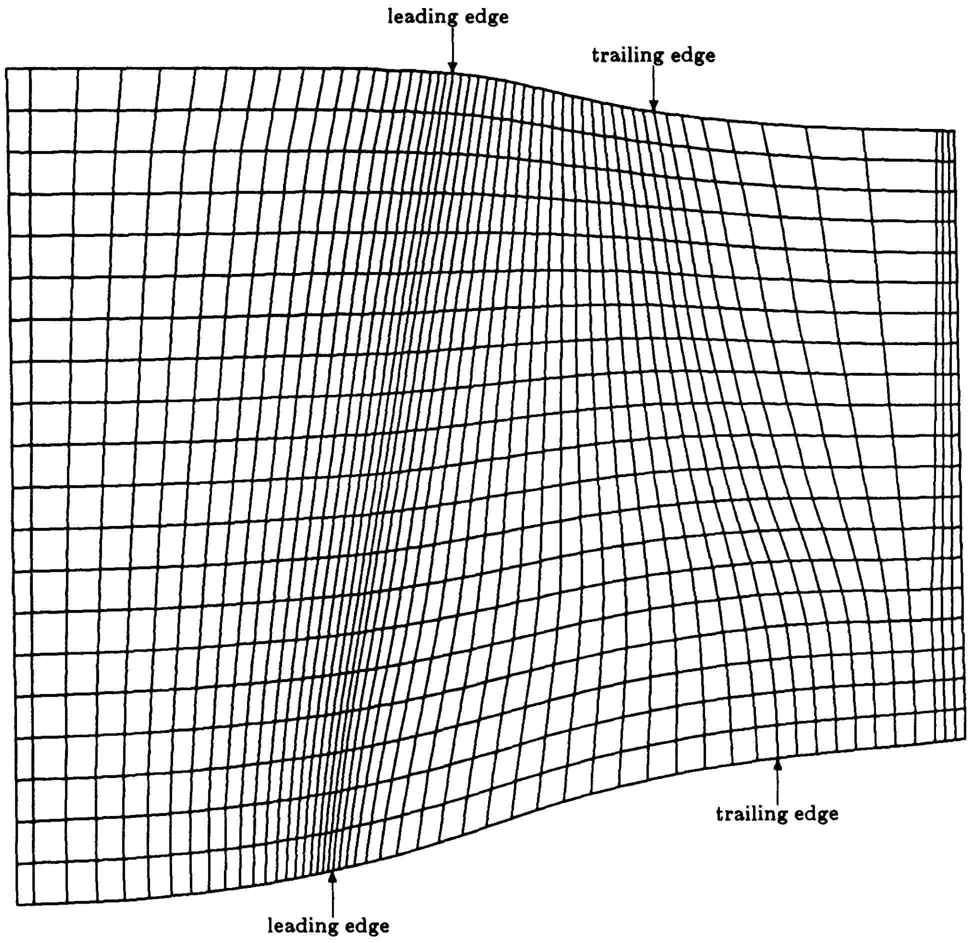


Figure 7.24: Grid in the *r*-*z* plane used to analyze the NASA 67 rotor. ni=2, nj=21, nk=54.

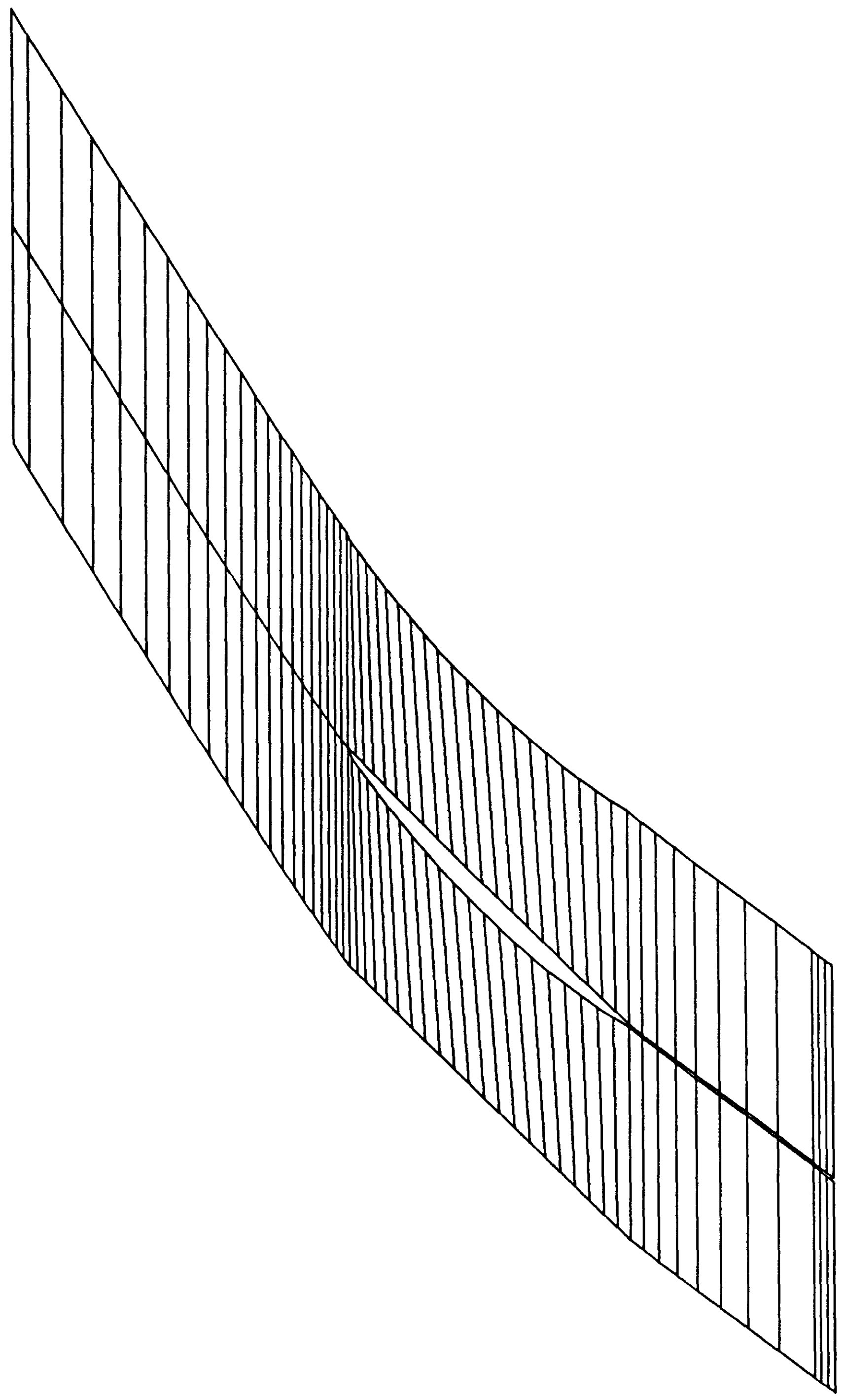


Figure 7.25: Grid at the pitch (j=11) in the $m'$-$\theta$ plane used to analyze the NASA 67 rotor. ni=2, nj=21, nk=54.

Table 7.5: Convergence history for the NASA 67 rotor.

Compressible flow, $S$ momentum equation applied, $V_\theta|_{\text{inlet}} = 0$, ni=2, nj=21, nk=54, GMRES max cycles = 10, line search algorithm used, $M_c^2 = 0.8$, leading edge movement constrained.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 1.24e-2 | - | |
| 1 | 2.25e-3 | 1.0 | matrix factored |
| 2 | 3.80e-4 | 1.0 | matrix factored |
| 3 | 3.59e-4 | 1.0 | matrix factored |
| 4 | 7.70e-5 | 1.0 | matrix factored |
| 5 | 3.80e-5 | 1.0 | matrix factored |
| 6 | 2.60e-5 | 1.0 | matrix factored |
| 7 | 1.94e-5 | 1.0 | matrix sol. in 7 GMRES cycles |
| 8 | 1.50e-5 | 1.0 | matrix sol. in 8 GMRES cycles |
| 9 | 1.18e-5 | 1.0 | matrix sol. in 8 GMRES cycles |
| 10 | 9.50e-6 | 1.0 | matrix sol. in 9 GMRES cycles |
| 11 | 7.83e-6 | 1.0 | matrix sol. in 9 GMRES cycles |
| 12 | 6.63e-6 | 1.0 | matrix sol. in 10 GMRES cycles |
| 13 | 5.77e-6 | 1.0 | matrix sol. in 10 GMRES cycles |
| 14 | 5.17e-6 | 1.0 | matrix sol. in 10 GMRES cycles |
| 15 | 4.72e-6 | 1.0 | matrix sol. in 10 GMRES cycles |
| | | | upstream $P_T$ eqn residual = 0.205e-3 near $M_{\text{rel}} = 1$. |

machine zero as shown in Table 7.7. The total CPU time for the first 15 iterations is 569 CRAY seconds.

The contours of $\psi_2$ (i.e. streamlines) are shown in Fig. 7.26 and the mid-passage Mach contours are in Fig. 7.27. For comparison, the suction surface Mach contours presented in reference [49] are shown in Fig. 7.28. These were calculated using an Euler solver with boundary layer coupling. The inlet region is quite similar, as it should be since the cases were run at the same flow condition. In the blade region, the plots should not be similar because one plot is at mid-passage and the other is at the suction surface. At the exit, Mach numbers predicted using the current solver are lower than those in Fig. 7.28. This is because the boundary layer coupling adds blockage effects which are not present with the two stream function solver. There is a passage shock predicted and the contours of the entropy variable ($\hat{s} = \exp(-s/R)$) show the entropy rise across the shock. There is work added by the rotor, so the total pressure cannot be

Table 7.6: Continuation of the convergence history from Table 7.5 with different upstream boundary conditions.

Relative flow angle boundary condition applied using angles from iteration 15. Compressible flow, $S$ momentum equation applied, ni=2, nj=21, nk=54, GMRES max cycles = 10, No line search algorithm used, $M_c^2 = 0.8$, leading edge movement constrained.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 15 | 4.65e-6 | - | |
| 16 | 1.27e-4 | 1.0 | matrix factored |
| 17 | 1.49e-6 | 1.0 | mat. sol. in 9 GMRES cycles, max $P_T$ err = .46e-5 |
| 18 | 1.43e-6 | 1.0 | mat. sol. in 7 GMRES cycles, max $P_T$ err = .17e-9 |
| 19 | 1.39e-6 | 1.0 | mat. sol. in 8 GMRES cycles, max $P_T$ err = .44e-12 |
| 20 | 1.35e-6 | 1.0 | mat. sol. in 8 GMRES cycles, max $P_T$ err = .41e-12 |
| 21 | 1.31e-6 | 1.0 | mat. sol. in 8 GMRES cycles, max $P_T$ err = .17e-12 |
| | | | max residual in momentum eqs near leading edge mid-span |

Table 7.7: Continuation of the convergence history from Table 7.6 after converting to duct flow.

Periodic boundaries are converted to walls to create a duct. Compressible flow, $S$ momentum equation applied, ni=2, nj=21, nk=54, GMRES max cycles = 10, line search algorithm used, $M_c^2 = 0.8$, leading edge movement constrained.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 21 | 1.31e-6 | - | |
| 22 | 1.62e-7 | 1.0 | matrix factored |
| 23 | 3.62e-11 | 1.0 | mat. sol. in 7 GMRES cycles |
| 24 | 3.04e-15 | 1.0 | mat. sol. in 5 GMRES cycles |
| 25 | 2.88e-15 | 1.0 | mat. sol. in 3 GMRES cycles |
| 26 | 2.84e-15 | 1.0 | mat. sol. in 3 GMRES cycles |
| 27 | 2.84e-15 | 0.0 | mat. sol. in 3 GMRES cycles |
| | | | line search could not decrease residual norm |

used as a measure of shock loss.

In Fig. 7.30, the static pressure rise ($P_s|_{\text{exit}}/P_T|_{\text{inlet}}$) against mass flow [1] are compared to the experimental results and the Euler solutions [49] run with and without boundary layers. The results using the two stream function solver are close to the Euler results without boundary layers at the hub and further off at the tip. This plot shows the effect the boundary layer blockage can have on a solution in this Mach number regime. The tip is further off due to the sensitivity of the flow in this transonic regime, and because without the blade-to-blade resolution, the precise shock structure and associated entropy rise cannot be captured.

The spanwise distribution of work added is related to the absolute tangential velocity through the Euler turbine equation [39]:

$$\Delta H = \omega \Delta (r V_\theta). \tag{7.11}$$

Since $V_\theta = 0$, at the inlet of this rotor

$$\Delta H = \omega r V_\theta|_{\text{exit}}. \tag{7.12}$$

A normalized value of the predicted exit $V_\theta$ is plotted in Fig. 7.31 along with the experimental results and the Euler solution results [49] with and without boundary layers. The prediction is better than either Euler solution near the hub, but deviates further at the tip. At the tip, the results are high, which is the trend if the boundary layer blockage is not accounted for. A minimum $V_\theta$ at 20% span is predicted and is also shown in the experimental results.

The cross stream grid lines are plotted in Fig. 7.32 which shows how the stream surfaces warp upstream of the leading edge and downstream of the trailing edge.

[1]Only one flow rate was analyzed, so the $\dot{m}/\dot{m}_{\text{max}}$ value was chosen to be an average of the other values presented.

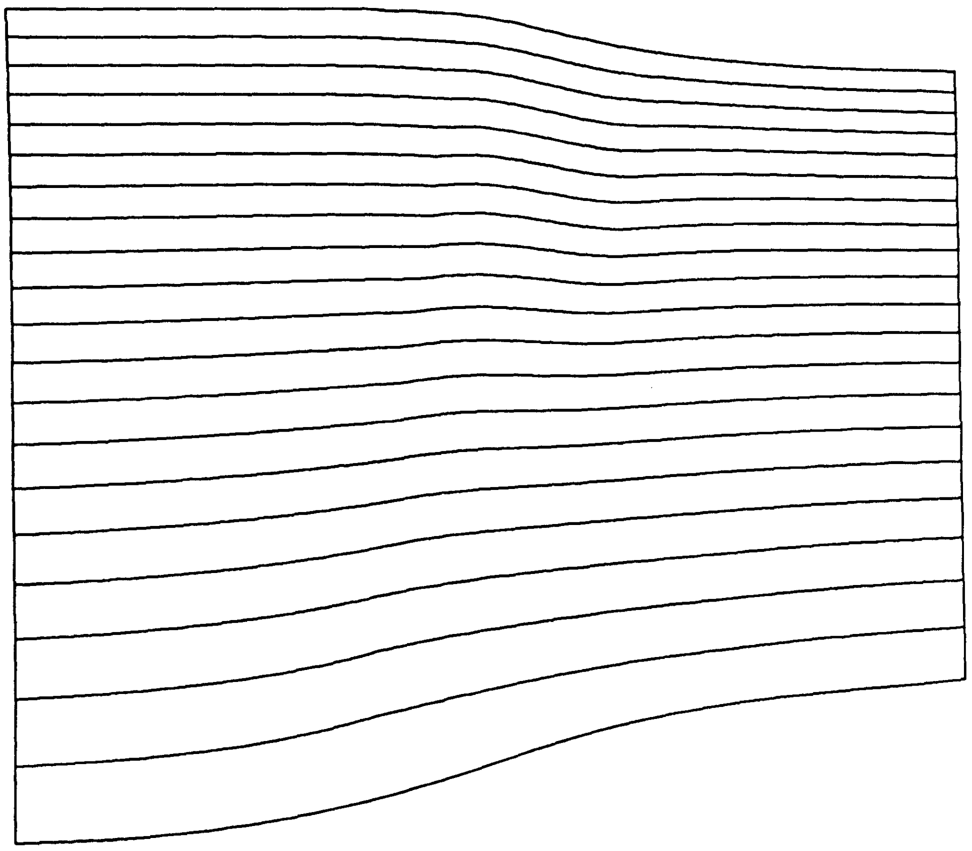


Figure 7.26: Contours of $\psi_2$ (streamlines) for the NASA 67 rotor.

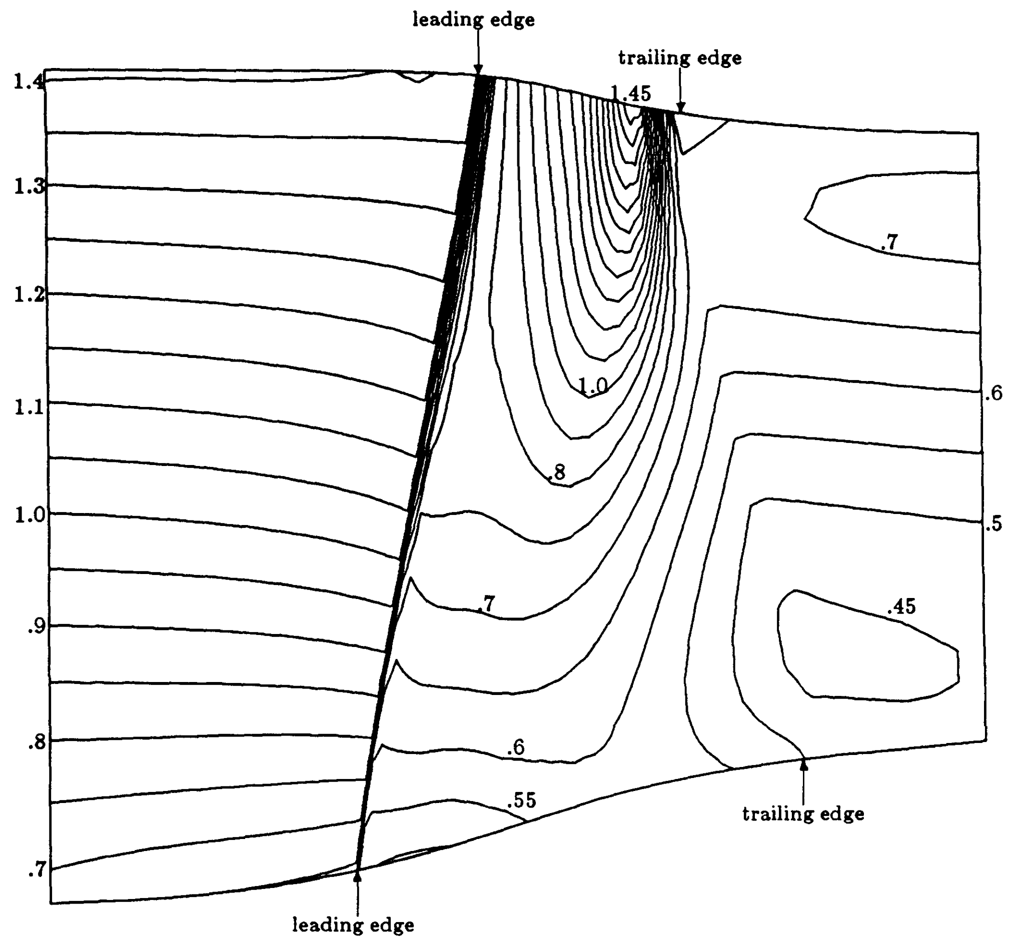


Figure 7.27: Contours of mid-passage Mach numbers for the NASA 67 rotor. Contour interval = 0.05.

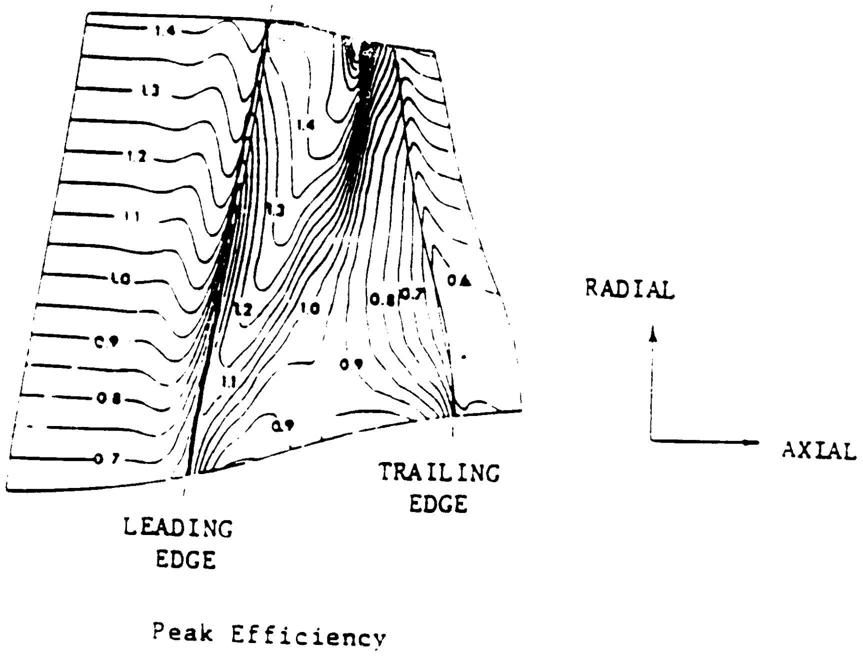


Figure 7.28: Contours of suction surface Mach numbers for the NASA 67 rotor. Calculated by an Euler solver with boundary layer coupling. From Fig. 11 in reference [49]
.

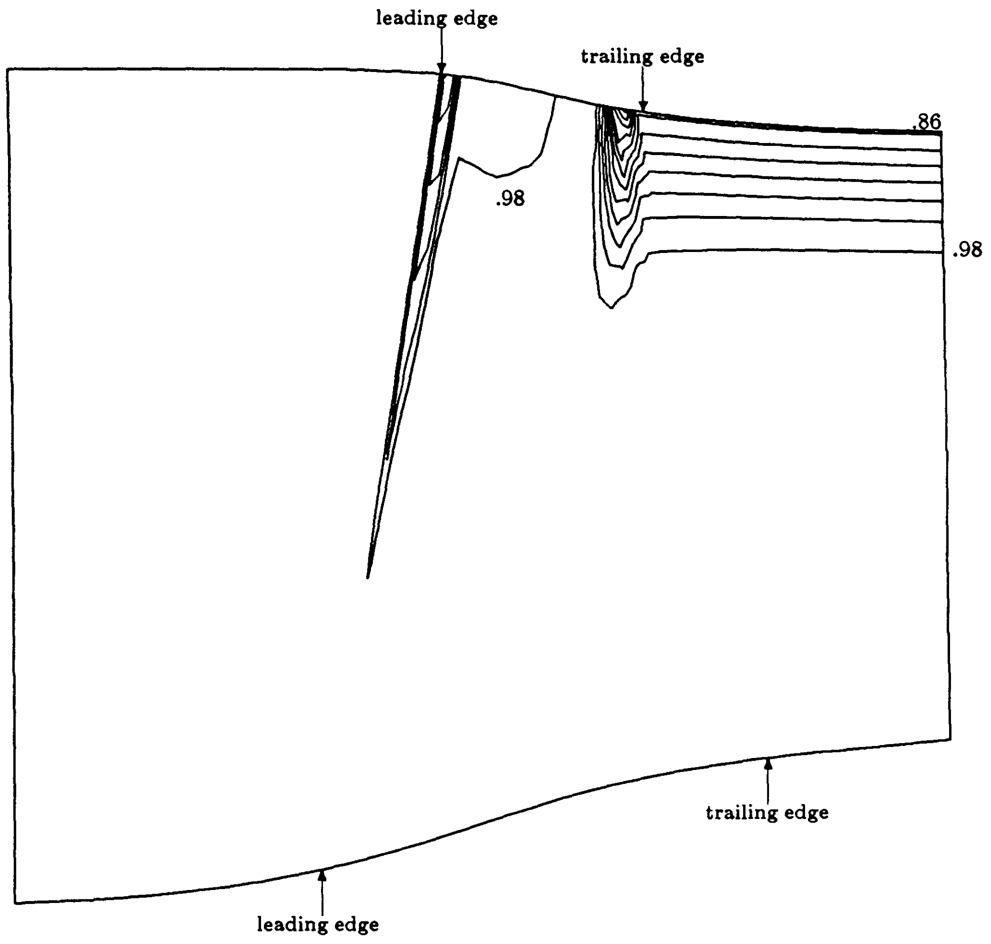


Figure 7.29: Contours of mid-passage entropy variable ($\hat{s} = \exp(-s/R)$) for the NASA 67 rotor. Contour interval=0.02.

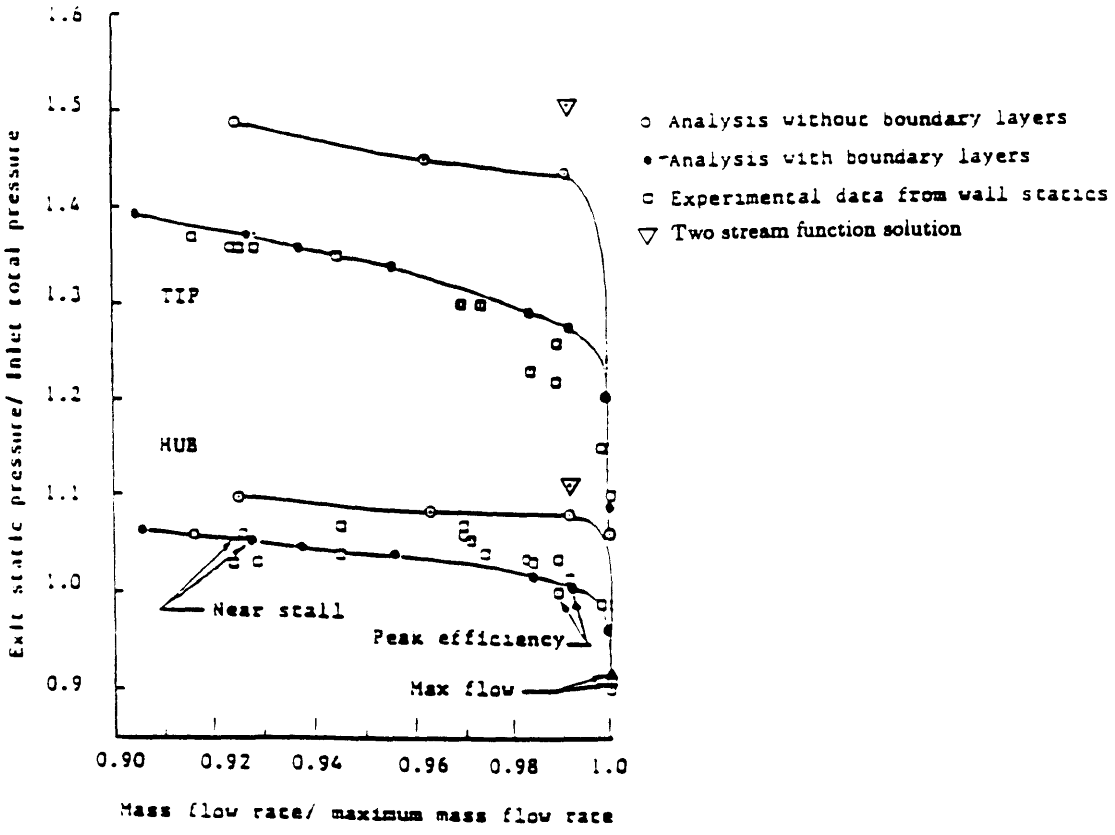


Figure 7.30: Static pressure ratio across the NASA 67 rotor.

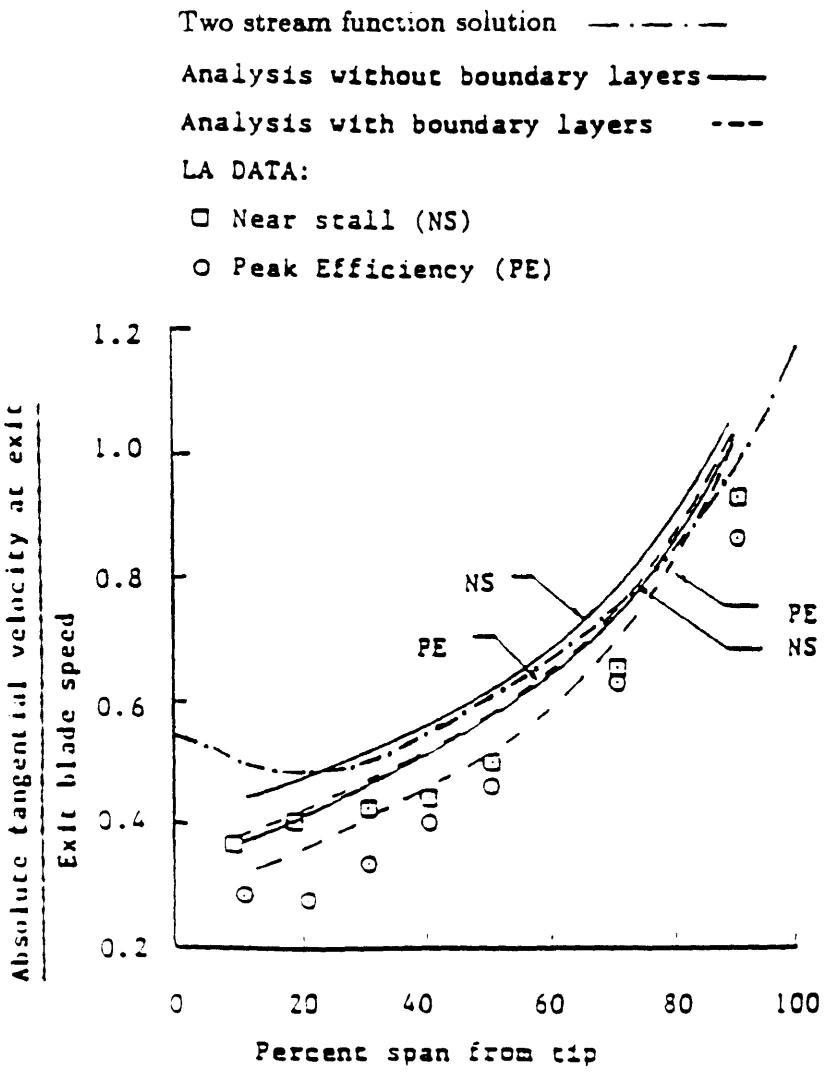
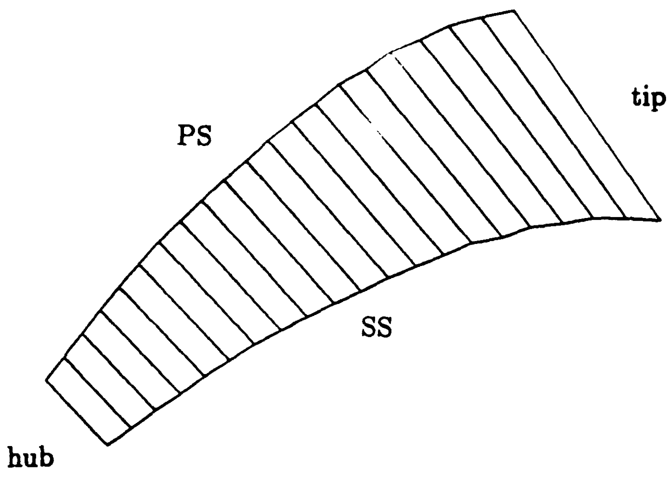
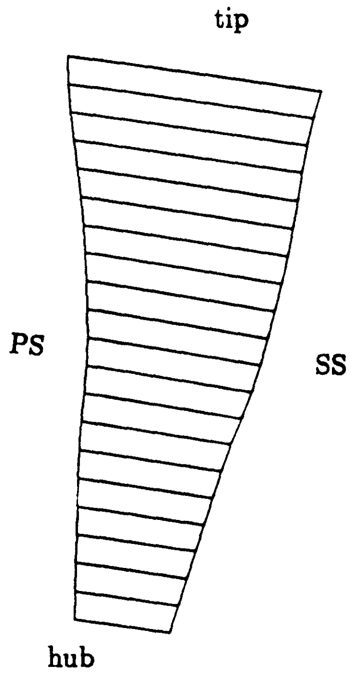


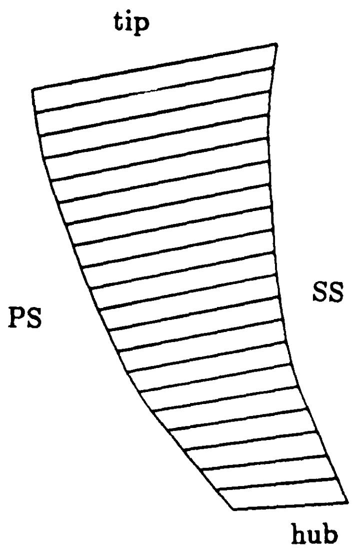
Figure 7.31: Spanwise variation of absolute tangential velocity at the exit of the NASA 67 rotor.



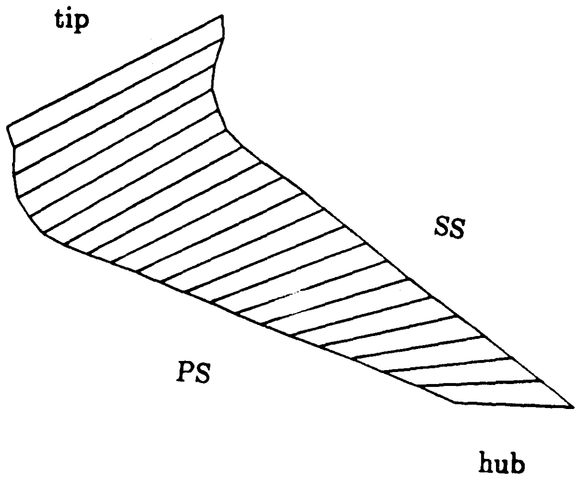
a) Inlet, K=1.



b) Leading edge, K=22.



c) Trailing edge. K=43.



d) Exit. K = 54.

Figure 7.32: Cross stream grid lines at four axial locations for the NASA 67 rotor showing the stagnation stream surfaces. Aft looking forward, PS and SS are the pressure and suction side respectively.

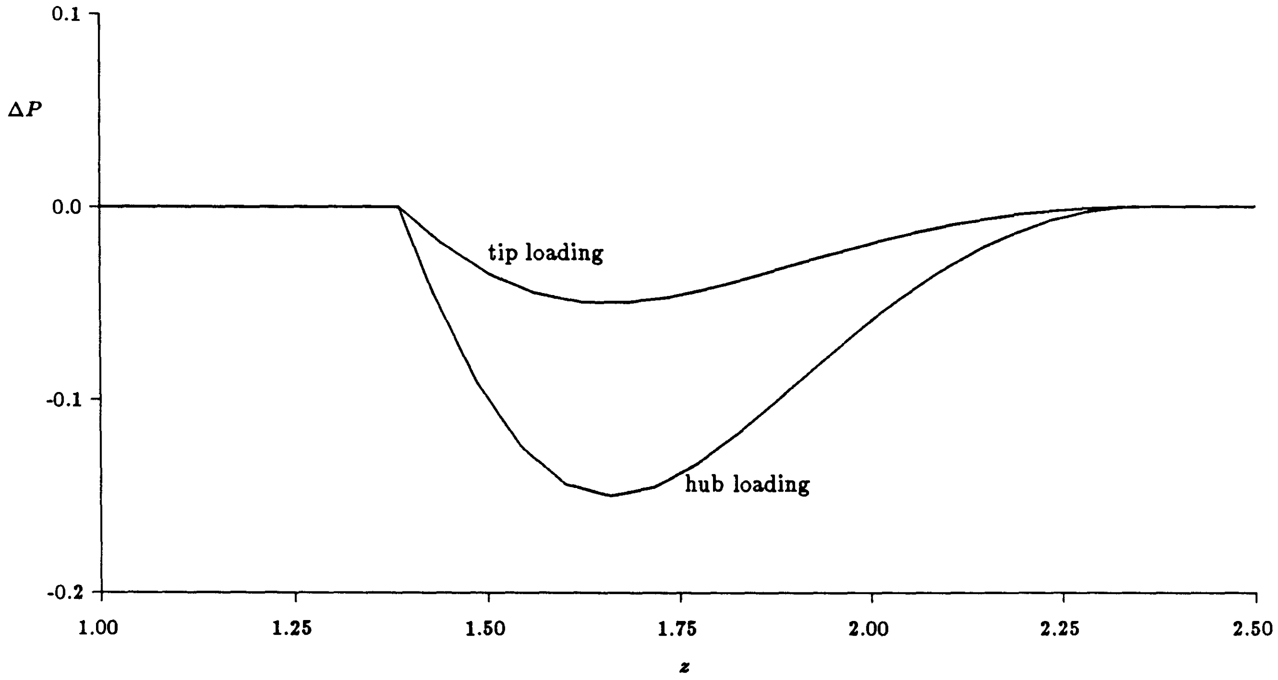


Figure 7.33: Pressure loading distribution imposed at hub and tip for the 3D design option case.

## 7.4 3D Inverse Design

To demonstrate the design option capability in 3D, an arbitrary pressure distribution has been imposed using the E$^3$ flow path and blade thickness distribution. The pressure loading was specified as

$$\Delta P = \Delta P_{\text{max}} \sin(\pi \; (1 - ch)^2), \tag{7.13}$$

where $ch$ is the percent chord and $\Delta P_{\text{max}}$ varied linearly from $-0.15P_{\text{ref}}$ at the hub to $-0.05P_{\text{ref}}$ at the tip. The loading applied at the hub and tip are shown in Fig. 7.33. The total pressure, $P_T = 1.065P_{\text{ref}}$. The mass flow, total enthalpy and $P_{\text{ref}}$ are the same as for the E$^3$ case with a shear layer.

Figure 7.34 shows the grid in the $r$-$z$ plane, and the initial grid (including the initial blade shape) at hub, pitch, and tip are shown in Fig. 7.35. These plots are in $m'$-$\theta$ coordinates. The axial projections vary from hub to tip because $\Delta\theta$ is the same.

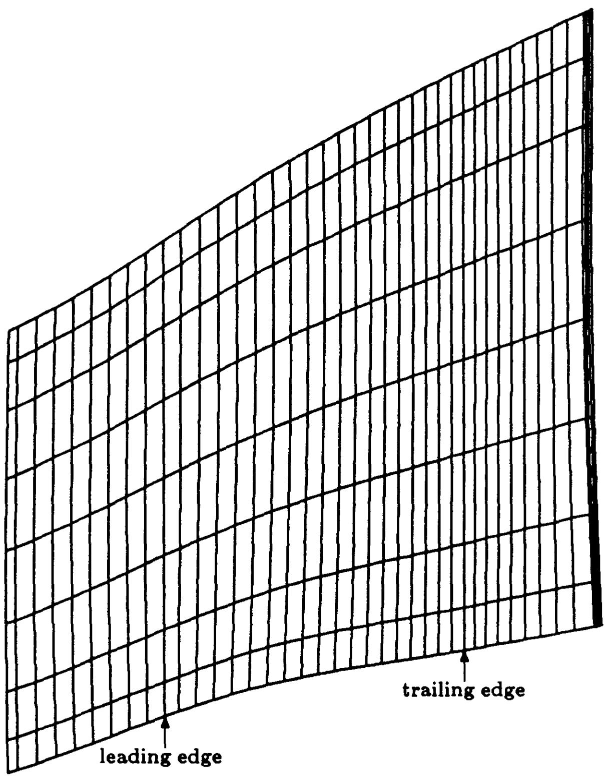


Figure 7.34: Grid in the *r*-*z* plane used to analyze the 3D design option case. ni=2, nj=9, nk=41.

a) Tip, J=9.

b) Pitch, J=5.

c) Hub, J=1.

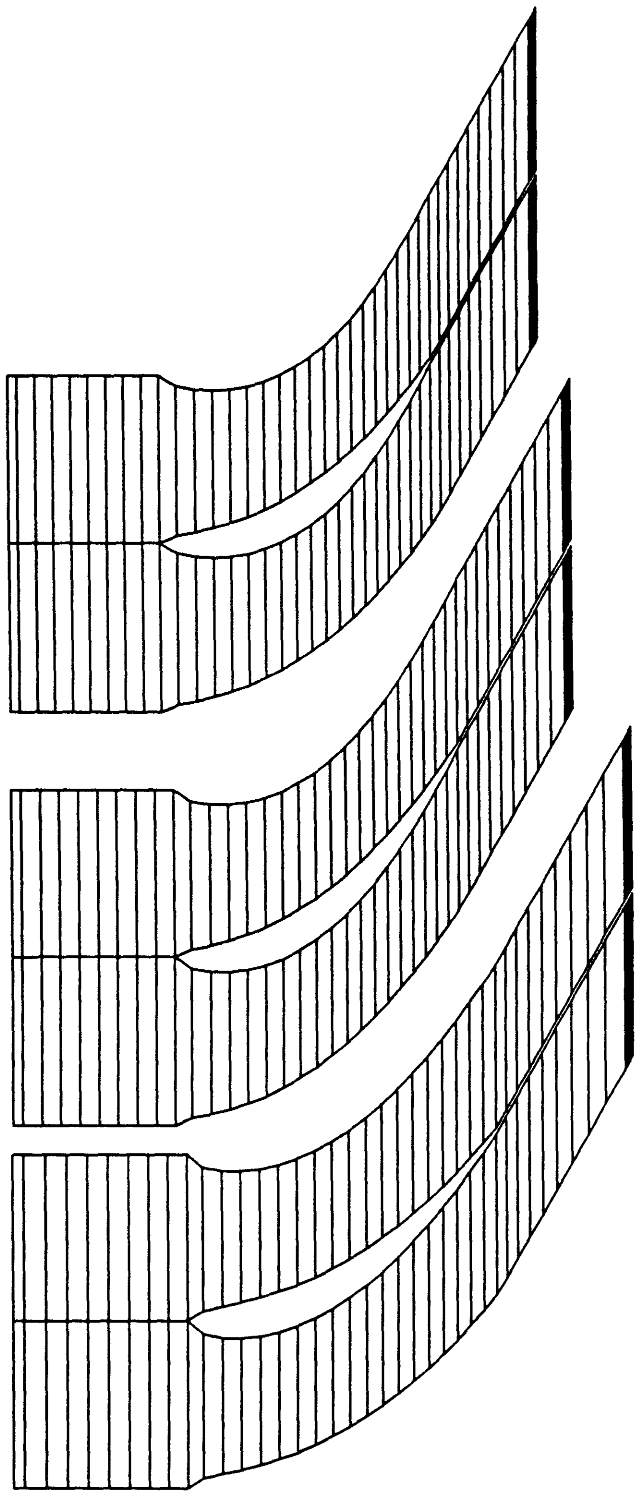


Figure 7.35: Initial grid and blade shape at the hub, pitch and tip in the $m'$-$\theta$ plane for the 3D design option case. ni=2, nj=9, nk=41.

Table 7.8: Convergence history for the 3D design option case.

The $E^3$ flow path and blade thicknesses were used with a uniform inlet total pressure. Compressible flow, entropy convection equation applied, ni=2, nj=9, nk=41, GMRES max cycles = 10, line search algorithm used.

| iter | error | damping $\lambda$ | comments |
|---|---|---|---|
| 0 | 1.41e-2 | - | |
| 1 | 4.07e-4 | 1.0 | matrix factored |
| 2 | 1.92e-5 | 1.0 | matrix factored |
| 3 | 5.32e-6 | 1.0 | matrix factored |
| 4 | 2.70e-6 | 1.0 | matrix factored |
| 5 | 1.75e-6 | 1.0 | matrix sol. in 9 GMRES cycles |
| 6 | 1.32e-6 | 1.0 | matrix sol. in 10 GMRES cycles |
| 7 | 1.05e-6 | 1.0 | matrix factored |
| 8 | 8.75e-7 | 1.0 | matrix sol. in 6 GMRES cycles |
| 9 | 7.36e-7 | 1.0 | matrix sol. in 7 GMRES cycles |
| 10 | 6.27e-7 | 1.0 | matrix sol. in 7 GMRES cycles |
| 11 | 5.42e-7 | 1.0 | matrix sol. in 8 GMRES cycles |
| 12 | 4.75e-7 | 1.0 | matrix sol. in 8 GMRES cycles |
| | | | engineering tolerance achieved |

The convergence history is shown in Table 7.8. The solution did not converge to machine zero, but did converge below an engineering tolerance by the twelth iteration. The total CPU time was 109.9 CRAY seconds.

The resulting blade shapes are shown in the grid plots in Fig. 7.36 for the hub, pitch, and tip sections. As expected, there is less turning at the tip where the loading is less. Mach number contours at mid-passage are shown in Fig. 7.37, and Fig. 7.38 is a plot of the Mach number distribution at hub, pitch and approximate pitch (these are plotted midway between grid lines).

The exit angle distribution is plotted in Fig. 7.39, again showing the variation of turning is consistent with the prescribed loading. The cross stream grid lines at several locations showing the stagnation stream surfaces are plotted in Fig. 7.40.

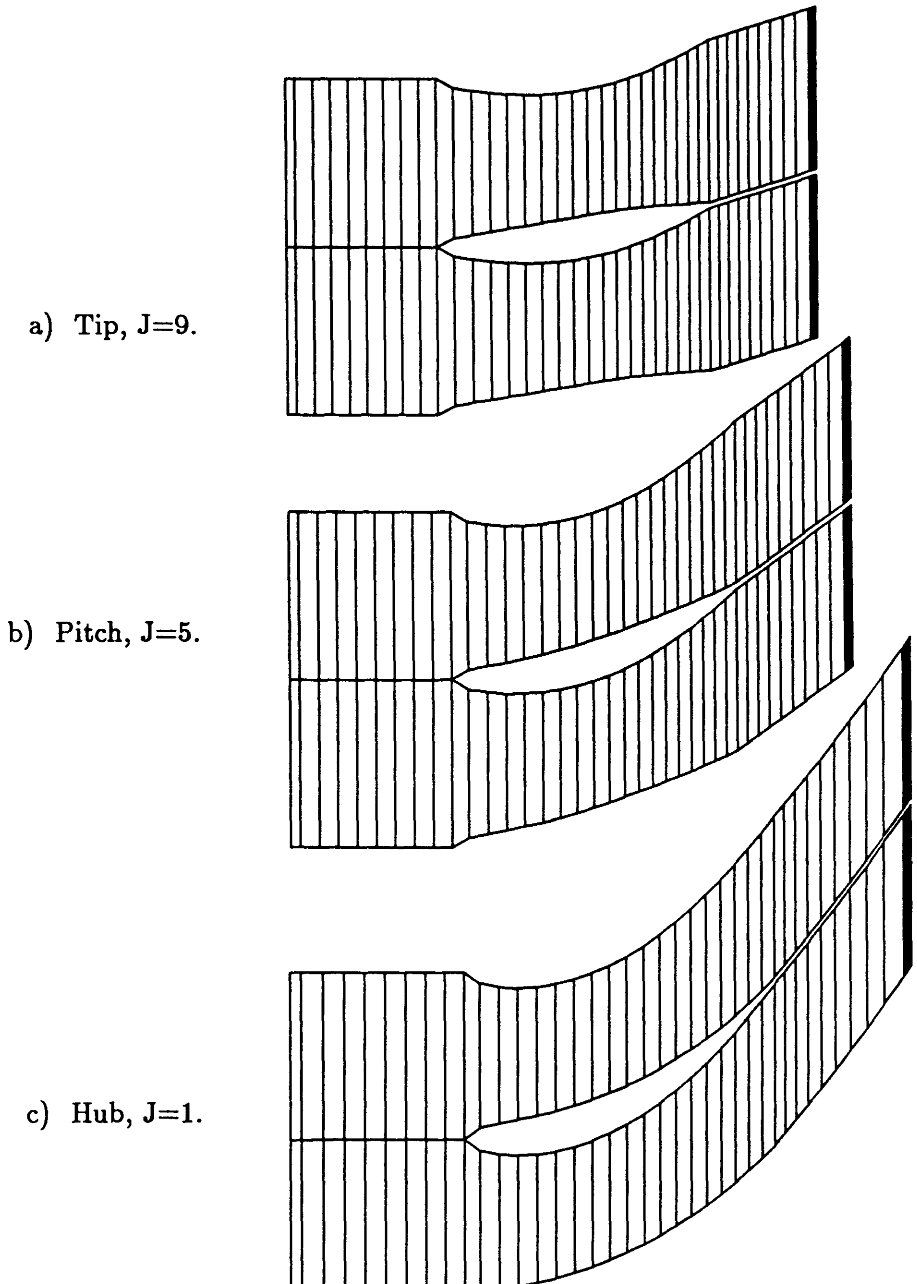


Figure 7.36: Converged grid and blade shape at the hub, pitch and tip in the $m'$-$\theta$ plane for the 3D design option case. ni=2, nj=9, nk=41.

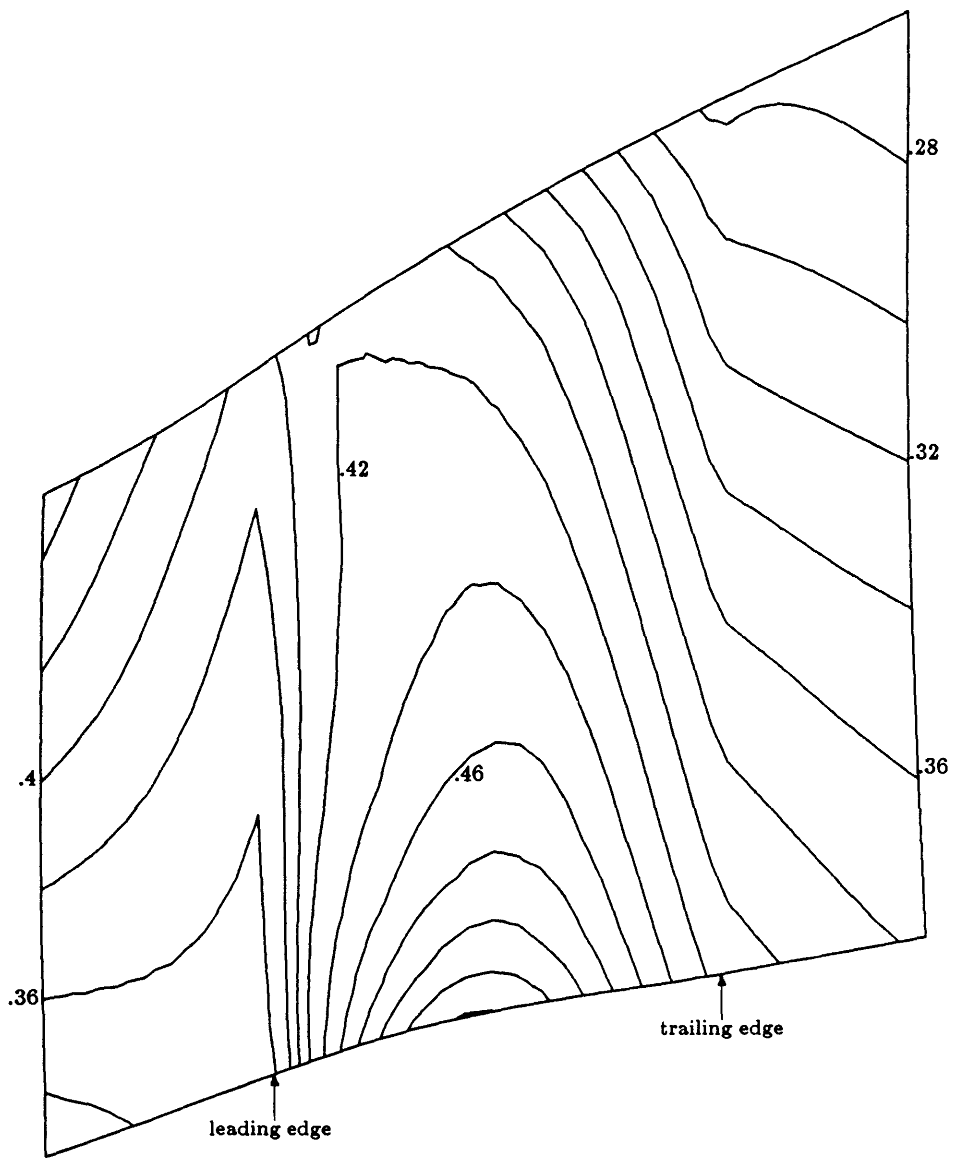


Figure 7.37: Mach contours in the *r*-*z* plane for the 3D design option case. Contour interval = 0.02.

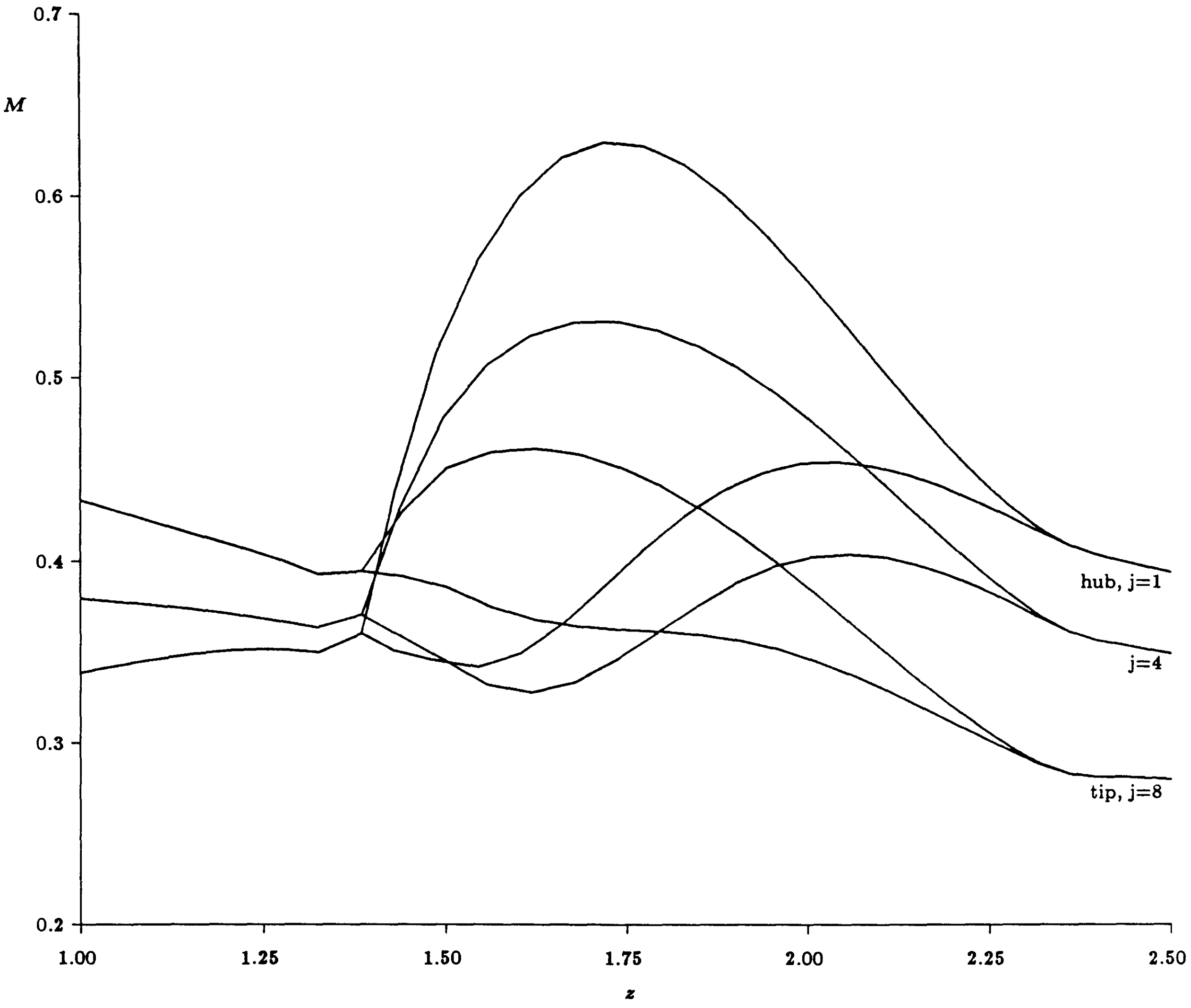


Figure 7.38: Mach number distributions at hub, approximate pitch and tip sections for the 3D design option case.

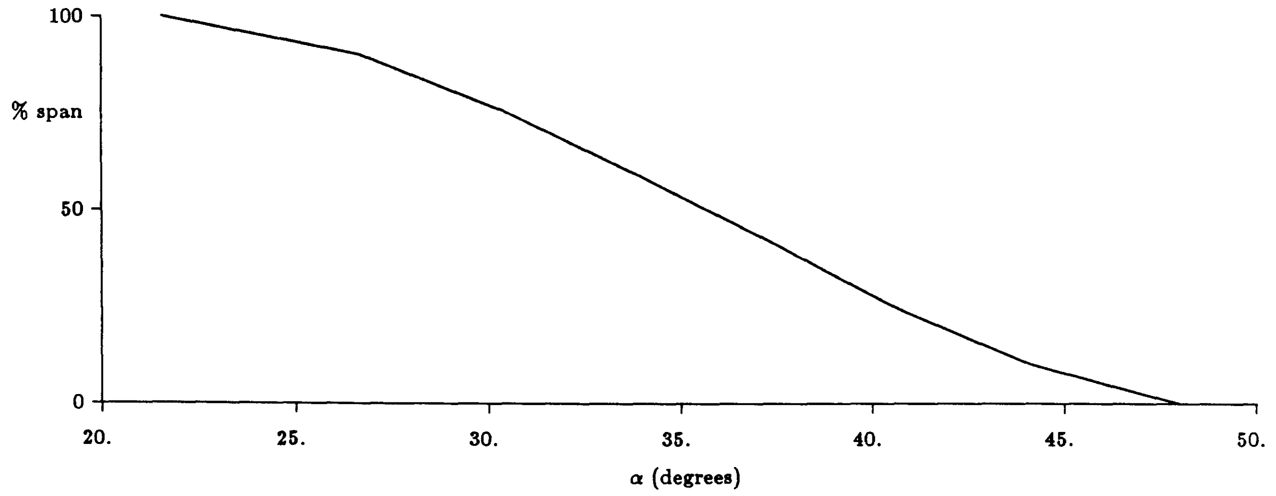


Figure 7.39: Exit angle distribution for the 3D design option case. Hub is at 0% span.

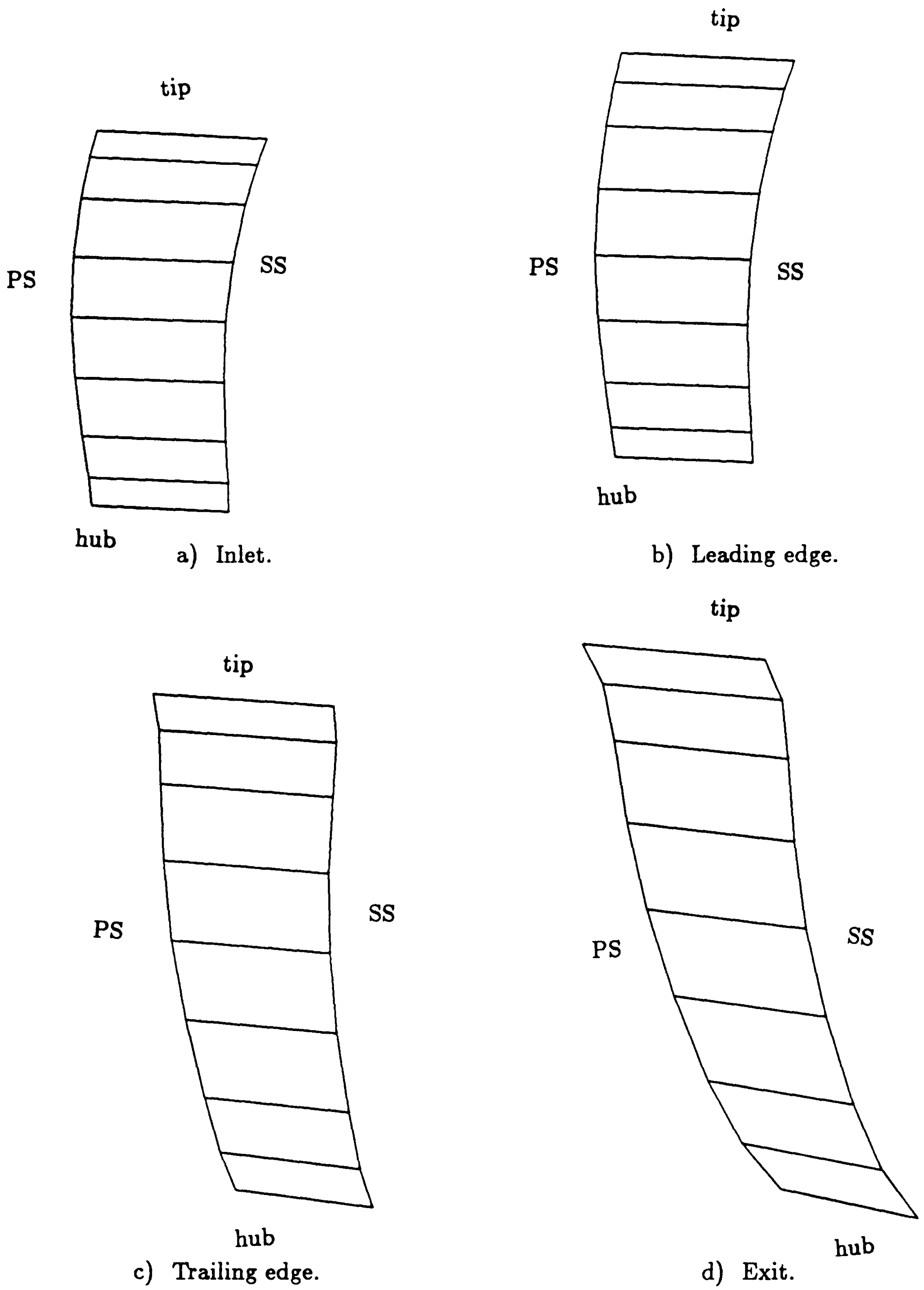


Figure 7.40: Cross stream grids showing the stagnation stream surfaces for the 3D design option case. Aft looking forward, PS and SS are the pressure and suction side respectively.

# Chapter 8
# Concluding Remarks

## 8.1 Summary

A 3D inviscid flow solver has been developed, using two stream functions, primarily for turbomachinery applications. It is an extension of the ISES algorithm (developed by Giles and Drela [25] [27] [16]) to fixed grids[1] and three dimensions. Contours of constant stream function represent stream surfaces so this technique can be interpreted as solving for the classical $S_1$, $S_2$ stream surfaces of Wu [66]. The conservative finite volume momentum equations are solved so the correct Rankine-Hugoniot shock jump conditions are satisfied in the limit of infinite grid resolution.

To solve for transonic flowfields, some artificial dissipation is required to capture shocks and ensure the well-posedness of the problem in the supersonic region. No artificial dissipation is required for subsonic problems. An upwinded pressure approach is used to provide the artificial dissipation in the supersonic region. Giles [27] used an artificial compressibility approach which effectively is a density upwinding scheme. The use of pressure allows the same code to be used for both compressible and incompressible flows by using the equation of state or treating density as a constant.

Newton's method has been used to solve the system of discrete non-linear equations. Newton's method updates the solution assuming the system is locally linear. The linearization of the system was performed using SMP (Symbolic Manipulation Program) which also wrote the FORTRAN code. This process is tedious and prone to mistakes if done manually. When performed automatically, changes are easily made and the

[1]The grid is fixed for a duct or in the blade region. However grid movement is allowed to determine the stagnation stream surfaces.

chance of errors is greatly reduced. The convergence rate of the solver demonstrated the correctness of the linearization.

The solution of a large matrix equation is required during each Newton iteration. A method has been developed which can use a matrix factor from a prior Newton iteration as a preconditioner for a conjugate gradient-like iterative procedure called GMRES [55]. The matrix solution is obtained (not an approximation) which preserves the quadratic convergence rate of Newton's method. Using this method saves CPU time since it reduces the number of matrix factorizations required.

As discussed in references [27] and [63], Newton's method can be very efficient when applied to 2D problems. The same conclusion has been reached in this thesis. The true 3D problems solved using the two stream function solver were limited to coarse grids due to the large amount of storage required for the matrix factorization. This coarse grid was adequate for the problem analyzed, but storage and CPU time are still a limitation for solving fine grid 3D problems using Newton's method.

The 3D solver reduces to 2D by using two identical grids displaced by a constant distance. The stream function $\psi_2$ is then specified to be a constant on both surfaces (i.e., 0 on the lower wall and 1 on the upper wall). This approach is essentially a fixed grid analog of ISES and $\psi_1$ becomes the classic 2D stream function $\psi$.

An order of accuracy study was performed using a smooth $\sin^2(\pi z)$ bump duct. The discretization has been shown to be second-order accurate. Comparisons of several matrix solution techniques were also made. Several other 2D cascade flows were calculated and compared to ISES, experimental results and analytical results. These include the incompressible Gostelow cascade, a high turning subsonic turbine cascade with a -71.5 degree exit angle which demonstrates the effect of grid shear on the solver, and a supercritical cascade designed by Garabedian. There is good agreement between the current method and the other data for each case. This validated the 2D applications of the 3D solver.

Results have been obtained for the subsonic GE $E^3$ low pressure turbine nozzle and the transonic NASA 67 fan. Comparisons have been made between the current method and experimental results of the exit flow angle for the $E^3$ nozzle with an incoming shear layer and the exit tangential velocity distribution for the NASA 67 fan. These comparisons showed reasonable agreement. This exit spanwise angular momentum distribution is the desired result of an inviscid analysis. These cases were run with only one volume in the blade-to-blade direction and therefore an axisymmetric stream surface is implied. The secondary flows for these cases are strong which leads to warping of the stream surfaces. This warping cannot be handled by the solver with more blade-to-blade resolution.

The $E^3$ nozzle was also run without an incoming shear layer. Runs were presented with 1 and 5 volumes in the tangential direction. This compared solutions from the solver run in the "axisymmetric" mode and in the full 3D mode. In the full 3D mode, the stagnation stream surface was solved for and treated as a stream sheet. The radial velocities were therefore not treated as periodic across the wake as they would be for many Euler solvers.

The 2D design option capability was demonstrated by producing a blade with a NACA 0012 thickness distribution and an arbitrary pressure loading and inlet angle. It was also shown that the overall tangential momentum equation was satisfied (i.e., the overall change in angular momentum, $rV_\theta$, is equal to the integrated blade loading).

The inverse design capability was also demonstrated in 3D. A 3D blade was obtained given the flow path and thickness distribution of the $E^3$ nozzle along with a prescribed spanwise and streamwise pressure loading distribution. The equations used in the design mode are the same as those used in the analysis mode so the resulting blade shape would produce the specified loading assuming inviscid flow.

## 8.2 Comparison to Through-Flow Calculation Methods

The results with only one volume blade-to-blade basically assume the stream function is axisymmetric and the pressure is linear across the blade passage. Outside of the blade region, the pressure is also axisymmetric. Run in this mode, the solver has many of the properties of a through-flow code which assumes all quantities are axisymmetric. These codes are described by Oates [47] and briefly described in the discussion which follows. Also included in this discussion is how the two stream function solver compares to these through-flow methods.

The current through-flow methods can usually be categorized as one of the following three approaches: streamline curvature, finite difference and finite element. The streamline curvature methods (see Novak [45]) use an intrinsic grid so the grid lines become streamlines. The streamline curvature is approximated from the grid point distribution and is expressed directly in a form of the momentum equation normal to the streamlines. The finite difference methods use a stream function equation and apply finite difference stencils to approximate the stream function derivatives. Both Galerkin and variational finite element methods have been applied to solve the through-flow equations.

In most of these through-flow approaches, the equations are uncoupled so that first the grid or stream function is determined given the density field (the velocity and pressure are related to the density though the rothalpy and equation of state) and then the density field is updated assuming the streamlines or stream function values are fixed. This uncoupling can cause problems in convergence especially for transonic solutions. The current two stream function solver solves the entire coupled system of equations and is, therefore, more robust for transonic applications.

In each of the existing through-flow approaches the streamwise momentum equation is replaced by an entropy transport equation in which loss models are used to determine the entropy rise in a given region. In the two stream function approach, the entropy rise across a shock can be determined by using the streamwise momentum equation. An entropy transport equation can also be applied, but no loss models have been used yet.

The tangential momentum equation relates the blade force in the tangential direction to the change in tangential momentum. To analyze a blade, the distribution of angular momentum is determined from a series of 2D blade-to-blade solutions with stream tube height and radius variations. Different angular momentum distributions will produce 2D blade sections with different radii and stream tube heights. The full quasi-3D solution is therefore obtained by iterating between the through-flow calculation and the blade-to-blade solutions. In the current two stream function approach, no iteration is required. The pressure distribution is treated to be essentially linear blade-to-blade, with a finite volume form of the tangential momentum equation used.

To account for the blade forces not in the tangential direction, the lean distribution of a blade must be specified to the through-flow solver. Most through-flow solvers assume the lean is the same on both sides of the blade and do not account for the lean due the thickness variation hub to tip. This lean force is also generally lagged an iteration which slows convergence. In the current two stream function approach, the full blade shape is either specified or determined in the design option mode so the lean effects are accounted for by using the appropriate area projection of the blade surfaces.

The through-flow solvers can handle multiple blade rows. For the current two stream function solver, a boundary would be required to go between a rotor reference frame and a stator reference frame in order to handle multiple blade rows.

The two stream function solver is essentially a sophisticated through-flow solver for one blade row when run with only one control volume blade-to-blade. The linear pressure variation blade-to-blade seems to be a crude approximation, but in fact works quite well. The approximation is worse near leading edges and is less applicable as the Mach number approaches unity.

## 8.3 Conclusions

This thesis has described a new numerical internal flowfield solver which has been developed using a two stream function formulation. These are the major conclusions of of this research.

1. Quadratic convergence rates have been demonstrated using Newton's method to solve the system of discrete equations.
   - The linearized equations have been derived using SMP (Symbolic Manipulation Program) which can determine the analytic derivatives of an equation and write the FORTRAN code. This eliminated many of the mistakes commonly associated with developing a solver using Newton's method.
   - Sparse matrix methods have been explored. The application of these methods in 2D is very effective. However, in 3D the CPU time and memory requirements are too large to be practical for fine grid solutions.
   - A new method has been developed which solves a matrix equation iteratively using a matrix factor from a previous Newton iteration. A Generalized Minimum Residual (GMRES) algorithm is then used to accelerate and stabilize the method. Savings in CPU time are realized because fewer matrix factorizations are needed.
2. The solver has been validated for solving 2D transonic cascade flows. Several 2D calculations have been made using the solver and the solutions compare well with theoretical, experimental and other numerical results. Second-order accuracy of the method has also been demonstrated.
3. A 3D flowfield solution has been obtained using the solver, and it is believed to be the first time a 3D numerical solution has been obtained using a two stream function approach.
   - The method can handle only a small amount of stream surface warping. The method, therefore, is restricted to flows with little or no streamwise vorticity if run in a full 3D mode.

- To overcome the stream surface warping limitation, several 3D applications have been run with only one control volume in the blade-to-blade direction. This implies an axisymmetric stream surface and is essentially a through-flow solver. There are, however, several advantages of this method over existing through-flow solvers.

4. A design option capability has been demonstrated in 2D and in 3D using one control volume blade-to-blade. A blade camber line is produced given the blade thickness and loading distribution. It is believed to be the first time a 3D design option capability has been demonstrated which is not limited to a free-vortex design.

## 8.4 Extensions and Future Work

Several direct extensions of this thesis are:

- To develop a true axisymmetric solver with a coupled blade-to-blade solver. The integrated static pressure field in $\theta$ would be applied to determine the $\psi_2$ distribution. The result would be a directly coupled version of existing quasi-3D design systems which iterate between the axisymmetric and blade-to-blade solutions. A plane-by-plane iterative matrix solver should be more robust for this type of application than it was found to be for the full 3D problem. Also, multiple blade row capability could be handled by adding boundaries which separate the rotor and stator reference frames.

- To add interface boundaries. The limitations of the current method are due to the stream surface warping. This causes the two stream function gradients to become large which requires a fine grid to resolve. The equations become more nonlinear, and the matrix becomes ill-conditioned when the finite differencing is no longer done in a "preferred direction." Because of the arbitrariness of the stream surfaces, interface boundaries could be applied where the stream surfaces are untwisted and start over. It would be similar to applying the upstream and downstream boundary condition on both sides of the interface.

- To make the matrix solution more efficient by correctly partitioning the matrix into two parts: its symmetric structure part and its lower triangular part due to the elliptic and convective operators applied. The sparse matrix methods could then be applied more effectively to the symmetric part.
- To extend the solver boundary conditions to handle choked flows.
- To use the code for a rotor application with an upstream rothalpy distribution.

Some less direct extensions will be discussed in the following sections.

### 8.4.1 2D Viscous Algorithm

The most promising extension of the current research is for solving 2D viscous flows. This could be for steady compressible or steady and unsteady incompressible flows. By working with the stream function on a fixed grid rather than an intrinsic grid, separated flows could be easily handled.

The shear stress would be needed in the momentum equation and could be evaluated from appropriate discrete forms of the velocity derivatives. The energy equation would have to be added with an energy variable (i.e., temperature, rothalpy, total enthalpy, etc.) stored at the center of an $S$ face. At the center of an $A$ face, the energy could be interpolated from its neighbors. The finite volume equations and upstream boundary conditions would be enough equations to satisfy the added unknowns. Other convection-diffusion equations such as a $k$-$\epsilon$ turbulence model could be treated similarly.

The efficiency of Newton's method and a direct solver in 2D has been demonstrated. The added stiffness of the Navier-Stokes equations can be handled best using an implicit procedure such as this rather than an iterative or time marching approach. Due to the limitations of the full 3D inviscid solver (i.e., stream surface warping and matrix size), a 3D viscous solver would be even more limiting. Therefore, a viscous algorithm would not be effective if extended to 3D. However, there are many applications where a robust 2D viscous algorithm would be useful. The 2D inviscid results have demonstrated the algorithm produces a very low level of numerical diffusion. Applied as a viscous solver, the low numerical diffusion would not mask the physical diffusion.

### 8.4.2 Use of Two Stream Functions in Postprocessing

Visualizing stream surfaces can be very instructive in understanding a complex 3D flowfield. A common approach to visualizing a stream sheet is to start a line of particles and calculate the trajectories. The resulting streamlines in steady flow are then presented graphically. These particle tracing calculations are very CPU intensive. Another way to accomplish this would be to draw contours of a stream function on 2D planes which go from upstream to downstream, or to draw the surface of a constant stream function which is a stream sheet.

The stream functions $\psi_1$ and $\psi_2$ can be calculated using any 3D algorithm which can solve the steady convection equation

$$\frac{D\psi}{Dt} = 0, \tag{8.1}$$

or

$$\overline{V} \cdot \nabla\psi = 0, \tag{8.2}$$

and given the steady quantities of density and velocity (these could be obtained from inviscid or viscous calculations, or experimental data). Many existing Euler algorithms solve the conservative form of these equations and are vectorizable. Solving this equation allows a whole family of stream surfaces to be calculated efficiently and viewed interactively because only contour plotting is required.

If $\psi_2$ is specified as a linear function of radius at the upstream boundary, then the convection equation would determine $\psi_2$ everywhere. Iso-surfaces of $\psi_2$ would be stream sheets. Iso-surfaces of $\psi_1$ in the blade-to-blade direction would be treated similarly, but would have to be a periodic function. The stream sheets could be started at any plane in the flowfield. Going upstream, the negative velocity components would be used and the convective operators applied.

### 8.4.3 Alternative Equation Linearization

The use of SMP allowed the current code to be developed with the correct analytic derivatives which make up the Jacobian matrix. For SMP to handle the complex equations, the chain rule was required. This was implemented through subroutine calls. The SMP generated code was not very efficient and the subroutine calls eliminated any vectorization available on the CRAY or other vector machine. The Jacobian calculation therefore became the most CPU intensive part of the solver. A more efficient way to develop a solver using Newton's method would be to calculate the elements of the Jacobian matrix numerically using finite differences. If appropriate step sizes are used, the quadratic convergence rate of Newton's method can still be achieved [15]. The possibility of mistakes associated with hand-derived derivatives would be minimized, and for many terms, the analytic derivatives may be more complex than calculating the derivatives numerically. If computational speed is an issue and the analytic derivatives are less costly to compute than the finite difference calculation, then the analytic expression can be compared with its finite difference counterpart to check for mistakes. Although the use of SMP did simplify the development of this solver as compared with hand-derived analytic derivatives, using finite differences to calculate the Jacobian matrix might have simplified development further.

# Bibliography

[1] Ira H. Abbott and Albert E. Von Doenhoff. *Theory of Wing Sections*. Dover Publications, 1959.

[2] G. G. Adkins, Jr. and L. H. Smith, Jr. Spanwise mixing in axial-flow turbomachines. *ASME Journal of Engineering for Power*, 104:97,100–110, January 1982.

[3] Dale A. Anderson, John C. Tannehill, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. Hemisphere Publishing Corporation, New York, 1984.

[4] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[5] Frances Bauer, Paul Garabedian, and David Korn. *Supercritical Wing Sections III*. Volume 150 of *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag, 1977.

[6] A. I. Borisenko and I. E. Tarapov. *Vector and Tensor Analysis with Applications*. Dover Publications, New York, revised English edition, 1968. Translated and Edited by Richard A. Silverman.

[7] M. J. Bridgeman, D. G. Cherry, and J. Pedersen. *NASA/GE Energy Efficient Engine Low Pressure Turbine Scaled Test Vehicle Performance Report*. Technical Report CR-168290, NASA, NASA-Lewis Research Center, Cleveland, Ohio, July 1983.

[8] Richard H. Burkhart and David P. Young. *Documentation for GMRES Acceleration and Optimization Codes*. Engineering and Scientific Services Technical Report ETA-TR-89R1, Boeing Computer Services, Engineering and Scientific Services Division, G-6500, M/S 7L-21, P.O. Box 24346, Seattle, Washington 98124, June 1988.

[9] Richard H. Burkhart and David P. Young. *GMRES Acceleration and Optimization Codes.* Engineering and Scientific Services Technical Report ETA-TR-88, Boeing Computer Services, Engineering and Scientific Services Division, G-6500, M/S 7L-21, P.O. Box 24346, Seattle, Washington 98124, May 1988.

[10] M. L. Celestina, R. A. Mulac, and J. J. Adamczyk. A numerical simulation of the inviscid flow through a counterrotating propeller. *Transactions of the ASME Journal of Turbomachinery*, 108:187–193, October 1986.

[11] D. G. Cherry. January 1990. Personal communication.

[12] Eleanor Chu, Alan George, Joseph Liu, and Esmond Ng. *SPARSPAK: Waterloo Sparse Matrix Package User's Guide for SPARSPAK-A.* Research Report CS-84-36, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, November 1984.

[13] Inference Corporation. *SMP Reference Manual.* Inference Corporation, Los Angeles, California, 1983.

[14] R. L. Davis, J. E. Carter, and M. Hafez. *Three-Dimensional Viscous Flow Solutions with a Vorticity-Stream Function Formulation.* AIAA-87-0601, 1987.

[15] J.E. Dennis, Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[16] Mark Drela. *Two-Dimensional Transonic Aerodynamic Design and Analysis Using the Euler Equations.* PhD thesis, MIT, December 1985. Also, MIT Gas Turbine & Plasma Dynamics Laboratory Report No. 187, February 1986.

[17] Robert P. Dring and Gordon C. Oates. *Through Flow Theory for Nonaxisymmetric Turbomachinery Flow. Part I – Formulation.* ASME 89-GT-304, June 1989.

[18] I. S. Duff. Direct methods for solving sparse systems of linear equations. *SIAM Journal on Scientific and Statistical Computing*, 5(3):605–619, September 1984.

[19] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Transactions on Mathematical Software*, 7(3):315–330, September 1981.

[20] I.S. Duff. Permutations for a zero-free diagonal. *ACM Transactions on Mathematical Software*, 7(3):387–390, September 1981.

[21] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Applications of an element model for Gaussian elimination. In J. R. Bunch and D. J. Rose, editors, *Sparse Matrix Computations*, pages 85–96, Academic Press, 1976.

[22] Salamon Eskinazi. *Vector Mechanics of Fluids and Magnetofluids*, page 221. Academic Press, New York, 1967.

[23] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[24] J. H. Giese. Stream functions for three-dimensional flows. *Journal of Mathematics and Physics*, 30(1):31–35, April 1951.

[25] M. Giles, M. Drela, and W.T. Thompkins Jr. *Newton Solution of Direct and Inverse Transonic Euler Equations.* AIAA-85-1530, 1985.

[26] Michael Giles. *Non-Reflecting Boundary Conditions for the Euler Equations.* CFDL Report TR-88-1, Computational Fluid Dynamics Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, February 1988.

[27] Michael B. Giles. *Newton Solution of Steady Two-Dimensional Transonic Flow.* PhD thesis, MIT, June 1985. Also, MIT Gas Turbine & Plasma Dynamics Laboratory Report No. 186, October 1985.

[28] J. P. Gostelow. *Cascade Aerodynamics.* Pergamon Press, 1984.

[29] E.M. Greitzer, R.W. Paterson, and C.S. Tan. An approximate substitution principle for viscous heat conducting flows. *Proceedings of the Royal Society of London*, A 401(1820):163–193, September 1985.

[30] C. Hah. *A Navier-Stokes Analysis of Three-Dimensional Turbulent Flows Inside Turbine Blade Rows at Design and Off-Design Conditions.* ASME 83-GT-40, June 1983.

[31] C. Hah and J. H. Leylek. *Numerical Solution of Three-Dimensional Turbulent Flows for Modern Gas Turbine Components.* ASME-87-GT-84, June 1987.

[32] A. Hamed. Internal three-dimensional viscous flow solution using the streamlike function. *Transactions of the ASME Journal of Fluids Engineering*, 108:348–353, September 1986.

[33] W. R. Hawthorne. *Secondary Vorticity in Stratified Compressible Fluids in Rotating Systems.* Technical Report CUED/A-Turbo/TR 63, University of Cambridge, Department of Engineering, 1974.

[34] D. G. Holmes and S. S. Tong. *A 3-D Euler Solver for Turbomachinery Blade Rows.* ASME 84-GT-79, 1984.

[35] M.L. James, G.M. Smith, and J.C. Wolford. *Applied Numerical Methods for Digital Computation with FORTRAN and CSMP.* IEP–A Dun-Donnelley Publisher, second edition, 1977.

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

[37] M. W. Johnson. *Secondary Flow in Rotating Bends.* Technical Report CUED/A-Turbo/TR 92, University of Cambridge, Department of Engineering, 1978.

[38] Krishnamurty Karamcheti. *Principals of Ideal-Fluid Aerodynamics.* John Wiley and Sons, 1966.

[39] Jack L. Kerrebrock. *Aircraft Engines and Gas Turbines.* The MIT Press, 1977.

[40] M. Koya and S. Kotake. *Numerical Analysis of Fully Three-Dimensional Periodic Flows Through a Turbine Stage.* ASME 85-GT-57, 1985.

[41] Erwin Kreyszig. *Advanced Engineering Mathematics*, page 765. John Wiley and Sons, fourth edition, 1979.

[42] T. E. Laskaris. Finite element analysis of three-dimensional potential flow in turbomachines. *AIAA Journal*, 16(7):717–722, July 1978.

[43] R. H. Ni. A multiple-grid scheme for solving the Euler equations. *AIAA Journal*, 20(11):1565–1571, November 1981.

[44] Ron-Ho R. Ni. *Prediction of 3D Multi-Stage Turbine Flow Using a Multiple-Grid Euler Solver.* AIAA-89-0203, 1989.

[45] R. A. Novak. *Streamline Curvature Computing Procedures for Fluid-Flow Problems.* ASME 66-WA/GT-3, November 1966.

[46] R. A. Novak and G. Haymann-Haber. *A Mixed-Flow Cascade Passage Design Procedure Based on a Power Series Expansion.* ASME 82-GT-121, 1982.

[47] Gordon C. Oates. *Aerothermodynamics of Gas Turbine and Rocket Propulsion.* American Institute of Aeronautics and Astronautics, 1988.

[48] Roger Peyret and Thomas D. Taylor. *Computational Methods for Fluid Flow.* Springer-Verlag, 1983.

[49] M. J. Pierzga and J. R. Wood. *Investigation of the Three-Dimensional Flow Field Within a Transonic Fan Rotor: Experiment and Analysis.* ASME-84-GT-200, June 1984.

[50] Sergio Pissanetzky. *Sparse Matrix Technology.* Academic Press, New York, 1984.

[51] T. C. Prince and A. C. Bryans. *Three Dimensional Inviscid Computation of an Impeller Flow.* ASME 83-GT-210, 1983.

[52] M. M. Rai. *Unsteady Three-dimensional Navier-Stokes Simulations of Turbine Rotor-Stator Interaction.* AIAA-87-2058, 1987.

[53] C. M. Rhie. *A Pressure Based Navier-Stokes Solver Using the Multigrid Method.* AIAA-86-0207, 1986.

[54] D.W. Riggins, R.W. Walters, and D. Pelletier. *The Use of Direct Solvers for Compressible Flow Computations.* AIAA-88-0229, 1988.

[55] Youcef Saad and Martin H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[56] Ascher H. Shapiro. *The Dynamics and Thermodynamics of Compressible Fluid Flow.* Volume I, John Wiley and Sons, 1953.

[57] L. H. Smith, Jr. Unducted fan aerodynamic design. *Transactions of the ASME Journal of Turbomachinery*, 109(3):313–324, July 1987.

[58] A. J. Strazisar. Investigation of flow phenomena in a transonic fan rotor using laser anemometry. *Journal of Engineering for Power*, 427–435, April 1985.

[59] W. T. Thompkins, S. S. Tong, R. H. Bush, W. J. Usab, and R. J. Norton. *Solution Procedures for Accurate Numerical Simulations of Flow in Turbomachinery Cascades.* AIAA-83-0257, January 1983.

[60] M.G. Turner and J.S. Keith. *An Implicit Algorithm for Solving 2D Rotational Flow in an Aircraft Engine Fan Frame.* AIAA-85-1534, 1985.

[61] Z. U. A. Warsi. *Tensors and Differential Geometry Applied to Analytic and Numerical Coordinate Generation.* Technical Report MSSU-EIRS-81-1, Mississippi State University, Engineering and Industrial Research Station, Department of Aerospace Engineering, January 1981.

[62] F.S. Weinig. Theory of two-dimensional flow through cascades. In W.R. Hawthorne, editor, *Aerodynamics of Turbines and Compressors, Volume X of the series High Speed Aerodynamics and Jet Propulsion*, chapter B, page 14, Princeton University Press, 1964.

[63] L.B. Wigton. *Application of MACSYMA and Sparse Matrix Technology to Multielement Airfoil Calculations.* AIAA-87-1142, June 1987.

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

[65] Jerry R. Wood, Anthony J. Strazisar, and P. Susan Simonyi. Shock structure in a transonic fan using laser anemometry. In *AGARD Conference Proceedings No. 401; Transonic and Supersonic Phenomena in Turbomachines*, chapter 2, AGARD, March 1987. AGARD-CP-401.

[66] Chung-Hua Wu. Three-dimensional turbomachine flow equations expressed with respect to non-orthogonal curvilinear coordinates and methods of solution. In *Third International Symposium on Air Breathing Engines*, March 1976.

# Appendix A
# Use of SMP (A 1D Example)

To demonstrate the use of SMP for linearizing equations, a 1D example will be explained. This 1D example was set up to investigate the pressure upwinding scheme.

The equation to be solved is the 1D momentum equation with area variation which is analogous to the system developed in 3D. The continuity equation is

$$\dot{m} = \rho V \lambda = \text{const.} \tag{A.1}$$

The conservative 1D momentum equation is:

$$\frac{d(\dot{m}V + P\lambda)}{dx} - P\frac{d\lambda}{dx} = 0, \tag{A.2}$$

and the discrete equation is

$$(\dot{m}V_{i+1} + \tilde{P}_{i+1}\lambda_{i+1}) - (\dot{m}V_i + \tilde{P}_i\lambda_i) - \frac{(P_i + P_{i+1})}{2}(\lambda_{i+1} - \lambda_i) = 0, \tag{A.3}$$

where $\lambda$ is the area and $\tilde{P}$ is the upwinded pressure. The energy equation is

$$H = \text{const.} \tag{A.4}$$

And the equation of state is

$$\rho = \frac{\gamma P + \sqrt{(\gamma P)^2 + 2(\gamma-1)^2 H(\rho V)^2}}{2(\gamma-1)H}, \tag{A.5}$$

$$= \frac{\gamma P + \sqrt{(\gamma P)^2 + 2(\gamma-1)^2 H\left(\frac{\dot{m}}{\lambda}\right)^2}}{2(\gamma-1)H}. \tag{A.6}$$

The upwinded pressure is

$$\tilde{P}_i = P_i + \mu(P_i - P_{i-1}), \tag{A.7}$$

where

$$\mu = \begin{cases} \frac{M_i^2}{1+(\gamma-1)M_i^2}\left(1-\frac{M_c^2}{M_i^2}\right) & ; \ M_i^2 \geq M_{i-1}^2 \text{ and } M_i^2 > M_c^2 \\ \frac{M_{i-1}^2}{1+(\gamma-1)M_{i-1}^2}\left(1-\frac{M_c^2}{M_{i-1}^2}\right) & ; \ M_{i-1}^2 \geq M_i^2 \text{ and } M_{i-1}^2 > M_c^2 \\ 0 & ; \ \text{otherwise} \end{cases} \tag{A.8}$$

$$M^2 = \frac{V^2}{c^2} = \frac{V^2}{(\gamma-1)(H-\frac{V^2}{2})}. \tag{A.9}$$

and

$$V = \frac{\dot{m}}{\rho\lambda}. \tag{A.10}$$

SMP must linearize the momentum equation with respect to the pressures. Given a three point stencil as shown in Fig. A.1, the momentum equation is to be solved between nodes 2 and 3. The values of $\lambda_2$ and $\lambda_3$ are known, along with $H$, $\dot{m}$, $\gamma$ and $M_c^2$. The pressures $P_1$, $P_2$ and $P_3$ are at the current iteration. The step-by-step algorithm is:

(For $i = 1$, 2, and 3)

$$\rho_i = \frac{\gamma P_i + \sqrt{(\gamma P_i)^2 + 2(\gamma-1)^2 H\left(\frac{\dot{m}}{\lambda_i}\right)^2}}{2(\gamma-1)H}, \tag{A.11}$$

$$V_i = \frac{\dot{m}}{\rho_i\lambda_i}, \tag{A.12}$$

$$M_i^2 = \frac{V_i^2}{(\gamma-1)(H-\frac{V_i^2}{2})}. \tag{A.13}$$

$$\mu_2 = \begin{cases} \frac{M_2^2}{1+(\gamma-1)M_2^2}\left(1-\frac{M_c^2}{M_2^2}\right) & ; \ M_2^2 \geq M_1^2 \text{ and } M_2^2 > M_c^2 \\ \frac{M_1^2}{1+(\gamma-1)M_1^2}\left(1-\frac{M_c^2}{M_1^2}\right) & ; \ M_1^2 \geq M_2^2 \text{ and } M_1^2 > M_c^2 \\ 0 & ; \ \text{otherwise} \end{cases} \tag{A.14}$$

$$\mu_3 = \begin{cases} \frac{M_3^2}{1+(\gamma-1)M_3^2}\left(1-\frac{M_c^2}{M_3^2}\right) & ; \ M_3^2 \geq M_2^2 \text{ and } M_3^2 > M_c^2 \\ \frac{M_2^2}{1+(\gamma-1)M_2^2}\left(1-\frac{M_c^2}{M_2^2}\right) & ; \ M_2^2 \geq M_3^2 \text{ and } M_2^2 > M_c^2 \\ 0 & ; \ \text{otherwise} \end{cases} \tag{A.15}$$

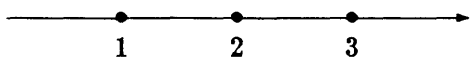


Figure A.1: Stencil for solving the 1D momentum equation between nodes 2 and 3.

$$\tilde{P}_2 = P_2 + \mu_2(P_2 - P_1), \tag{A.16}$$

$$\tilde{P}_3 = P_3 + \mu_3(P_3 - P_2), \tag{A.17}$$

$$EM = (\dot{m}V_3 + \tilde{P}_3\lambda_3)) - (\dot{m}V_2 + \tilde{P}_2\lambda_2)) - \frac{(P_2 + P_3)}{2}(\lambda_3 - \lambda_2). \tag{A.18}$$

Table A.1 shows the SMP/FORTRAN variables and their correspondence to the equation symbols used.

The sequence of equations has been set up in an SMP file. The approach is similar to that discussed by Wigton [63] for using MACSYMA. He discussed a naive approach which substitutes each sub-expression into each equation and then differentiates the resulting expression. A better approach is to take the partial derivative of each sub-expression with respect to each unknown, and then use the chain rule to determine the derivatives of the resulting expression. Several SMP procedures have been written to accomplish these tasks. These are listed in Section A.4. The capability of handling a case dependent equation such as Eqs. (A.14) and (A.15) has also been provided.

It was desired that a complete subroutine module be created automatically. Therefore, the SMP input file contains commands to print text to the file. This text contains the FORTRAN statements which are necessary to make it a complete subroutine. The file which is output has the extension `.SGF` for SMP Generated FORTRAN. This contains many embedded declarations and double precision constants. A routine written in VAX TPU (Text Processing Utility) is used to produce FORTRAN source which can be compiled from the `.SGF` file. The following section contains the SMP input for the 1D example problem.

Table A.1: SMP/FORTRAN symbols used in the 1D example. SMP uses lower case. FORTRAN is upper case.

| SMP/FORTRAN variable | symbol |
|---|---|
| mas | $\dot{m}$ |
| h | $H$ |
| gam | $\gamma$ |
| a2 | $\lambda_2$ |
| a3 | $\lambda_3$ |
| p1 | $P_1$ |
| p2 | $P_2$ |
| p3 | $P_3$ |
| mcs | $M_c^2$ |
| ro1 | $\rho_1$ |
| ro2 | $\rho_2$ |
| ro3 | $\rho_3$ |
| v1 | $V_1$ |
| v2 | $V_2$ |
| v3 | $V_3$ |
| m1s | $M_1^2$ |
| m2s | $M_2^2$ |
| m3s | $M_3^2$ |
| vs2 | $\mu_2$ |
| vs3 | $\mu_3$ |
| pb2 | $\tilde{P}_2$ |
| pb3 | $\tilde{P}_3$ |
| em | $EM$ |

## A.1 SMP Input for the 1D Momentum Equation

Below is listed the SMP input for the example problem. A few of the SMP commands are explained:

- Anything between the delimiters `/*` and `*/` is a comment and not interpreted by SMP.
- ! is used to access a VAX DCL command. This is copy, delete or create (@cgf) a file.
- < loads in an existing SMP procedure.
- Lpr outputs text to a file or the screen.
- \ is a continuation character.
- : is the assignment operator.
- `invar : {p1,p2,p3}` is the list of dependent variables. Each expression will be differentiated with respect to these variables. The resulting expression such as `v1d1` is the derivative of `v1` with respect to the first element in `invar`.
- `_var[Const] : 1` means var is a constant and will be treated as such by the differentiation operator.
- `deriv`, `casederiv` and `decfile` are SMP procedures listed in Section A.4.

```
/*
     This SMP input creates the FORTRAN source for the error in the 1D
momentum equation and its dependence on the static pressure at three
locations
     Case 2.  This is for first order upwinding with mu determined by
the Max of upstream or downstream Mach number, and with full linearization.
*/


/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mom1d.sgf;* mom1d.osf
!dele mom1d.sgf;*
!@cgf mom1d.sgf

filename : "mom1d.sgf"
```

```
/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

/* Output FORTRAN statements for the beginning of the subroutine. */
Lpr["
      SUBROUTINE MOM1D(I)
C     Momentum eq for 1-D.

      INTEGER I

C     T1DCOM is included and contains the parameter statement and common
C     arrays and variables.
      INCLUDE 'T1DCOM.FOR'

C     These declarations are generated by SMP.  When compiling, the /declare
C     option is used.  This ensures that there are no undefined variables.
      INCLUDE 'MOM1DDEC.FOR'

      REAL DERI

C     Case 2.  This is for first order upwinding with mu determined by
C     the Max of upstream or downstream Mach number, and with full
C     linearization.
      INTEGER CASE
      COMMON /CAS/CASE
      DATA CASE/2/

C     Get the face variables for this face.  Data passed
C     through common.  A1, A2, A3, P1, P2, and P3 are set here.
C     1 is for i-1, 2 is for i, and 3 is for i+1.  For i=1, point 1 is i.
      CALL IMOM(I)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. These are p1, p2, and p3. */
invar : {p1,p2,p3}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p1[Const] : 1 ; _p2[Const] : 1
_p3[Const] : 1 ; _mas[Const] : 1
_gam[Const] : 1 ; _mcs[Const] : 1
_h[Const] : 1 ; _a1[Const] : 1
_a2[Const] : 1 ; _a3[Const] : 1

/* Algorithm */
```

```
ro1 : (gam p1 + ( (gam p1)^2 + 2 (gam-1)^2 h (mas/a1)^2 )^(1/2) )/(2 (gam-1) h)
deriv["Rho 1.",'ro1,"mom1d.sgf"]

ro2 : (gam p2 + ( (gam p2)^2 + 2 (gam-1)^2 h (mas/a2)^2 )^(1/2) )/(2 (gam-1) h)
deriv["Rho 2.",'ro2,"mom1d.sgf"]

ro3 : (gam p3 + ( (gam p3)^2 + 2 (gam-1)^2 h (mas/a3)^2 )^(1/2) )/(2 (gam-1) h)
deriv["Rho 3.",'ro3,"mom1d.sgf"]

v1 : mas/(ro1 a1)
deriv["V 1.",'v1,"mom1d.sgf"]

v2 : mas/(ro2 a2)
deriv["V 2.",'v2,"mom1d.sgf"]

v3 : mas/(ro3 a3)
deriv["V 3.",'v3,"mom1d.sgf"]

m1s : v1^2/((gam-1) (h - v1^2/2))
deriv["(Mach1)^2.",'m1s,"mom1d.sgf"]

m2s : v2^2/((gam-1) (h - v2^2/2))
deriv["(Mach2)^2.",'m2s,"mom1d.sgf"]

m3s : v3^2/((gam-1) (h - v3^2/2))
deriv["(Mach3)^2.",'m3s,"mom1d.sgf"]

/* For the artificial viscosity at 2, use the maximum of the
Mach number at 1 or 2. */
vs2case : {m2s(1 - mcs/m2s)/(1 + (gam-1)m2s),\
           m1s(1 - mcs/m1s)/(1 + (gam-1)m1s),\
           0}
vs2cond : {"M2S.GE.M1S .AND. M2S.GT.MCS",\
           "M2S.LT.M1S .AND. M1S.GT.MCS",\
           ".TRUE."}
casederiv[3,"The artificial viscosity mu at 2.",vs2,vs2case,vs2cond,"mom1d.sgf"]

/* For the artificial viscosity at 3, use the maximum of the
Mach number at 2 or 3. */
vs3case : {m3s(1 - mcs/m3s)/(1 + (gam-1)m3s),\
           m2s(1 - mcs/m2s)/(1 + (gam-1)m2s),\
           0}
vs3cond : {"M3S.GE.M2S .AND. M3S.GT.MCS",\
           "M3S.LT.M2S .AND. M2S.GT.MCS",\
           ".TRUE."}
casederiv[3,"The artificial viscosity mu at 3.",vs3,vs3case,vs3cond,"mom1d.sgf"]

pb2 : p2 + vs2(p2 - p1)
deriv["Biased P at 2",'pb2,"mom1d.sgf"]

pb3 : p3 + vs3(p3 - p2)
deriv["Biased P at 3",'pb3,"mom1d.sgf"]
```

```
em : (mas v3 + pb3 a3) - (mas v2 + pb2 a2) - (p2 + p3)/2 (a3 - a2)
deriv["residual",'em,"mom1d.sgf"]

/* Output the FORTRAN statements which store the residual and derivatives
and end the subroutine. */
Lpr["
C     The residual.
      RHS(I) = -EM

C     Put Jacobians in correct location in matrix.
C     C2 is the diagonal of a tridiagonal matrix.
C     C1 and C3 are the off-diagonals of the tridiagonal matrix.
C     EMD1 is the dependence of the momentum error on p1.
C     EMD2 is the dependence of the momentum error on p2.
C     EMD3 is the dependence of the momentum error on p3.

      C1(I) = EMD1
      C2(I) = EMD2
      C3(I) = EMD3

C     Update the arrays in common for post processing.
      IF(I.EQ.1)THEN
        IF(M2S.GT.0.)THEN
          M(1) = SQRT(M2S)
        ELSE
          M(1) = 0.
        END IF
        MU(1) = VS2
        RHO(1) = RO2
        U(1) = V2
      END IF
      IF(M3S.GT.0.)THEN
        M(I+1) = SQRT(M3S)
      ELSE
        M(I+1) = 0.
      END IF
      MU(I+1) = VS3
      RHO(I+1) = RO3
      U(I+1) = V3

      RETURN
      END

",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mom1ddec.sgf;* mom1ddec.osf
!dele mom1ddec.sgf;*
!@cgf mom1ddec.sgf
```

```
<"decfile.in"

decfile["mom1ddec.sgf"]
```

## A.2 FORTRAN Subroutine Produced by SMP

Below is listed the resulting FORTRAN subroutine produced by SMP and the TPU (Text Processing Utility) procedure. Notice how the derivatives are handled. Also notice the way the case dependent equations, Eqs. (A.14) and (A.15), are applied using the `IF-THEN`, `ELSE IF` and `END IF` statements. This routine can be compiled and linked with a larger routine. This larger routine calls `MOM1D(I)` for `I=1,NI-1`. The back pressure is set so the boundary condition `P(NI)` is specified. The resulting tridiagonal matrix is solved and the pressures updated. This continues until convergence is achieved. In the following section, results obtained using this routine are shown.

```
      SUBROUTINE MOM1D(I)
C     Momentum eq for 1-D.

      INTEGER I

C     T1DCOM is included and contains the parameter statement and common
C     arrays and variables.
      INCLUDE 'T1DCOM.FOR'

C     These declarations are generated by SMP.  When compiling, the /declare
C     option is used.  This ensures that there are no undefined variables.
      INCLUDE 'MOM1DDEC.FOR'

      REAL DERI

C     Case 2.  This is for first order upwinding with mu determined by
C     the Max of upstream or downstream Mach number, and with full
C     linearization.
      INTEGER CASE
      COMMON /CAS/CASE
      DATA CASE/2/
```

```
C     Get the face variables for this face.  Data passed
C     through common.  A1, A2, A3, P1, P2, and P3 are set here.
C     1 is for i-1, 2 is for i, and 3 is for i+1.  For i=1, point 1 is i.
      CALL IMOM(I)

C     Rho 1.
      RO1 = 0.5E0 * ((GAM * P1 + (2 * ((H * MAS ** 2 * ((-1) + GAM) **
     $    2) / A1 ** 2) + GAM ** 2 * P1 ** 2) ** 0.5E0) / (H * ((-1) +
     $    GAM)))

C     Derivatives
      DERI = 0.5E0 * ((GAM + (GAM ** 2 * P1) / (2 * ((H * MAS ** 2 * ((
     $    -1) + GAM) ** 2) / A1 ** 2) + GAM ** 2 * P1 ** 2) ** 0.5E0) /
     $    (H * ((-1) + GAM)))

      ro1d1 = deri
C
C     Rho 2.
      RO2 = 0.5E0 * ((GAM * P2 + (2 * ((H * MAS ** 2 * ((-1) + GAM) **
     $    2) / A2 ** 2) + GAM ** 2 * P2 ** 2) ** 0.5E0) / (H * ((-1) +
     $    GAM)))

C     Derivatives
      DERI = 0.5E0 * ((GAM + (GAM ** 2 * P2) / (2 * ((H * MAS ** 2 * ((
     $    -1) + GAM) ** 2) / A2 ** 2) + GAM ** 2 * P2 ** 2) ** 0.5E0) /
     $    (H * ((-1) + GAM)))

      ro2d2 = deri
C
C     Rho 3.
      RO3 = 0.5E0 * ((GAM * P3 + (2 * ((H * MAS ** 2 * ((-1) + GAM) **
     $    2) / A3 ** 2) + GAM ** 2 * P3 ** 2) ** 0.5E0) / (H * ((-1) +
     $    GAM)))

C     Derivatives
      DERI = 0.5E0 * ((GAM + (GAM ** 2 * P3) / (2 * ((H * MAS ** 2 * ((
     $    -1) + GAM) ** 2) / A3 ** 2) + GAM ** 2 * P3 ** 2) ** 0.5E0) /
     $    (H * ((-1) + GAM)))

      ro3d3 = deri
C
C     V 1.
      V1 = MAS / (A1 * RO1)

C     Derivatives
      DERI = (-1) * ((MAS * RO1D1) / (A1 * RO1 ** 2))

      v1d1 = deri
C
C     V 2.
      V2 = MAS / (A2 * RO2)

C     Derivatives
```

```
      DERI = (-1) * ((MAS * RO2D2) / (A2 * RO2 ** 2))

      v2d2 = deri
C
C     V 3.
      V3 = MAS / (A3 * RO3)

C     Derivatives
      DERI = (-1) * ((MAS * RO3D3) / (A3 * RO3 ** 2))

      v3d3 = deri
C
C     (Mach1)^2.
      M1S = V1 ** 2 / (((-1) + GAM) * (H + (-0.5E0) * V1 ** 2))

C     Derivatives
      DERI = (V1 ** 3 * V1D1 * ((-1) + GAM) + 2 * V1 * V1D1 * ((-1) +
     $    GAM) * (H + (-0.5E0) * V1 ** 2)) / (((-1) + GAM) ** 2 * (H +
     $    (-0.5E0) * V1 ** 2) ** 2)

      m1sd1 = deri
C
C     (Mach2)^2.
      M2S = V2 ** 2 / (((-1) + GAM) * (H + (-0.5E0) * V2 ** 2))

C     Derivatives
      DERI = (V2 ** 3 * V2D2 * ((-1) + GAM) + 2 * V2 * V2D2 * ((-1) +
     $    GAM) * (H + (-0.5E0) * V2 ** 2)) / (((-1) + GAM) ** 2 * (H +
     $    (-0.5E0) * V2 ** 2) ** 2)

      m2sd2 = deri
C
C     (Mach3)^2.
      M3S = V3 ** 2 / (((-1) + GAM) * (H + (-0.5E0) * V3 ** 2))

C     Derivatives
      DERI = (V3 ** 3 * V3D3 * ((-1) + GAM) + 2 * V3 * V3D3 * ((-1) +
     $    GAM) * (H + (-0.5E0) * V3 ** 2)) / (((-1) + GAM) ** 2 * (H +
     $    (-0.5E0) * V3 ** 2) ** 2)

      m3sd3 = deri
C
      IF(M2S.GE.M1S .AND. M2S.GT.MCS)THEN
C     The artificial viscosity mu at 2.
      VS2 = (M2S * (1 + (-1) * (MCS / M2S))) / (1 + M2S * ((-1) + GAM))

C     Derivatives
      DERI = 0

         vs2d1 = deri
C
C     Derivatives
      DERI = ((1 + M2S * ((-1) + GAM)) * ((M2SD2 * MCS) / M2S + M2SD2 *
     $    (1 + (-1) * (MCS / M2S))) + (-1) * M2S * M2SD2 * ((-1) + GAM)
     $    * (1 + (-1) * (MCS / M2S))) / (1 + M2S * ((-1) + GAM)) ** 2

        vs2d2 = deri
C
      ELSE IF(M2S.LT.M1S .AND. M1S.GT.MCS)THEN
C     The artificial viscosity mu at 2.
      VS2 = (M1S * (1 + (-1) * (MCS / M1S))) / (1 + M1S * ((-1) + GAM))

C     Derivatives
      DERI = ((1 + M1S * ((-1) + GAM)) * ((M1SD1 * MCS) / M1S + M1SD1 *
     $    (1 + (-1) * (MCS / M1S))) + (-1) * M1S * M1SD1 * ((-1) + GAM)
     $    * (1 + (-1) * (MCS / M1S))) / (1 + M1S * ((-1) + GAM)) ** 2

        vs2d1 = deri
C
C     Derivatives
      DERI = 0

        vs2d2 = deri
C
      ELSE IF(.TRUE.)THEN
C     The artificial viscosity mu at 2.
      VS2 = 0

C     Derivatives
      DERI = 0

        vs2d1 = deri
C
C     Derivatives
      DERI = 0

        vs2d2 = deri
C
      END IF
      IF(M3S.GE.M2S .AND. M3S.GT.MCS)THEN
C     The artificial viscosity mu at 3.
      VS3 = (M3S * (1 + (-1) * (MCS / M3S))) / (1 + M3S * ((-1) + GAM))

C     Derivatives
      DERI = 0

        vs3d2 = deri
C
C     Derivatives
      DERI = ((1 + M3S * ((-1) + GAM)) * ((M3SD3 * MCS) / M3S + M3SD3 *
     $    (1 + (-1) * (MCS / M3S))) + (-1) * M3S * M3SD3 * ((-1) + GAM)
     $    * (1 + (-1) * (MCS / M3S))) / (1 + M3S * ((-1) + GAM)) ** 2

        vs3d3 = deri
C
      ELSE IF(M3S.LT.M2S .AND. M2S.GT.MCS)THEN
```

```
C     The artificial viscosity mu at 3.
      VS3 = (M2S * (1 + (-1) * (MCS / M2S))) / (1 + M2S * ((-1) + GAM))

C     Derivatives
      DERI = ((1 + M2S * ((-1) + GAM)) * ((M2SD2 * MCS) / M2S + M2SD2 *
     $    (1 + (-1) * (MCS / M2S))) + (-1) * M2S * M2SD2 * ((-1) + GAM)
     $    * (1 + (-1) * (MCS / M2S))) / (1 + M2S * ((-1) + GAM)) ** 2

        vs3d2 = deri
C
C     Derivatives
      DERI = 0

        vs3d3 = deri
C
      ELSE IF(.TRUE.)THEN
C     The artificial viscosity mu at 3.
      VS3 = 0

C     Derivatives
      DERI = 0

        vs3d2 = deri
C
C     Derivatives
      DERI = 0

        vs3d3 = deri
C
      END IF
C     Biased P at 2
      PB2 = P2 + VS2 * ((-1) * P1 + P2)

C     Derivatives
      DERI = (-1) * VS2 + VS2D1 * ((-1) * P1 + P2)

      pb2d1 = deri
C
C     Derivatives
      DERI = 1 + VS2 + VS2D2 * ((-1) * P1 + P2)

      pb2d2 = deri
C
C     Biased P at 3
      PB3 = P3 + VS3 * ((-1) * P2 + P3)

C     Derivatives
      DERI = (-1) * VS3 + VS3D2 * ((-1) * P2 + P3)

      pb3d2 = deri
C
C     Derivatives
      DERI = 1 + VS3 + VS3D3 * ((-1) * P2 + P3)
      pb3d3 = deri
C
C     residual
      EM = (-1) * A2 * PB2 + A3 * PB3 + (-1) * MAS * V2 + MAS * V3 + (
     $    -0.5E0) * ((-1) * A2 + A3) * (P2 + P3)

C     Derivatives
      DERI = (-1) * A2 * PB2D1

      emd1 = deri
C
C     Derivatives
      DERI = 0.5E0 * A2 + (-0.5E0) * A3 + (-1) * A2 * PB2D2 + A3 *
     $    PB3D2 + (-1) * MAS * V2D2

      emd2 = deri
C
C     Derivatives
      DERI = 0.5E0 * A2 + (-0.5E0) * A3 + A3 * PB3D3 + MAS * V3D3

      emd3 = deri
C
C     The residual.
      RHS(I) = -EM

C     Put Jacobians in correct location in matrix.
C     C2 is the diagonal of a tridiagonal matrix.
C     C1 and C3 are the off-diagonals of the tridiagonal matrix.
C     EMD1 is the dependence of the momentum error on p1.
C     EMD2 is the dependence of the momentum error on p2.
C     EMD3 is the dependence of the momentum error on p3.

      C1(I) = EMD1
      C2(I) = EMD2
      C3(I) = EMD3

C     Update the arrays in common for post processing.
      IF(I.EQ.1)THEN
        IF(M2S.GT.0.)THEN
          M(1) = SQRT(M2S)
        ELSE
          M(1) = 0.
        END IF
        MU(1) = VS2
        RHO(1) = RO2
        U(1) = V2
      END IF
      IF(M3S.GT.0.)THEN
        M(I+1) = SQRT(M3S)
      ELSE
        M(I+1) = 0.
      END IF
      MU(I+1) = VS3
      RHO(I+1) = RO3
      U(I+1) = V3

      RETURN
      END
```

## A.3 Results of 1D Example

The SMP generated subroutine was the main routine for a program which investigated the pressure upwinding scheme. A Laval nozzle was investigated with the following area distribution:

$$\lambda = \begin{cases} \frac{1}{8} & x < 0, x > 1 \\ \frac{1}{8} - \frac{1}{40}\sin(\pi x) & x < 0, x > 1 \end{cases} \tag{A.19}$$

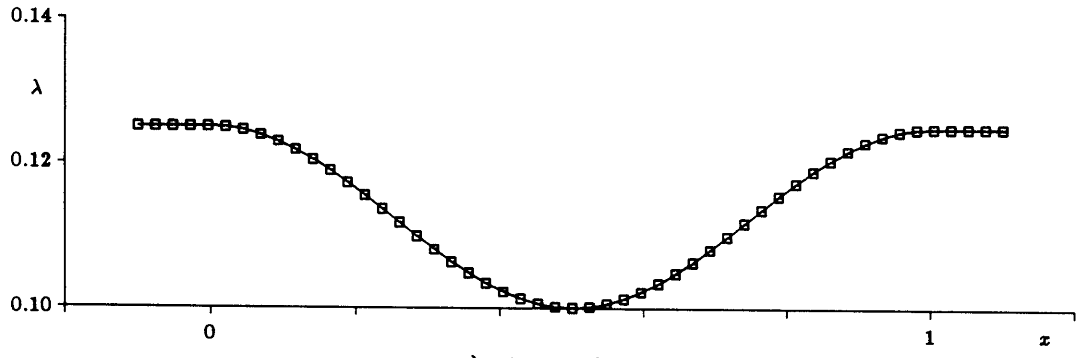
with inlet at $x = -0.1$ and exit at $x = 1.1$. Upstream, $P_T = 1$, $H = 1$, $\gamma = 1.4$. The choked flow rate, $\dot{m}$, for $\lambda = 0.1$ is calculated. The exit pressure $P_e$ is specified and the pressure initialized as

$$P_{init} = \begin{cases} P_1 & x < 0 \\ P_e & x > 1 \\ P_1 + x(P_e - P_1) & x < 0, x > 1 \end{cases} \tag{A.20}$$

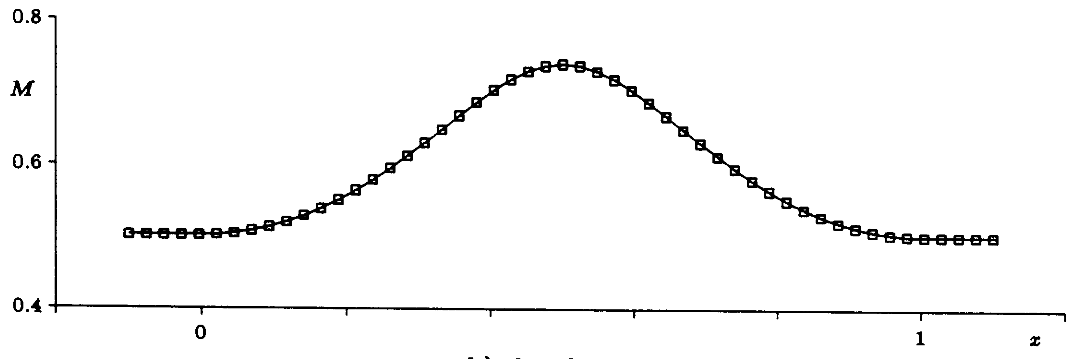
where $P_1 = 0.814 P_T$.

For a back pressure of $P_e = 0.9$, the flow is subsonic and the solution for $ni = 51$ converged to machine accuracy in three Newton iterations. This demonstrates the linearization was done correctly. The nozzle area, Mach number, pressure and total pressure loss distributions for this back pressure are shown in Fig. A.2.

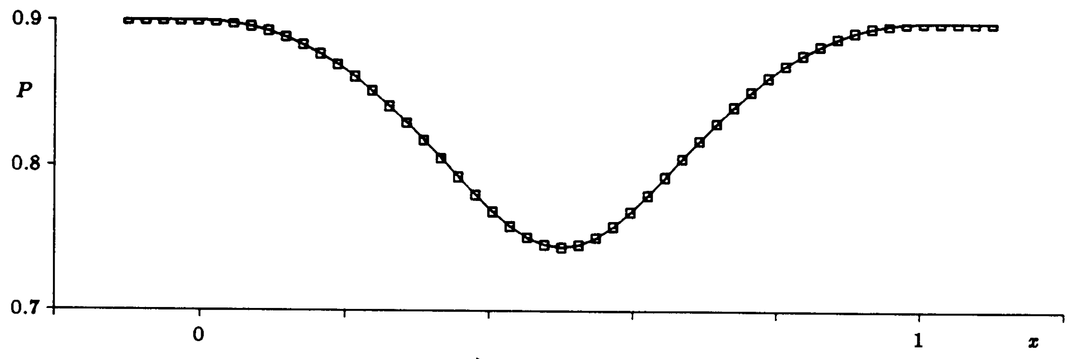
For a back pressure of $P_e = 0.7$, the flow is transonic. $M_c^2 = 1$ was chosen and $ni = 51$. This solution took 16 Newton iterations due to the poor initial solution and the strong nonlinearities. The Mach number, pressure, total pressure loss and artificial viscosity distributions are shown in Fig. A.3.



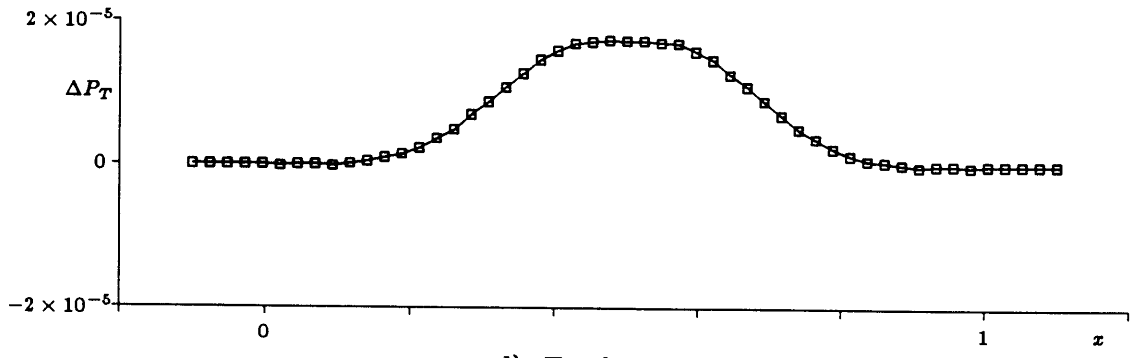
a) Area Distribution.



b) Mach Number.

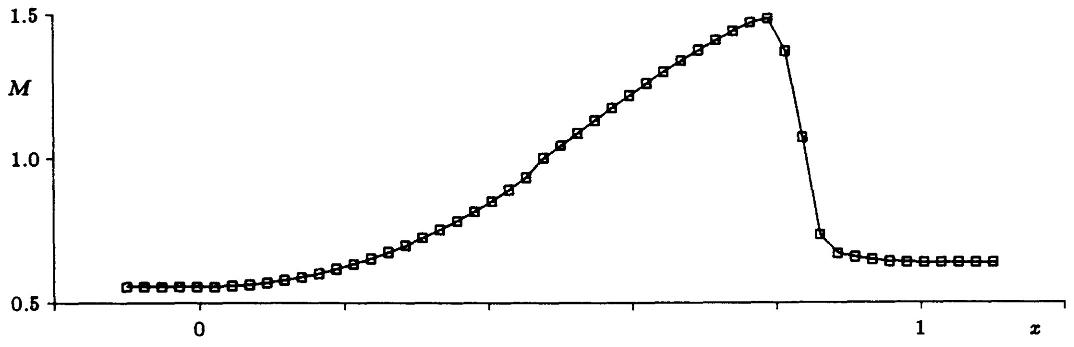


c) Static Pressure.

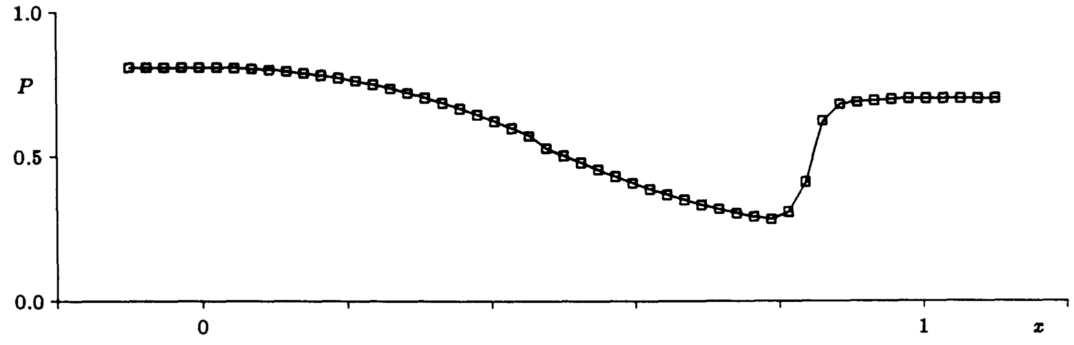


d) Total Pressure Loss.

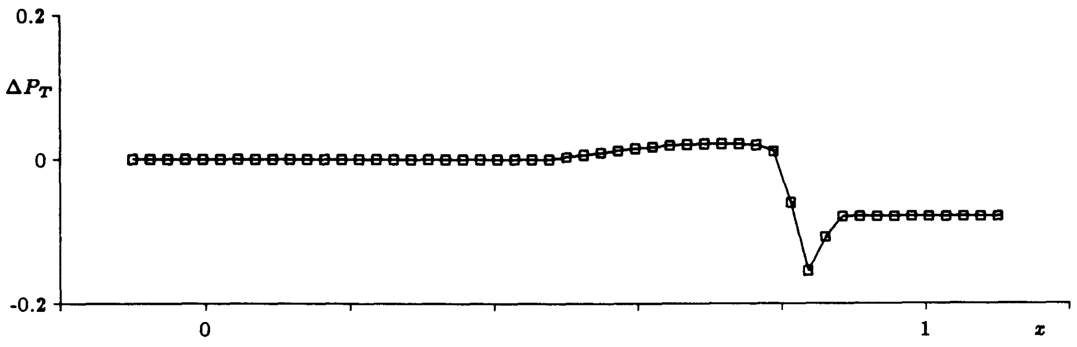
Figure A.2: One dimensional results for Laval nozzle. Back pressure of 0.9, 51 points. Converged in 3 Newton iterations.



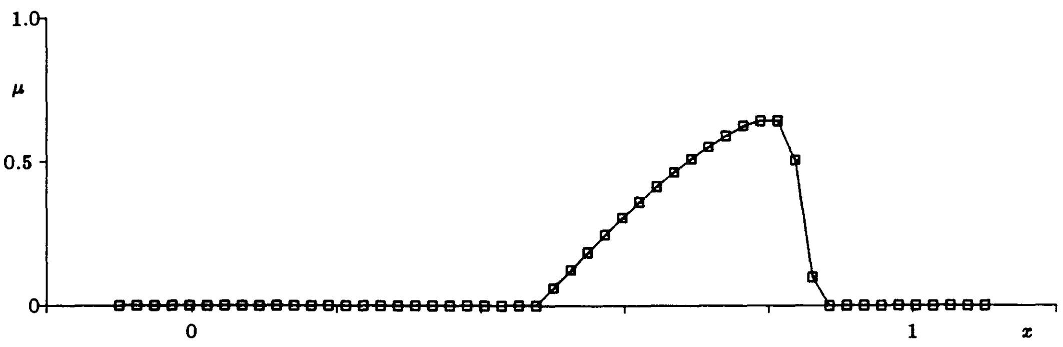
b) Mach Number.



b) Static Pressure.



c) Total Pressure Loss.



d) Artificial Viscosity Factor.

Figure A.3: One dimensional results for Laval nozzle. Back pressure of 0.7, 51 points, $M_c^2 = 1$. Converged in 16 Newton iterations.

## A.4 SMP Procedures

The SMP procedures for differentiating and outputting FORTRAN code are listed in the following subsections. One of these procedures, `decfile`, writes a file containing variable declarations and initializations of expressions in the list `exprlist`. In each of the other procedures, each new expression name is added to this list. This declaration file is then included in the subroutine. Compiling with the `/DECLARE` option on the VAX checks that all variables are declared. This reduces the chance of undefined variables and ensures that an algorithm in SMP is complete.

A brief explanation of some of the additional SMP commands follows:

- a `$` indicates a symbol which represents an argument in a procedure.
- `Proc` defines a procedure.
- `Prog` converts an expression to FORTRAN code.
- `Lcl` declares local symbols.
- `Dt` takes the total derivative of the first argument with respect to the second.
- `Do` and `If` are do loop and conditional constructs.
- `Make` concatenates text and converts numbers to text.

### A.4.1 FORTRAN Generation Procedure

The SMP procedure for converting an SMP expression to FORTRAN and writing this to a file is listed below. It also takes care of creating the expression list and has the same form as the derivative routine.

```
/* This procedure outputs a comment and expression to the SMP generated
FORTRAN file.  The expression name is then added to exprlist so a declaration
file can be written.  The expression is then nulled. The expression must be
held on input by '.  The expression nel must be initialized. */
```

```
fort[$comment,$expr,$filename] :: Proc[\
Lpr[Make["C     ",$comment],$filename],\
Prog[$expr,$filename,,2],\
$expr : ,\
nel : nel + 1 ,\
exprlist[nel] : Rel[$expr] ,\
]
```

### A.4.2 Derivative Procedure

The SMP procedure for differentiating an SMP expression with respect to several variables, converting these to FORTRAN and writing to a file is:

```
/* This procedure finds the derivatives of an expression w.r.t. the
niv independant variables stored in invar.  The symbol for these derivatives
will be (expression symbol)di where i is the number of the variable in invar.
The expression and derivatives will then be written as FORTRAN statements to
the file specified.  The symbols will be nulled and the derivatives defined.
The procedure fort is used and must also be input.  The expression
nel must be initalized.  */

deriv[$comment,$expr,$filename] :: Proc[\
Lcl[i,dii,der,isder,deri],\
/* Get derivatives */ \
Do[i,niv,\
  der[i] : Dt[Rel[$expr],invar[i]];\
  If[der[i]=0,\
    isder[i] : 0,\
    isder[i] : 1,\
    isder[i] : 1 \
  ]\
],\
Lpr["derivatives:"],\
Lpr[der],\
/* Convert expression and derivatives to FORTRAN. */ \
fort[$comment,$expr,$filename],\
Do[i,niv,\
  dii : Make[d,i];\
  If[isder[i]=1,\
    Lpr["C     Derivatives",$filename];\
    deri : der[i];\
    Prog[deri,$filename,,2];\
    Lpr[Make[  Make[ "      ",Make[Rel[$expr],dii] ]," = deri"  ],$filename];\
    Lpr["C     ",$filename];\
    deri :  ;\
    nel : nel + 1 ;\
    exprlist[nel] : Make[Rel[$expr],dii] \
  ]\
],\
/* Assign derivatives. */ \
Do[i,niv,\
  dii : Make[d,i];\
  If[isder[i]=1,\
    Dt[Rel[$expr],invar[i]] : Make[Rel[$expr],dii],\
    Dt[Rel[$expr],invar[i]] : 0\
  ]\
],\
]
```

### A.4.3 Case FORTRAN Generation Procedure

The SMP procedure for handling a case expression and converting this to FORTRAN is:

```
/* This procedure is similar to the function fort, except that an expression
depends on a certain condition.  Therefore, a list of $nc expressions ($exprc)
and conditions ($cond) are input.  The expression name is $expr.  The IF
statements will be written along with the FORTRAN version of the expression.
Invoke the routine as follows for an expression a defined as ac
(the filename is not used so the result is printed on the screen):

nel : 0
invar : {x,y,z}
niv : Len[invar]
ac : {x^2 + y^2,x^3}
cond : {"IA.EQ.0","IA.EQ.1"}
_x[Const] : 1 ; _y[Const] : 1 ; _z[Const] : 1
casefort[2,"expression a",a,ac,cond,]

*/

casefort[$nc,$comment,$expr,$exprc,$cond,$filename] :: Proc[\
Lcl[i,dii,der,isder,deri,j],\
/* For each case, output the expression. */ \
Do[j,$nc,\
  /* Print IF statement to file. */ \
  If[j=1,\
    Lpr[Make[  Make[ "      IF(",$cond[j]], ")THEN"  ],$filename],\
    Lpr[Make[  Make[ "      ELSE IF(",$cond[j]], ")THEN"  ],$filename],\
    Lpr[Make[  Make[ "      ELSE IF(",$cond[j]], ")THEN"  ],$filename]\
  ];\
  Lpr[Make["C     ",$comment],$filename];\
  $expr : $exprc[j]; \
  Prog[$expr,$filename,,2];\
  $expr : ; \
  nel : nel + 1 ;\
  exprlist[nel] : Rel[$expr] ;\
],\
Lpr["      END IF",$filename],\
/* Set the expression to Null so it won't be substituted */ \
$exprc : ,\
]
```

### A.4.4 Case Derivative Procedure

The SMP procedure for taking the derivatives of a case expression and converting this to FORTRAN is:

```
/* This procedure is similar to the function deriv, except that an expression
depends on a certain condition.  Therefore, a list of $nc expressions ($exprc)
and conditions ($cond) are input.  The expression name is $expr.  The IF
statements will be written along with the FORTRAN version of the expression
and the derivatives.  Invoke the routine as follows for an expression a
defined as ac (the filename is not used so the result is printed on the screen):

nel : 0
invar : {x,y,z}
niv : Len[invar]
ac : {x^2 + y^2,x^3}
cond : {"IA.EQ.0","IA.EQ.1"}
_x[Const] : 1 ; _y[Const] : 1 ; _z[Const] : 1
casederiv[2,"expression a",a,ac,cond,]

*/

casederiv[$nc,$comment,$expr,$exprc,$cond,$filename] :: Proc[\
Lcl[i,dii,der,isder,deri,j],\
/* Get derivatives */ \
Do[i,niv,\
  isder[i] : 0;\
  Do[j,$nc,\
    der[j,i] : Dt[Rel[$exprc[j]],invar[i]];\
    If[der[j,i]=0,\
      ,\
      isder[i] : 1,\
      isder[i] : 1 \
    ]\
  ]\
],\
/* For each case, output the expression and derivatives. */ \
Do[j,$nc,\
  Lpr["derivatives: Case"];\
  Lpr[j];\
  Lpr[der[j]];\
  /* Print IF statement to file. */ \
  If[j=1,\
    Lpr[Make[  Make[ "      IF(",$cond[j]], ")THEN"  ],$filename],\
    Lpr[Make[  Make[ "      ELSE IF(",$cond[j]], ")THEN"  ],$filename],\
    Lpr[Make[  Make[ "      ELSE IF(",$cond[j]], ")THEN"  ],$filename]\
  ];\
  Lpr[Make["C     ",$comment],$filename];\
  $expr : $exprc[j]; \
  Prog[$expr,$filename,,2];\
  $expr : ; \
  nel : nel + 1 ;\
  exprlist[nel] : Rel[$expr] ;\
  Do[i,niv,\
    dii : Make[d,i];\
    If[isder[i]=1,\
      Lpr["C     Derivatives",$filename];\
      deri : der[j,i];\
      Prog[deri,$filename,,2];\
      Lpr[Make[  Make[ "        ",Make[$expr,dii] ]," = deri"  ],$filename];\
      Lpr["C     ",$filename];\
      deri :  ;\
      nel : nel + 1 ;\
      exprlist[nel] : Make[Rel[$expr],dii] \
    ]\
  ]\
],\
Lpr["      END IF",$filename],\
/* Set the expression to Null so it won't be substituted */ \
$exprc : ,\
/* Assign derivatives. */ \
Do[i,niv,\
  dii : Make[d,i];\
  If[isder[i]=1,\
    Dt[Rel[$expr],invar[i]] : Make[Rel[$expr],dii],\
    Dt[Rel[$expr],invar[i]] : 0\
  ]\
],\
]
```

## A.4.5 Expression Declaration Procedure

The SMP procedure for writing out FORTRAN declarations is:

```
/* This procedure takes the expressions in exprlist and writes to
```

```
the file $filename the declarations for these as real variables . */

decfile[$filename] :: Proc[ \
Lcl[comlist,ncl] ,\
comlist : Union[exprlist] ,\
ncl : Len[comlist] ,\
Do[i,ncl,Lpr[Make["      REAL ",comlist[i]],$filename]] ,\
Do[i,ncl,Lpr[Make[Make["      DATA ",comlist[i]],"/0./"],$filename]] ,\
]
```

# Appendix B

# Pressure Upwinding Stability Analysis

The steady transonic flow calculations using the Euler equations requires some artificial dissipation to exclude non-physical solutions. The jump conditions across a flow discontinuity are derived from the conservative Euler equations [56]. In this derivation, a loss of entropy is predicted for a rarefaction shock. This violates the second law of thermodynamics and is therefore impossible. No information about the second law is contained in the Euler equations. Therefore, to exclude the rarefaction shock, a mechanism must be added to allow only the physical solution. This is an artificial dissipation term which acts like physical dissipation and is an irreversible process. To determine the form of this dissipation, a stability analysis has been performed. The approach follows directly from Giles [27] who applied the dissipation by upwinding density. Drela [16] used a bulk viscosity approach which upwinds velocity. In this analysis, an upwinded pressure is used.

Pressure has been chosen as a primary variable (as opposed to density, which is used in ISES) so that incompressible solutions are also possible. It was, therefore, felt that a simpler linearized equation would result if a pressure upwinding scheme was used.

For a uniform compressible flow in a constant area channel, the steady Euler equations are:

$$\rho u = \dot{m} = \text{constant}, \tag{B.1a}$$

$$\dot{m} u + P + \mu \, \Delta x \, \frac{dP}{dx} = \text{constant}, \tag{B.1b}$$

$$\frac{\gamma}{\gamma - 1} \frac{P}{\rho} + \frac{u^2}{2} = \text{constant}. \tag{B.1c}$$

A dissipative term has been introduced in the momentum equation through the pressure gradient. A perturbation analysis is now performed such that

$$P = \bar{P}(1 + P'), \tag{B.2a}$$

$$\rho = \bar{\rho}(1 + \rho'), \tag{B.2b}$$

$$u = \bar{u} + \bar{c}u', \tag{B.2c}$$

where

$$\bar{c} = \frac{\gamma \bar{P}}{\bar{\rho}}, \tag{B.3}$$

and is the local speed of sound.

Substituting Eq. (B.2) into Eq. (B.1), and neglecting higher order terms yields the following linearized equations:

$$u' + M\rho' = 0, \tag{B.4a}$$

$$2\gamma M u' + \gamma M^2 \rho' + P' + \mu \, \Delta x \, \frac{dP'}{dx} = 0, \tag{B.4b}$$

$$P' - \rho' + (\gamma - 1) M u' = 0, \tag{B.4c}$$

where $M = \bar{u}/\bar{c}$ is the base flow Mach number. Eliminating $u'$ and $\rho'$ from these equations yields the following first order ODE for $P'$:

$$\frac{dP'}{dx} - kP' = 0 \quad \text{where} \quad k = \frac{1}{\mu \, \Delta x} \left( \frac{M^2 - 1}{1 + (\gamma - 1) M^2} \right). \tag{B.5}$$

This equation has exponential solutions of the form:

$$P' \propto \exp(kx). \tag{B.6}$$

The value of $k$ changes sign at $M = 1$. In subsonic regions, $k < 0$ implying that perturbations decay away downstream, and in supersonic regions, $k > 0$ implying perturbations decay away upstream. For a shock which goes from supersonic to subsonic, the perturbations decay away from the shock.

The discrete form of the Euler equations are:

$$\rho_2 u_2 = \dot{m} = \text{constant}, \tag{B.7a}$$

$$\dot{m} u_2 + P_2 + \mu(P_2 - P_1) = \text{constant}, \tag{B.7b}$$

$$\frac{\gamma}{\gamma - 1} \frac{P_2}{\rho_2} + \frac{u_2^2}{2} = \text{constant}. \tag{B.7c}$$

After linearizing and combining equations, the following difference equation results:

$$(\mu - \mu_c)P_2' - \mu P_1' = 0 \tag{B.8}$$

where

$$\mu_c = \left(\frac{M^2 - 1}{1 + (\gamma - 1)M^2}\right). \tag{B.9}$$

The solution to Eq. (B.8) is

$$P_i' \propto z^i \quad \text{where} \quad z = \frac{\mu}{\mu - \mu_c}, \tag{B.10}$$

and $i$ is the spatial mode.

Consistency with the analytic equation behavior requires that the perturbation decay upstream for supersonic flow and downstream for subsonic flow.

For subsonic flow, $M < 1$, $\mu_c < 0$, $z < 1$. Therefore, perturbations always decay downstream and consistency is achieved.

For supersonic flow, $M > 1$ and there are three regions to consider:

1. $\mu > \mu_c \implies 1 < z < \infty$ which implies the consistent exponential decay upstream.
2. $\mu_c > \mu > \mu_c/2 \implies -\infty < z < -1$. This implies the consistent exponential decay upstream, but since $z$ is negative, $P'$ will alternate sign yielding an oscillatory decay.
3. $\mu_c/2 > \mu > 0 \implies -1 < z < 0$. In this range, the perturbation does not decay, because $|z| < 1$. There is not enough artificial dissipation to be consistent with the proper analytic decay. The Jacobian matrix becomes nearly singular and a converged solution is almost impossible.

This analysis has provided a form for $\mu$ to take as well as its magnitude. For $\mu = \mu_c$, the most rapid decay is possible resulting in the sharpest shocks. However, $\mu = \mu_c/2$ provides the lowest value on $\mu$ while still being consistent, and should therefore produce the lowest dissipation error. To allow for this analysis to actually represent a non-uniform 3D flow, the dissipation should be turned on slightly below Mach one, or at

$M = M_c$. The actual formula for $\mu$ is therefore

$$\mu = \begin{cases} \frac{M^2}{1+(\gamma-1)M^2}\left(1 - \frac{M_c^2}{M^2}\right), & M > M_c \\ 0, & M \leq M_c \end{cases} . \tag{B.11}$$

# Appendix C

# Additional Useful Relations for Turbomachinery Applications

For turbomachinery applications, some additional relations are useful for pre- and post-processing.

Based on the cylindrical coordinate system described in Chapter 3, the relative velocity is

$$\overline{W} = W_\theta \hat{e}_\theta + W_r \hat{e}_r + W_z \hat{k}, \tag{C.1}$$

the absolute velocity is

$$\overline{C} = C_\theta \hat{e}_\theta + C_r \hat{e}_r + C_z \hat{k}, \tag{C.2}$$

and from the sign convention on $\omega$, the velocity of the coordinate system is:

$$\overline{U} = -\omega r \hat{e}_\theta. \tag{C.3}$$

Since

$$\overline{C} = \overline{W} + \overline{U}, \tag{C.4}$$

then

$$C_\theta = W_\theta - \omega r, \tag{C.5a}$$

$$C_r = W_r = V_r, \tag{C.5b}$$

$$C_z = W_z = V_z. \tag{C.5c}$$

As shown in Fig. C.1, the component of velocity in the $r$-$z$ plane is

$$V_m = \sqrt{V_r^2 + V_z^2}, \tag{C.6}$$

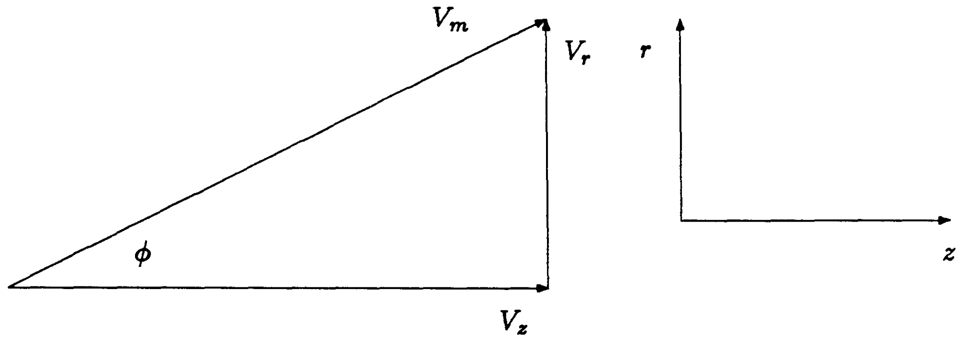


Figure C.1: Velocity triangle in the $r$-$z$ plane.

and its orientation is defined by

$$\phi = \arctan\left(\frac{V_r}{V_z}\right). \tag{C.7}$$

For an $m'$ surface which is a surface of revolution in the $r$-$z$ plane, $\phi'$ is the slope of this surface.

$$\phi' = \arctan\left(\frac{dr}{dz}\right). \tag{C.8}$$

In Fig. C.2, the velocity triangle in the $\theta$-$z$ plane is shown. Based on this:

$$\tan\alpha = \frac{C_\theta}{C_z}, \tag{C.9}$$

$$\tan\beta = \frac{W_\theta}{W_z}. \tag{C.10}$$

A velocity triangle can also be drawn in the $m'$-$\theta$ plane as shown in Fig. C.3. Based on this figure:

$$\tan\alpha_m = \frac{C_\theta}{V_m} = \cos\phi\frac{C_\theta}{C_z}, \tag{C.11}$$

$$\tan\beta_m = \frac{W_\theta}{V_m} = \cos\phi\frac{W_\theta}{W_z}. \tag{C.12}$$

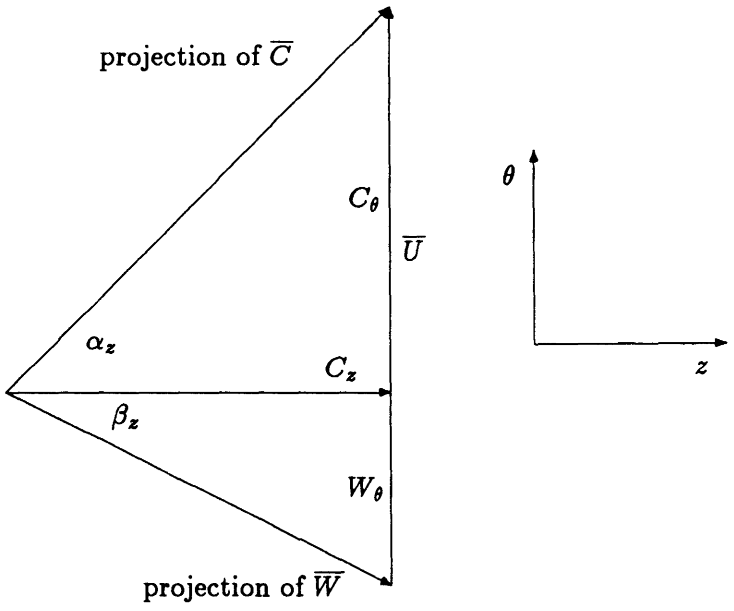


Figure C.2: Velocity triangle in the $\theta$-$z$ plane.

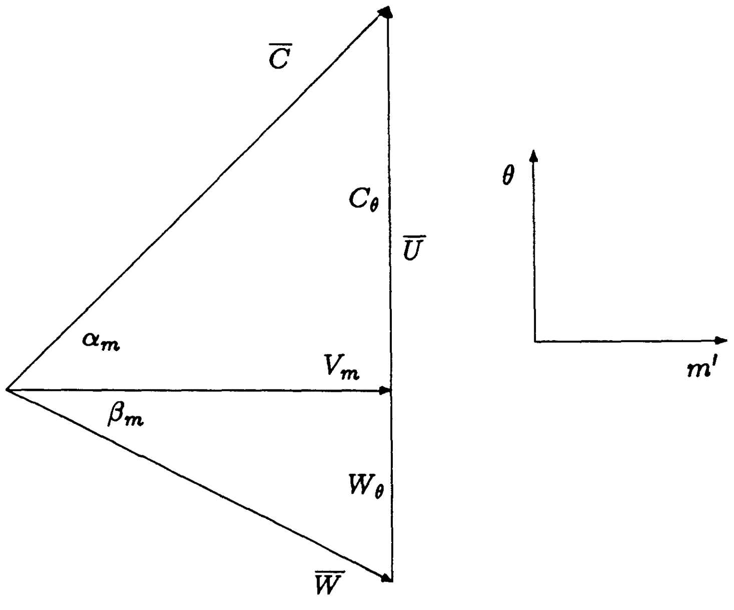


Figure C.3: Velocity triangle in the $m'$-$\theta$ plane.

The absolute total pressure is determined by the relations:

$$P_T = P\left(\frac{H}{h}\right)^{\gamma/(\gamma-1)}, \tag{C.13}$$

$$h = \frac{\gamma}{\gamma-1}\frac{P}{\rho}, \tag{C.14}$$

$$H = h + \frac{C^2}{2}. \tag{C.15}$$

An alternative formula for rothalpy is based on substituting

$$\begin{aligned} W^2 &= W_\theta^2 + W_r^2 + W_z^2, \\ W^2 &= (C_\theta + \omega r)^2 + C_r^2 + C_z^2, \\ W^2 &= C^2 + 2\omega r C_\theta + (\omega r)^2, \end{aligned} \tag{C.16}$$

into the definition for rothalpy:

$$\begin{aligned} I &= h + \frac{W^2}{2} - \frac{(\omega r)^2}{2}, \\ I &= h + \frac{C^2}{2} + \omega r C_\theta, \\ I &= H + \omega r C_\theta. \end{aligned} \tag{C.17}$$

This shows why for a fan with constant $T_T$ and $C_\theta = 0$, the rothalpy is uniform. It also shows that for $I =$ constant along a streamline the work added in a rotor $(\Delta H)$ is $\omega\Delta(rC_\theta)$.

# Appendix D

# Stream Function Derivative Discretization for Each Face Type

This appendix contains the interior $A$ and $B$ face stencils which are used for the stream function derivatives as well as the stencils used for 2D problems ($ni = 2$ or $nj = 2$). The interior $S$ face stencil is in Chapter 3.

The interior $A$ face stencil is shown in Fig. D.1. The derivatives are discretized at this $A$ face as follows:

$$\left.\frac{\partial\psi_1}{\partial\xi^1}\right|_A = \frac{1}{2}\left(\psi_1|_7 - \psi_1|_5\right), \tag{D.1a}$$

$$\left.\frac{\partial\psi_1}{\partial\xi^2}\right|_A = \frac{1}{2}\left(\psi_1|_8 - \psi_1|_4\right), \tag{D.1b}$$

$$\left.\frac{\partial\psi_1}{\partial\xi^3}\right|_A = \frac{1}{2}\left(\psi_1|_{10} - \psi_1|_2\right), \tag{D.1c}$$

$$\left.\frac{\partial\psi_2}{\partial\xi^1}\right|_A = \frac{1}{2}\left[\left(\psi_2|_6 + \psi_2|_8\right) - \left(\psi_2|_5 + \psi_2|_7\right)\right], \tag{D.2a}$$

$$\left.\frac{\partial\psi_2}{\partial\xi^2}\right|_A = \frac{1}{2}\left[\left(\psi_2|_7 + \psi_2|_8\right) - \left(\psi_2|_5 + \psi_2|_6\right)\right], \tag{D.2b}$$

$$\left.\frac{\partial\psi_2}{\partial\xi^3}\right|_A = \frac{1}{8}\left[\left(\psi_2|_9 + \psi_2|_{10} + \psi_2|_{11} + \psi_2|_{12}\right) - \left(\psi_2|_1 + \psi_2|_2 + \psi_2|_3 + \psi_2|_4\right)\right]. \tag{D.2c}$$

Two sets of grid derivatives are evaluated. One for the $\psi_1$ gradients which use finite difference operators similar to the $\psi_1$ derivatives, and one for the $\psi_2$ gradients which use operators similar to the $\psi_2$ derivatives.

To evaluate the mass flux, the stream functions are needed at the vertices of the face. The interpolation formulas are:

$$\psi_1|_a = \frac{1}{4}\left(\psi_1|_4 + \psi_1|_6 + \psi_1|_9 + \psi_1|_{10}\right), \tag{D.3a}$$

$$\psi_1|_b = \frac{1}{4}\left(\psi_1|_6 + \psi_1|_8 + \psi_1|_{10} + \psi_1|_{11}\right), \tag{D.3b}$$

$$\psi_1|_c = \frac{1}{4}\left(\psi_1|_2 + \psi_1|_3 + \psi_1|_6 + \psi_1|_8\right), \tag{D.3c}$$

$$\psi_1|_d = \frac{1}{4}\left(\psi_1|_1 + \psi_1|_2 + \psi_1|_4 + \psi_1|_6\right), \tag{D.3d}$$

$$\psi_2|_a = \frac{1}{4}\left(\psi_2|_5 + \psi_2|_6 + \psi_2|_9 + \psi_2|_{10}\right), \tag{D.3e}$$

$$\psi_2|_b = \frac{1}{4}\left(\psi_2|_7 + \psi_2|_8 + \psi_2|_{11} + \psi_2|_{12}\right), \tag{D.3f}$$

$$\psi_2|_c = \frac{1}{4}\left(\psi_2|_3 + \psi_2|_4 + \psi_2|_7 + \psi_2|_8\right), \tag{D.3g}$$

$$\psi_2|_d = \frac{1}{4}\left(\psi_2|_1 + \psi_2|_2 + \psi_2|_5 + \psi_2|_6\right). \tag{D.3h}$$

For non-interior faces, one-sided differences are used for the derivatives and linear extrapolation is used instead of interpolation.

The primary $B$ face stencil is shown in Fig. D.2. This is very similar to the $A$ face stencil except it is oriented in the other coordinate direction. The derivatives are discretized:

$$\left.\frac{\partial \psi_1}{\partial \xi^1}\right|_B = \frac{1}{2}\left[\left(\psi_1|_6 + \psi_1|_8\right) - \left(\psi_1|_5 + \psi_1|_7\right)\right], \tag{D.4a}$$

$$\left.\frac{\partial \psi_1}{\partial \xi^2}\right|_B = \frac{1}{2}\left[\left(\psi_1|_7 + \psi_1|_8\right) - \left(\psi_1|_5 + \psi_1|_6\right)\right], \tag{D.4b}$$

$$\left.\frac{\partial \psi_1}{\partial \xi^3}\right|_B = \frac{1}{8}\left[\left(\psi_1|_9 + \psi_1|_{10} + \psi_1|_{11} + \psi_1|_{12}\right) - \left(\psi_1|_1 + \psi_1|_2 + \psi_1|_3 + \psi_1|_4\right)\right], \tag{D.4c}$$

$$\left.\frac{\partial \psi_2}{\partial \xi^1}\right|_B = \frac{1}{2}\left(\psi_2|_7 - \psi_2|_5\right), \tag{D.5a}$$

$$\left.\frac{\partial \psi_2}{\partial \xi^2}\right|_B = \frac{1}{2}\left(\psi_2|_8 - \psi_2|_4\right), \tag{D.5b}$$

$$\left.\frac{\partial \psi_2}{\partial \xi^3}\right|_B = \frac{1}{2}\left(\psi_2|_{10} - \psi_2|_2\right). \tag{D.5c}$$

Again, two sets of grid derivatives are evaluated for both the $\psi_1$ and $\psi_2$ gradients.

The face vertex values are interpolated:

$$\psi_1|_a = \frac{1}{4}\left(\psi_1|_1 + \psi_1|_3 + \psi_1|_5 + \psi_1|_7\right), \tag{D.6a}$$

$$\psi_1|_b = \frac{1}{4}\left(\psi_1|_2 + \psi_1|_4 + \psi_1|_6 + \psi_1|_8\right), \tag{D.6b}$$

$$\psi_1|_c = \frac{1}{4}\left(\psi_1|_6 + \psi_1|_8 + \psi_1|_{10} + \psi_1|_{12}\right), \tag{D.6c}$$

$$\psi_1|_d = \frac{1}{4}\left(\psi_1|_5 + \psi_1|_7 + \psi_1|_9 + \psi_1|_{11}\right), \tag{D.6d}$$

$$\psi_2|_a = \frac{1}{4}\left(\psi_2|_1 + \psi_2|_2 + \psi_2|_5 + \psi_2|_6\right), \tag{D.6e}$$

$$\psi_2|_b = \frac{1}{4}\left(\psi_2|_2 + \psi_2|_3 + \psi_2|_6 + \psi_2|_7\right), \tag{D.6f}$$

$$\psi_2|_c = \frac{1}{4}\left(\psi_2|_6 + \psi_2|_7 + \psi_2|_{10} + \psi_2|_{11}\right), \tag{D.6g}$$

$$\psi_2|_d = \frac{1}{4}\left(\psi_2|_5 + \psi_2|_6 + \psi_2|_9 + \psi_2|_{10}\right). \tag{D.6h}$$

The 2D stencils are shown in Fig. D.3 for an $S$ face, Fig. D.4 for an $A$ face, and Fig. D.5 for a $B$ face. For an $S$ face, the following formulas are used instead of those for an interior face if either $ni = 2$ or $nj = 2$ (refer to Fig. D.3):

$$\left.\frac{\partial \psi_1}{\partial \xi^2}\right|_S = 0. \tag{D.7}$$

$$\left.\frac{\partial \psi_2}{\partial \xi^1}\right|_S = 0. \tag{D.8}$$

$$\psi_1|_a = \frac{1}{2}\left(\psi_1|_3 + \psi_1|_9\right), \tag{D.9a}$$

$$\psi_1|_b = \frac{1}{2}\left(\psi_1|_3 + \psi_1|_9\right), \tag{D.9b}$$

$$\psi_1|_c = \frac{1}{2}\left(\psi_1|_4 + \psi_1|_{10}\right), \tag{D.9c}$$

$$\psi_1|_d = \frac{1}{2}\left(\psi_1|_4 + \psi_1|_{10}\right), \tag{D.9d}$$

$$\psi_2|_a = \frac{1}{2}\left(\psi_2|_2 + \psi_2|_8\right), \tag{D.9e}$$

$$\psi_2|_b = \frac{1}{2}\left(\psi_2|_5 + \psi_2|_{11}\right), \tag{D.9f}$$

$$\psi_2|_c = \frac{1}{2}\left(\psi_2|_5 + \psi_2|_{11}\right), \tag{D.9g}$$

$$\psi_2|_d = \frac{1}{2}\left(\psi_2|_2 + \psi_2|_8\right). \tag{D.9h}$$

For an $A$ face, the following formulas are used instead of those for an interior face if $nj = 2$ (refer to Fig. D.4):

$$\left.\frac{\partial \psi_1}{\partial \xi^2}\right|_A = 0. \tag{D.10}$$

$$\begin{aligned}
\psi_1|_a &= \frac{1}{2}(\psi_1|_6 + \psi_1|_{10}), && \text{(D.11a)}\\
\psi_1|_b &= \frac{1}{2}(\psi_1|_6 + \psi_1|_{10}), && \text{(D.11b)}\\
\psi_1|_c &= \frac{1}{2}(\psi_1|_2 + \psi_1|_6), && \text{(D.11c)}\\
\psi_1|_d &= \frac{1}{2}(\psi_1|_2 + \psi_1|_6). && \text{(D.11d)}
\end{aligned}$$

For a $B$ face, the following formulas are used instead of those for an interior face if $ni = 2$ (refer to Fig. D.5):

$$\left.\frac{\partial \psi_2}{\partial \xi^1}\right|_B = 0. \tag{D.12}$$

$$\begin{aligned}
\psi_2|_a &= \frac{1}{2}(\psi_2|_2 + \psi_2|_6), && \text{(D.13a)}\\
\psi_2|_b &= \frac{1}{2}(\psi_2|_2 + \psi_2|_6), && \text{(D.13b)}\\
\psi_2|_c &= \frac{1}{2}(\psi_2|_6 + \psi_2|_{10}), && \text{(D.13c)}\\
\psi_2|_d &= \frac{1}{2}(\psi_2|_6 + \psi_2|_{10}). && \text{(D.13d)}
\end{aligned}$$

Even though certain $\psi_1$ and $\psi_2$ derivatives are 0 for these 2D cases, the grid derivatives are not 0 and appropriate finite differences of these must be used.

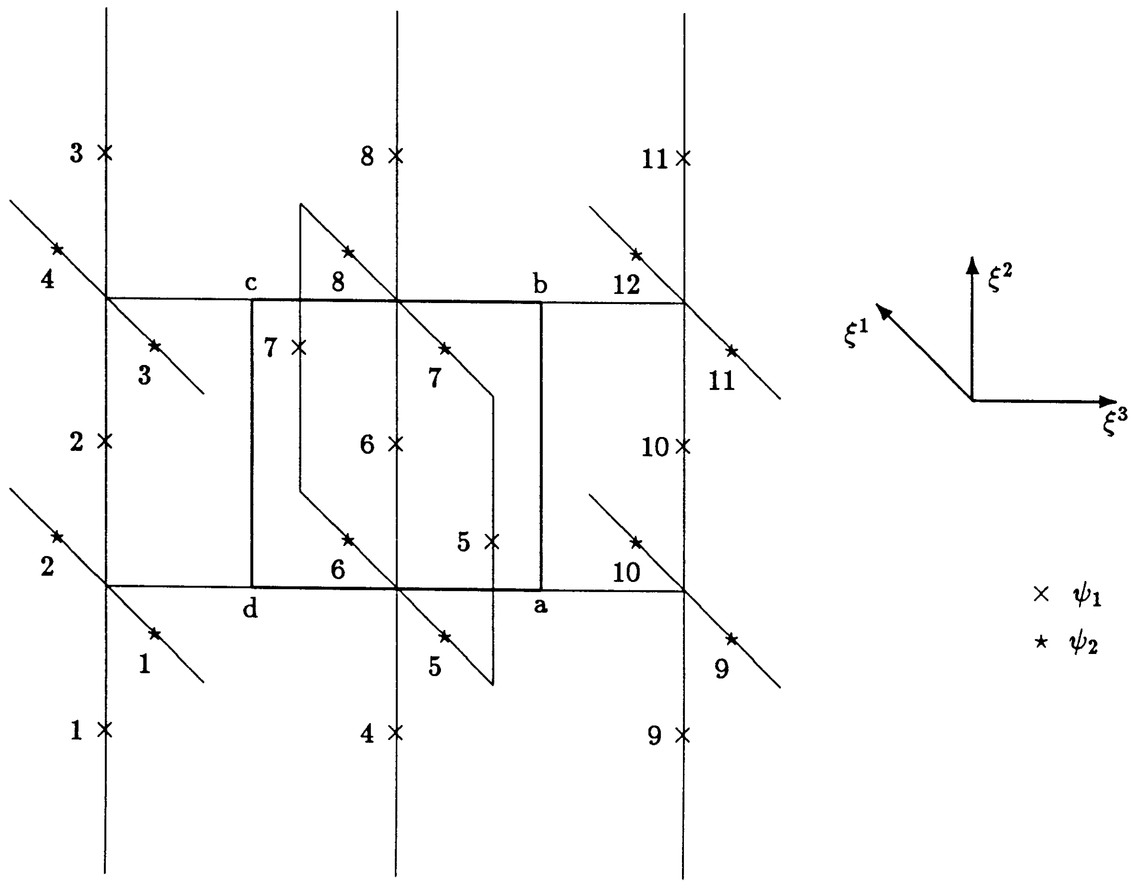


Figure D.1: Stencil for $\psi_1$ and $\psi_2$ on an interior $A$ face (type 0).

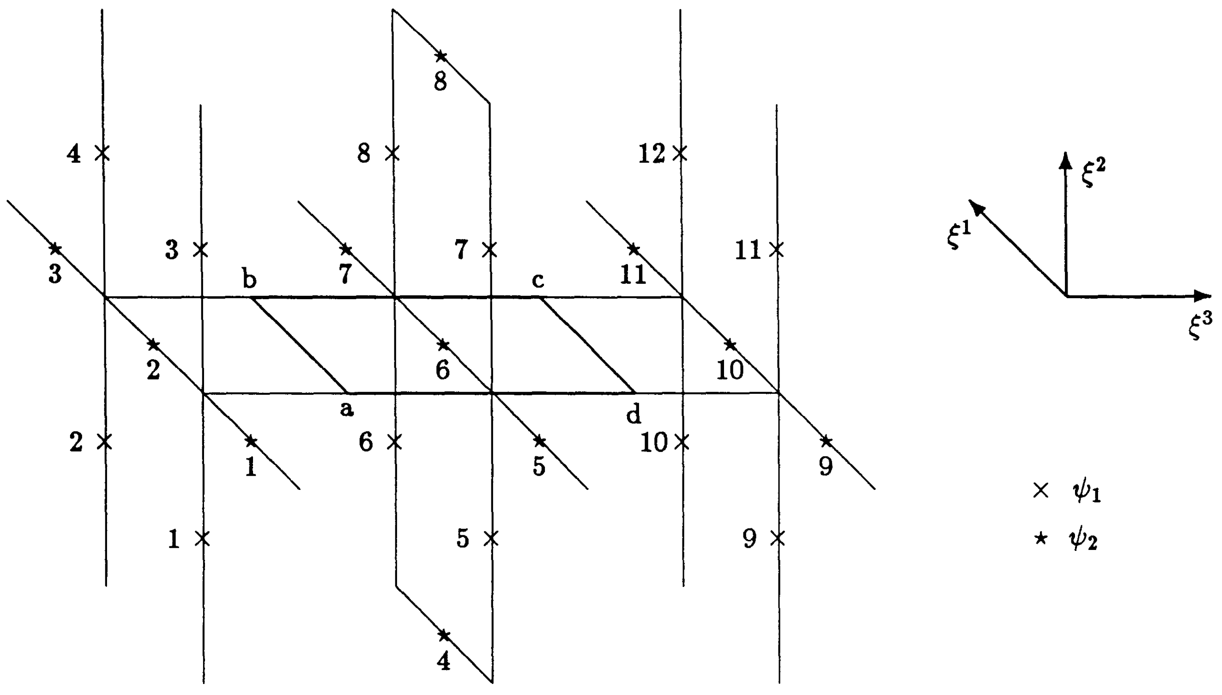


Figure D.2: Stencil for $\psi_1$ and $\psi_2$ on an interior $B$ face (type 0).

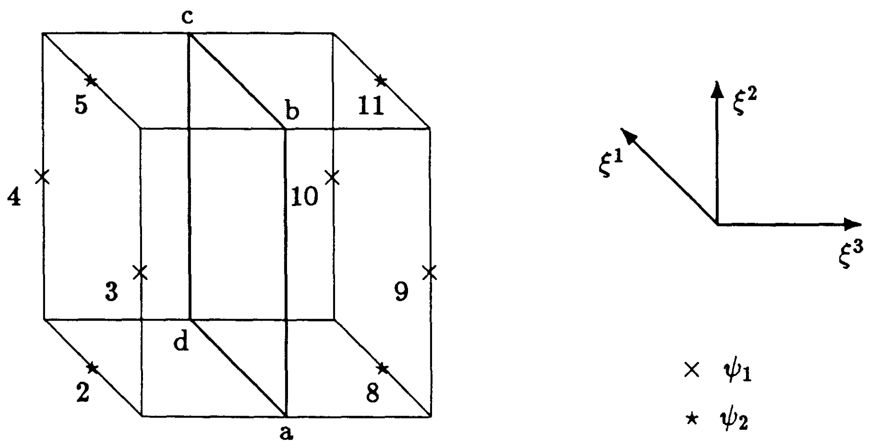


Figure D.3: Stencil for $\psi_1$ and $\psi_2$ on an $S$ face if either $ni = 2$ or $nj = 2$ (type 9).

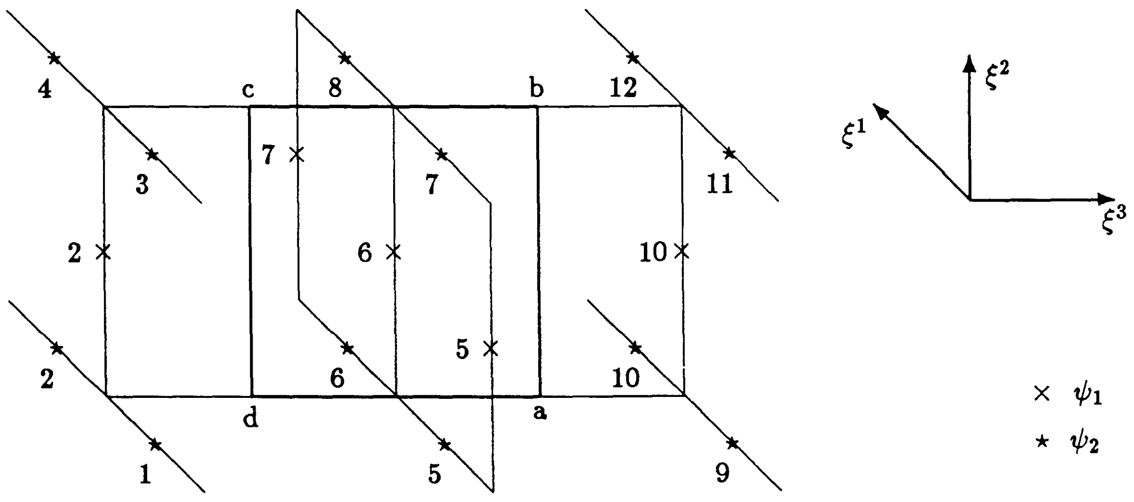


Figure D.4: Stencil for $\psi_1$ and $\psi_2$ on an $A$ face if $nj = 2$ (type 3).

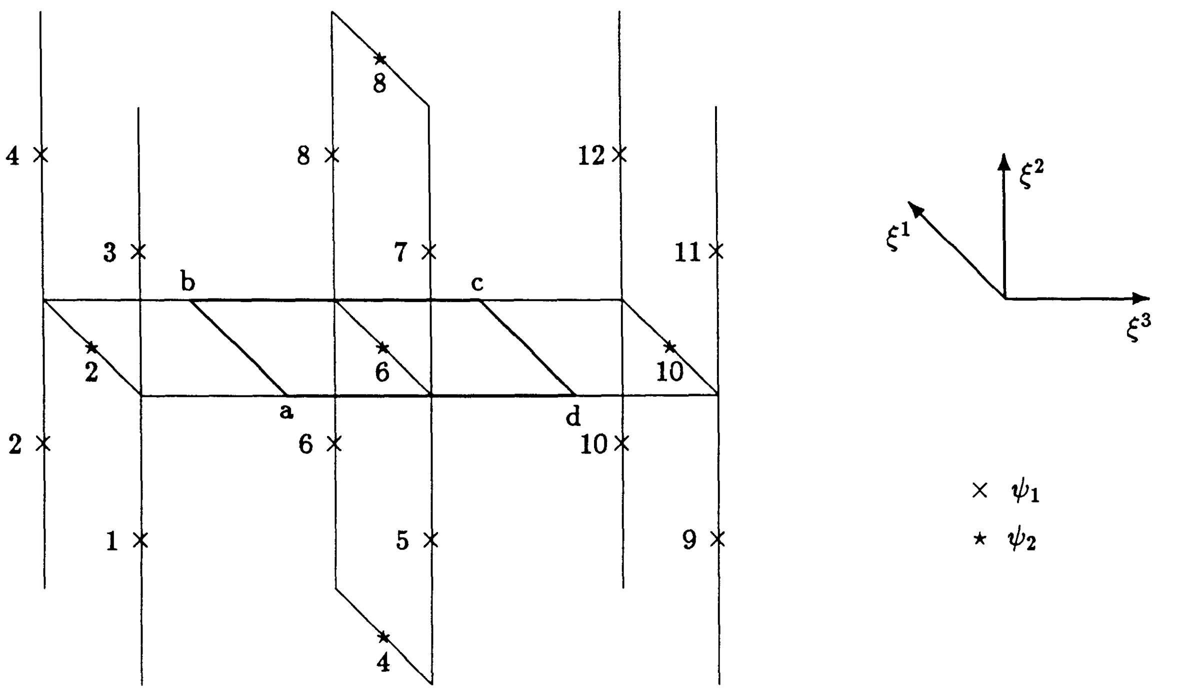


Figure D.5: Stencil for $\psi_1$ and $\psi_2$ on a $B$ face if $ni = 2$ (type 3).

# Appendix E
# SMP Input which Produces the Solver Subroutines

The following files are the inputs to SMP which produce the FORTRAN subroutines used in the solver. The main method of data transfer is through common blocks which are in FACE.FOR which is listed in Appendix F.

## File MA.IN

```
/* This session creates the subroutine which calculate the contribution
of an A face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy ma.sgf;* ma.osf
!dele ma.sgf;*
!@cgf ma.sgf

filename : "ma.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMA(IAF,FM)
C     Right Momentum equation for an A-face.
C
C     This routine calculates the contribution of the
C     A face IAF to the residuals
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IAF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMADEC.FOR'

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
```

```
      REAL AX,AY,AZ

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMA(IAF)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
fort["X component of pressure",'px,"ma.sgf"]

py : p ay
fort["Y component of pressure",'py,"ma.sgf"]

pz : p az
fort["Z component of pressure",'pz,"ma.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to residual.",filename]
Lpr["C       variables and return.",filename]

fx : px fm
fort["Face contribution to X-momentum eq.",'fx,"ma.sgf"]

fy : py fm
fort["Face contribution to Y-momentum eq.",'fy,"ma.sgf"]

fz : pz fm
fort["Face contribution to Z-momentum eq.",'fz,"ma.sgf"]

Lpr["
        RESX = RESX - FX
        RESY = RESY - FY
        RESZ = RESZ - FZ
        RETURN
      END IF

C     Non-wall A Faces.

C     Get the mass flux and the velocities.
      CALL RMASA(MAS)
      IF(PLANER)THEN
        CALL RVELP(1,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(1,RWX,RWY,RWZ)
      END IF
```

```
",filename]

/* Incompressible flow */
rho : roc
fort["density",'rho,"ma.sgf"]

fx : ( (mas rwx)/rho + px ) fm
fort["Face contribution to X-momentum eq.",'fx,"ma.sgf"]

fy : ( (mas rwy)/rho + py ) fm
fort["Face contribution to Y-momentum eq.",'fy,"ma.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
fort["Face contribution to Z-momentum eq.",'fz,"ma.sgf"]

Lpr["
      RESX = RESX - FX
      RESY = RESY - FY
      RESZ = RESZ - FZ

C     Save some of the calculated quantities.
      IF(ISAVE.EQ.1)CALL SAVEA(IAF,RWX,RWY,RWZ,RHO)

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy RMAdec.sgf;* RMAdec.osf
!dele RMAdec.sgf;*
!@cgf RMAdec.sgf

<"decfile.in"

decfile["RMAdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "ma.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMA(IAF,FM)
C     Left Momentum equation for an A-face.
C
C     This routine calculates the contribution of the
C     A face IAF to the left hand sides
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IAF
      REAL FM

      INCLUDE 'FACE.FOR'
```

```
C     These declarations are generated by SMP:
      INCLUDE 'LMADEC.FOR'
      REAL DERI
C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMA(IAF)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,ax,ay,az,mas,rwx,rwy,rwz}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_ax[Const] : 1 ; _ay[Const] : 1 ; _az[Const] : 1
_mas[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_gam[Const] : 1 ; _h[Const] : 1 ; _fm[Const] : 1 ; _or2[Const] : 1
_roc[Const] : 1

/* Algorithm */

Lpr["

C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
deriv["X component of pressure",'px,"ma.sgf"]

py : p ay
deriv["Y component of pressure",'py,"ma.sgf"]

pz : p az
deriv["Z component of pressure",'pz,"ma.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to matrix.",filename]
Lpr["C       and return.",filename]

fx : px fm
deriv["Face contribution to X-momentum eq.",'fx,"ma.sgf"]

fy : py fm
deriv["Face contribution to Y-momentum eq.",'fy,"ma.sgf"]

fz : pz fm
```

```
deriv["Face contribution to Z-momentum eq.",'fz,"ma.sgf"]

Lpr["
C
C       Put Jacobians in correct location in matrix.
C       X - Momentum Equation
C       NCX must be set already.
C
C       pa
        CALL ADCOFS(FXD1,IPAO+IP,COFX,ICOLX,NCX)
C
C       Y - Momentum Equation
C       NCY must be set already.
C
C       pa
        CALL ADCOFS(FYD1,IPAO+IP,COFY,ICOLY,NCY)
C
C       Z - Momentum Equation
C       NCZ must be set already.
C
C       pa
        CALL ADCOFS(FZD1,IPAO+IP,COFZ,ICOLZ,NCZ)
C

C       The dependence on the area normals.
        IF(PLANER)THEN
          CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        ELSE
          CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        END IF

        RETURN
      END IF

C     Non-wall A Faces.

C     Get the mass flux and the velocities.
      CALL RMASA(MAS)
      IF(PLANER)THEN
        CALL RVELP(1,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(1,RWX,RWY,RWZ)
      END IF

",filename]

/* Incompressible flow */
rho : roc
deriv["density",'rho,"ma.sgf"]

fx : ( (mas rwx)/rho + px ) fm
deriv["Face contribution to X-momentum eq.",'fx,"ma.sgf"]

fy : ( (mas rwy)/rho + py ) fm
deriv["Face contribution to Y-momentum eq.",'fy,"ma.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
deriv["Face contribution to Z-momentum eq.",'fz,"ma.sgf"]

/* Include in exprelist the dependence of fx, fy, and fz on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[fx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fz,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C     pa
      CALL ADCOFS(FXD1,IPAO+IP,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C     pa
      CALL ADCOFS(FYD1,IPAO+IP,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C     pa
      CALL ADCOFS(FZD1,IPAO+IP,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
C     The dependence on the area normals.
      IF(PLANER)THEN
        CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1            FZD2,FZD3,FZD4)
      ELSE
        CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1            FZD2,FZD3,FZD4)
      END IF

",filename]

Lpr["
C     The dependence on the mass flux.
      CALL LMASA(FXD5,FYD5,FZD5)

",filename]

Lpr["
C     The dependence on the velocities.
      IF(PLANER)THEN
        CALL LVELP(1,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
      ELSE
```

```
          CALL LVELC(1,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
        END IF

",filename]

Lpr["

        RETURN
        END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy LMAdec.sgf;* LMAdec.osf
!dele LMAdec.sgf;*
!@cgf LMAdec.sgf

<"decfile.in"

decfile["LMAdec.sgf"]
```

## File MB.IN

```
/* This session creates the subroutine which calculate the contribution
of an B face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mb.sgf;* mb.osf
!dele mb.sgf;*
!@cgf mb.sgf

filename : "mb.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMB(IBF,FM)
C     Right Momentum equation for a B-face.
C
C     This routine calculates the contribution of the
C     B face IBF to the residuals
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IBF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMBDEC.FOR'

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ
C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMB(IBF)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
fort["X component of pressure",'px,"mb.sgf"]

py : p ay
fort["Y component of pressure",'py,"mb.sgf"]

pz : p az
fort["Z component of pressure",'pz,"mb.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to residual.",filename]
Lpr["C       variables and return.",filename]

fx : px fm
fort["Face contribution to X-momentum eq.",'fx,"mb.sgf"]

fy : py fm
fort["Face contribution to Y-momentum eq.",'fy,"mb.sgf"]

fz : pz fm
fort["Face contribution to Z-momentum eq.",'fz,"mb.sgf"]

Lpr["
        RESX = RESX - FX
        RESY = RESY - FY
        RESZ = RESZ - FZ
        RETURN
      END IF

C     Non-wall A Faces.

C     Get the mass flux and the velocities.
      CALL RMASB(MAS)
      IF(PLANER)THEN
        CALL RVELP(2,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(2,RWX,RWY,RWZ)
      END IF

",filename]
```

```
/* Incompressible flow */
rho : roc
fort["density",'rho,"mb.sgf"]

fx : ( (mas rwx)/rho + px ) fm
fort["Face contribution to X-momentum eq.",'fx,"mb.sgf"]

fy : ( (mas rwy)/rho + py ) fm
fort["Face contribution to Y-momentum eq.",'fy,"mb.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
fort["Face contribution to Z-momentum eq.",'fz,"mb.sgf"]

Lpr["
      RESX = RESX - FX
      RESY = RESY - FY
      RESZ = RESZ - FZ

C     Save some of the calculated quantities.
      IF(ISAVE.EQ.1)CALL SAVEB(IBF,RWX,RWY,RWZ,RHO)

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmbdec.sgf;* Rmbdec.osf
!dele Rmbdec.sgf;*
!@cgf Rmbdec.sgf

<"decfile.in"

decfile["Rmbdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mb.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMB(IBF,FM)
C     Left Momentum equation for a B-face.
C
C     This routine calculates the contribution of the
C     B face IBF to the left hand sides
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IBF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMBDEC.FOR'
      REAL DERI
C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMB(IBF)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,ax,ay,az,mas,rwx,rwy,rwz}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_ax[Const] : 1 ; _ay[Const] : 1 ; _az[Const] : 1
_mas[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_gam[Const] : 1 ; _h[Const] : 1 ; _fm[Const] : 1 ; _or2[Const] : 1
_roc[Const] : 1

/* Algorithm */

Lpr["

C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
deriv["X component of pressure",'px,"mb.sgf"]

py : p ay
deriv["Y component of pressure",'py,"mb.sgf"]

pz : p az
deriv["Z component of pressure",'pz,"mb.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to matrix.",filename]
Lpr["C       and return.",filename]

fx : px fm
deriv["Face contribution to X-momentum eq.",'fx,"mb.sgf"]

fy : py fm
deriv["Face contribution to Y-momentum eq.",'fy,"mb.sgf"]

fz : pz fm
deriv["Face contribution to Z-momentum eq.",'fz,"mb.sgf"]
```

```
Lpr["
C
C       Put Jacobians in correct location in matrix.
C       X - Momentum Equation
C       NCX must be set already.
C
C       pb
        CALL ADCOFS(FXD1,IPBO+IP,COFX,ICOLX,NCX)
C
C       Y - Momentum Equation
C       NCY must be set already.
C
C       pb
        CALL ADCOFS(FYD1,IPBO+IP,COFY,ICOLY,NCY)
C
C       Z - Momentum Equation
C       NCZ must be set already.
C
C       pb
        CALL ADCOFS(FZD1,IPBO+IP,COFZ,ICOLZ,NCZ)
C

C       The dependence on the area normals.
        IF(PLANER)THEN
          CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        ELSE
          CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        END IF

        RETURN
      END IF

C     Non-wall B Faces.

C     Get the mass flux and the velocities.
      CALL RMASB(MAS)
      IF(PLANER)THEN
        CALL RVELP(2,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(2,RWX,RWY,RWZ)
      END IF

",filename]

/* Incompressible flow */
rho : roc
deriv["density",'rho,"mb.sgf"]

fx : ( (mas rwx)/rho + px ) fm
deriv["Face contribution to X-momentum eq.",'fx,"mb.sgf"]

fy : ( (mas rwy)/rho + py ) fm
deriv["Face contribution to Y-momentum eq.",'fy,"mb.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
deriv["Face contribution to Z-momentum eq.",'fz,"mb.sgf"]

/* Include in exprelist the dependence of fx, fy, and fz on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[fx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fz,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      X - Momentum Equation
C      NCX must be set already.
C
C      pb
       CALL ADCOFS(FXD1,IPBO+IP,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      Y - Momentum Equation
C      NCY must be set already.
C
C      pb
       CALL ADCOFS(FYD1,IPBO+IP,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      Z - Momentum Equation
C      NCZ must be set already.
C
C      pb
       CALL ADCOFS(FZD1,IPBO+IP,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
C      The dependence on the area normals.
       IF(PLANER)THEN
         CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
      1            FZD2,FZD3,FZD4)
       ELSE
         CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
      1            FZD2,FZD3,FZD4)
       END IF

",filename]

Lpr["
C      The dependence on the mass flux.
       CALL LMASB(FXD5,FYD5,FZD5)

",filename]

Lpr["
C      The dependence on the velocities.
       IF(PLANER)THEN
         CALL LVELP(2,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
       ELSE
         CALL LVELC(2,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
       END IF
```

```
",filename]

Lpr["

      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmbdec.sgf;* Lmbdec.osf
!dele Lmbdec.sgf;*
!@cgf Lmbdec.sgf

<"decfile.in"

decfile["Lmbdec.sgf"]
```

## File MS.IN

```
/* This session creates the subroutine which calculate the contribution
of an S face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy ms.sgf;* ms.osf
!dele ms.sgf;*
!@cgf ms.sgf

filename : "ms.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMS(ISF,FM)
C     Right Momentum equation for an S-face.
C
C     This routine calculates the contribution of the
C     S face ISF to the residuals
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER ISF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMSDEC.FOR'

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ
C     Declare the volume variable.
      REAL VOL
C     Declare the r and theta at the face center.
```

```
      REAL R,X1

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the mass flux and the velocities.
      CALL RMASS(MAS)
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

Lpr["

C     Get the area normals and volume.  The volume is set to 0 if omega is 0.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
        IF(OMG.EQ.O.)THEN
          VOL = 0.
        ELSE
          CALL RVOLP(VOL)
        END IF
      ELSE
        CALL RANC(AX,AY,AZ)
        IF(OMG.EQ.O.)THEN
          VOL = 0.
        ELSE
          CALL RVOLC(VOL)
        END IF
      END IF

C     Get the r and theta.  If planer, the routine returns 0 for both.
      CALL RRTHFC(R,X1)

",filename]

/* Incompressible flow */
rho : roc
fort["density",'rho,"ms.sgf"]

pb : p
fort["upwind pressure",'pb,"ms.sgf"]

Lpr["C     Pressure terms.",filename]

px : pb ax
fort["X component of pressure",'px,"ms.sgf"]

py : pb ay
fort["Y component of pressure",'py,"ms.sgf"]

pz : pb az
fort["Z component of pressure",'pz,"ms.sgf"]
```

```
Lpr["C      Source terms.",filename]

x : r Sin[x1]
fort["x at face",'x,"ms.sgf"]

y : r Cos[x1]
fort["y at face",'y,"ms.sgf"]

sx : vol (-rho omg^2 x - 2 omg rwy)/2
fort["X component of rotating source term",'sx,"ms.sgf"]

sy : vol (-rho omg^2 y + 2 omg rwx)/2
fort["Y component of rotating source term",'sy,"ms.sgf"]

Lpr["C      Face contibutions to momentum eqs.",filename]

fx : ( (mas rwx)/rho + px ) fm + sx
fort["Face contribution to X-momentum eq.",'fx,"ms.sgf"]

fy : ( (mas rwy)/rho + py ) fm + sy
fort["Face contribution to Y-momentum eq.",'fy,"ms.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
fort["Face contribution to Z-momentum eq.",'fz,"ms.sgf"]

Lpr["
      RESX = RESX - FX
      RESY = RESY - FY
      RESZ = RESZ - FZ

C     Save some of the calculated quantities.
      IF(ISAVE.EQ.1)CALL SAVES(ISF,RWX,RWY,RWZ,RHO)

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmsdec.sgf;* Rmsdec.osf
!dele Rmsdec.sgf;*
!@cgf Rmsdec.sgf

<"decfile.in"

decfile["Rmsdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "ms.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMS(ISF,FM)
C     Left Momentum equation for an S-face.
```

```
C
C     This routine calculates the contribution of the
C     S face ISF to the left hand sides
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER ISF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMSDEC.FOR'
      REAL DERI

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ
C     Declare the volume variable.
      REAL VOL
C     Declare the r and theta at the face center.
      REAL R,X1

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,ax,ay,az,mas,rwx,rwy,rwz,r,x1,vol}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_ax[Const] : 1 ; _ay[Const] : 1 ; _az[Const] : 1
_mas[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_gam[Const] : 1 ; _omg[Const] : 1 ; _fm[Const] : 1
_roc[Const] : 1
_vol[Const] : 1
_r[Const] : 1 ; _x1[Const] : 1
_vsc[Const] : 1 ; _mcs[Const] : 1 ;
_p1[Const] : 1 ; _m1s[Const] : 1 ; _p0[Const] : 1

/* Algorithm */

Lpr["

C     Get the mass flux and the velocities.
      CALL RMASS(MAS)
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

Lpr["
```

```
C     Get the area normals and volume.  The volume is set to 0 if omega is 0.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
        IF(OMG.EQ.O.)THEN
          VOL = 0.
        ELSE
          CALL RVOLP(VOL)
        END IF
      ELSE
        CALL RANC(AX,AY,AZ)
        IF(OMG.EQ.O.)THEN
          VOL = 0.
        ELSE
          CALL RVOLC(VOL)
        END IF
      END IF

C     Get the r and theta.  If planer, the routine returns 0 for both.
      CALL RRTHFC(R,X1)

",filename]

/* Incompressible flow */
rho : roc
deriv["density",'rho,"ms.sgf"]

pb : p
deriv["upwind pressure",'pb,"ms.sgf"]

Lpr["C     Pressure terms.",filename]

px : pb ax
deriv["X component of pressure",'px,"ms.sgf"]

py : pb ay
deriv["Y component of pressure",'py,"ms.sgf"]

pz : pb az
deriv["Z component of pressure",'pz,"ms.sgf"]

Lpr["C     Source terms.",filename]

x : r Sin[x1]
deriv["x at face",'x,"ms.sgf"]

y : r Cos[x1]
deriv["y at face",'y,"ms.sgf"]

sx : vol (-rho omg^2 x - 2 omg rwy)/2
deriv["X component of rotating source term",'sx,"ms.sgf"]

sy : vol (-rho omg^2 y + 2 omg rwx)/2
deriv["Y component of rotating source term",'sy,"ms.sgf"]

Lpr["C     Face contibutions to momentum eqs.",filename]

fx : ( (mas rwx)/rho + px ) fm + sx
deriv["Face contribution to X-momentum eq.",'fx,"ms.sgf"]

fy : ( (mas rwy)/rho + py ) fm + sy
deriv["Face contribution to Y-momentum eq.",'fy,"ms.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
deriv["Face contribution to Z-momentum eq.",'fz,"ms.sgf"]

/* Include in exprelist the dependence of fx, fy, and fz on all the
```

```
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[fx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fz,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      X - Momentum Equation
C      NCX must be set already.
C
C      ps
       CALL ADCOFS(FXD1,IPSO+IP,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      Y - Momentum Equation
C      NCY must be set already.
C
C      ps
       CALL ADCOFS(FYD1,IPSO+IP,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      Z - Momentum Equation
C      NCZ must be set already.
C
C      ps
       CALL ADCOFS(FZD1,IPSO+IP,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
C      The dependence on the area normals and volume.
       IF(PLANER)THEN
         CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1             FZD2,FZD3,FZD4)
         IF(OMG.NE.O.)CALL LVOLP(FXD11,FYD11,FZD11)
       ELSE
         CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1             FZD2,FZD3,FZD4)
         IF(OMG.NE.O.)CALL LVOLC(FXD11,FYD11,FZD11)
       END IF

",filename]

Lpr["
C      The dependence on the mass flux.
       CALL LMASS(FXD5,FYD5,FZD5)

",filename]
```

```
Lpr["
C     The dependence on the velocities.
      IF(PLANER)THEN
        CALL LVELP(3,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
      ELSE
        CALL LVELC(3,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
      END IF
",filename]

Lpr["
C     The dependence on r and theta.
      CALL LRTHFC(FXD9,FXD10,FYD9,FYD10,FZD9,FZD10)

",filename]

Lpr["


      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmsdec.sgf;* Lmsdec.osf
!dele Lmsdec.sgf;*
!@cgf Lmsdec.sgf

<"decfile.in"

decfile["Lmsdec.sgf"]
```

## File MASA.IN

```
/* This session creates the subroutine which calculates the contribution
of the mass flux for an A face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy masa.sgf;* masa.osf
!dele masa.sgf;*
!@cgf masa.sgf

filename : "masa.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMASA(MASA)
C     Right MASs flux calculation for an A-face.
C
C     This routine calculates the mass flux on an A face
C     This routine is called by RMA or LMA so the variables
C     in common are already initialized.
C
      REAL MASA
      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMASADEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

aacase : {3/4 (a6 + a10) - 1/4 (a8 + a11),\
          (a6 + a10)/2,\
          (a4 + a6 + a9 + a10)/4}
aacond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi1 at a",aa,aacase,aacond,"masa.sgf"]

abcase : {3/4 (a6 + a10) - 1/4 (a4 + a9),\
          (a6 + a10)/2,\
          (a6 + a8 + a10 + a11)/4}
abcond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi1 at b",ab,abcase,abcond,"masa.sgf"]

accase : {3/4 (a2 + a6) - 1/4 (a1 + a4),\
          (a2 + a6)/2,\
          (a2 + a3 + a6 + a8)/4}
accond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi1 at c",ac,accase,accond,"masa.sgf"]

adcase : {3/4 (a2 + a6) - 1/4 (a3 + a8),\
          (a2 + a6)/2,\
          (a1 + a2 + a4 + a6)/4}
adcond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi1 at d",ad,adcase,adcond,"masa.sgf"]

ba : (b5 + b6 + b9 + b10)/4
fort["Psi2 at a",'ba,"masa.sgf"]

bb : (b7 + b8 + b11 + b12)/4
fort["Psi2 at b",'bb,"masa.sgf"]

bc : (b3 + b4 + b7 + b8)/4
fort["Psi2 at c",'bc,"masa.sgf"]

bd : (b1 + b2 + b5 + b6)/4
fort["Psi2 at d",'bd,"masa.sgf"]

mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
fort["Mass flux",'mas,"masa.sgf"]

Lpr["
      MASA = MAS

C
      RETURN
      END
```

```
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy RMASAdec.sgf;* RMASAdec.osf
!dele RMASAdec.sgf;*
!@cgf RMASAdec.sgf

<"decfile.in"

decfile["RMASAdec.sgf"]

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy LMASAdec.sgf;* LMASAdec.osf
!dele LMASAdec.sgf;*
!@cgf LMASAdec.sgf

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "masa.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMASA(DFXDM,DFYDM,DFZDM)
C     Left MASs flux calculation for an A-face.
C
C     This routine calculates contribution of the mass flux on
C     an A face to the x, y, and z momentum equations.
C     This routine is called by LMA so the variables
C     in common are already initialized.
C
      REAL DFXDM,DFYDM,DFZDM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMASADEC.FOR'
      REAL DERI

",filename]


/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,\
  b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_a1[Const] : 1 ; _a2[Const] : 1 ; _a3[Const] : 1 ; _a4[Const] : 1
_a5[Const] : 1 ; _a6[Const] : 1 ; _a7[Const] : 1 ; _a8[Const] : 1
_a9[Const] : 1 ; _a10[Const] : 1 ; _a11[Const] : 1 ; _a12[Const] : 1
_b1[Const] : 1 ; _b2[Const] : 1 ; _b3[Const] : 1 ; _b4[Const] : 1
_b5[Const] : 1 ; _b6[Const] : 1 ; _b7[Const] : 1 ; _b8[Const] : 1
_b9[Const] : 1 ; _b10[Const] : 1 ; _b11[Const] : 1 ; _b12[Const] : 1
```

```
/* Algorithm */

aacase : {3/4 (a6 + a10) - 1/4 (a8 + a11),\
          (a6 + a10)/2,\
          (a4 + a6 + a9 + a10)/4}
aacond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi1 at a",aa,aacase,aacond,"masa.sgf"]

abcase : {3/4 (a6 + a10) - 1/4 (a4 + a9),\
          (a6 + a10)/2,\
          (a6 + a8 + a10 + a11)/4}
abcond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi1 at b",ab,abcase,abcond,"masa.sgf"]

accase : {3/4 (a2 + a6) - 1/4 (a1 + a4),\
          (a2 + a6)/2,\
          (a2 + a3 + a6 + a8)/4}
accond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi1 at c",ac,accase,accond,"masa.sgf"]

adcase : {3/4 (a2 + a6) - 1/4 (a3 + a8),\
          (a2 + a6)/2,\
          (a1 + a2 + a4 + a6)/4}
adcond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi1 at d",ad,adcase,adcond,"masa.sgf"]

ba : (b5 + b6 + b9 + b10)/4
deriv["Psi2 at a",'ba,"masa.sgf"]

bb : (b7 + b8 + b11 + b12)/4
deriv["Psi2 at b",'bb,"masa.sgf"]

bc : (b3 + b4 + b7 + b8)/4
deriv["Psi2 at c",'bc,"masa.sgf"]

bd : (b1 + b2 + b5 + b6)/4
deriv["Psi2 at d",'bd,"masa.sgf"]

mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
deriv["Mass flux",'mas,"masa.sgf"]

/* Include in exprelist the dependence of mas on all the dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[mas,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
```

```
C     Psi1 at 1
      CALL ADCOFS(DFXDM*MASD1,IA1+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 2
      CALL ADCOFS(DFXDM*MASD2,IA2+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 3
      CALL ADCOFS(DFXDM*MASD3,IA3+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 4
      CALL ADCOFS(DFXDM*MASD4,IA4+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 5
      CALL ADCOFS(DFXDM*MASD5,IA5+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 6
      CALL ADCOFS(DFXDM*MASD6,IA6+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 7
      CALL ADCOFS(DFXDM*MASD7,IA7+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 8
      CALL ADCOFS(DFXDM*MASD8,IA8+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 9
      CALL ADCOFS(DFXDM*MASD9,IA9+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 10
      CALL ADCOFS(DFXDM*MASD10,IA10+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 11
      CALL ADCOFS(DFXDM*MASD11,IA11+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 12
      CALL ADCOFS(DFXDM*MASD12,IA12+IPSI10,COFX,ICOLX,NCX)

C
C     Psi2 at 1
      CALL ADCOFS(DFXDM*MASD13,IB1+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 2
      CALL ADCOFS(DFXDM*MASD14,IB2+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 3
      CALL ADCOFS(DFXDM*MASD15,IB3+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 4
      CALL ADCOFS(DFXDM*MASD16,IB4+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 5
      CALL ADCOFS(DFXDM*MASD17,IB5+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 6
      CALL ADCOFS(DFXDM*MASD18,IB6+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 7
      CALL ADCOFS(DFXDM*MASD19,IB7+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 8
      CALL ADCOFS(DFXDM*MASD20,IB8+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 9
      CALL ADCOFS(DFXDM*MASD21,IB9+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 10
      CALL ADCOFS(DFXDM*MASD22,IB10+IPSI20,COFX,ICOLX,NCX)
```

```
C
C     Psi2 at 11
      CALL ADCOFS(DFXDM*MASD23,IB11+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 12
      CALL ADCOFS(DFXDM*MASD24,IB12+IPSI20,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C


C
C     Psi1 at 1
      CALL ADCOFS(DFYDM*MASD1,IA1+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 2
      CALL ADCOFS(DFYDM*MASD2,IA2+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 3
      CALL ADCOFS(DFYDM*MASD3,IA3+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 4
      CALL ADCOFS(DFYDM*MASD4,IA4+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 5
      CALL ADCOFS(DFYDM*MASD5,IA5+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 6
      CALL ADCOFS(DFYDM*MASD6,IA6+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 7
      CALL ADCOFS(DFYDM*MASD7,IA7+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 8
      CALL ADCOFS(DFYDM*MASD8,IA8+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 9
      CALL ADCOFS(DFYDM*MASD9,IA9+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 10
      CALL ADCOFS(DFYDM*MASD10,IA10+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 11
      CALL ADCOFS(DFYDM*MASD11,IA11+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 12
      CALL ADCOFS(DFYDM*MASD12,IA12+IPSI10,COFY,ICOLY,NCY)

C
C     Psi2 at 1
      CALL ADCOFS(DFYDM*MASD13,IB1+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 2
      CALL ADCOFS(DFYDM*MASD14,IB2+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 3
      CALL ADCOFS(DFYDM*MASD15,IB3+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 4
```

```
      CALL ADCOFS(DFYDM*MASD16,IB4+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 5
      CALL ADCOFS(DFYDM*MASD17,IB5+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 6
      CALL ADCOFS(DFYDM*MASD18,IB6+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 7
      CALL ADCOFS(DFYDM*MASD19,IB7+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 8
      CALL ADCOFS(DFYDM*MASD20,IB8+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 9
      CALL ADCOFS(DFYDM*MASD21,IB9+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 10
      CALL ADCOFS(DFYDM*MASD22,IB10+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 11
      CALL ADCOFS(DFYDM*MASD23,IB11+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 12
      CALL ADCOFS(DFYDM*MASD24,IB12+IPSI20,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi1 at 1
      CALL ADCOFS(DFZDM*MASD1,IA1+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 2
      CALL ADCOFS(DFZDM*MASD2,IA2+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 3
      CALL ADCOFS(DFZDM*MASD3,IA3+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 4
      CALL ADCOFS(DFZDM*MASD4,IA4+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 5
      CALL ADCOFS(DFZDM*MASD5,IA5+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 6
      CALL ADCOFS(DFZDM*MASD6,IA6+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 7
      CALL ADCOFS(DFZDM*MASD7,IA7+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 8
      CALL ADCOFS(DFZDM*MASD8,IA8+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 9
      CALL ADCOFS(DFZDM*MASD9,IA9+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 10
```

```
      CALL ADCOFS(DFZDM*MASD10,IA10+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 11
      CALL ADCOFS(DFZDM*MASD11,IA11+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 12
      CALL ADCOFS(DFZDM*MASD12,IA12+IPSI10,COFZ,ICOLZ,NCZ)

C
C     Psi2 at 1
      CALL ADCOFS(DFZDM*MASD13,IB1+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 2
      CALL ADCOFS(DFZDM*MASD14,IB2+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 3
      CALL ADCOFS(DFZDM*MASD15,IB3+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 4
      CALL ADCOFS(DFZDM*MASD16,IB4+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 5
      CALL ADCOFS(DFZDM*MASD17,IB5+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 6
      CALL ADCOFS(DFZDM*MASD18,IB6+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 7
      CALL ADCOFS(DFZDM*MASD19,IB7+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 8
      CALL ADCOFS(DFZDM*MASD20,IB8+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 9
      CALL ADCOFS(DFZDM*MASD21,IB9+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 10
      CALL ADCOFS(DFZDM*MASD22,IB10+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 11
      CALL ADCOFS(DFZDM*MASD23,IB11+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 12
      CALL ADCOFS(DFZDM*MASD24,IB12+IPSI20,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["LMASAdec.sgf"]
```

## File RMASB.IN

```
/* This session creates the subroutine which calculates the contribution
of the mass flux for an B face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum. */
```

```
/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy rmasb.sgf;* rmasb.osf
!dele rmasb.sgf;*
!@cgf rmasb.sgf

filename : "rmasb.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMASB(MASB)
C     Right MASs flux calculation for a B-face.
C
C     This routine calculates the mass flux on an B face
C     This routine is called by RMB or LMB so the variables
C     in common are already initialized.
C
      REAL MASB

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMASBDEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

aa : (a1 + a3 + a5 + a7)/4
fort["Psi1 at a",'aa,"rmasb.sgf"]

ab : (a2 + a4 + a6 + a8)/4
fort["Psi1 at b",'ab,"rmasb.sgf"]

ac : (a6 + a8 + a10 + a12)/4
fort["Psi1 at c",'ac,"rmasb.sgf"]

ad : (a5 + a7 + a9 + a11)/4
fort["Psi1 at d",'ad,"rmasb.sgf"]

bacase : {3/4 (b2 + b6) - 1/4 (b3 + b7),\
          (b2 + b6)/2,\
          (b1 + b2 + b5 + b6)/4}
bacond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi2 at a",ba,bacase,bacond,"rmasb.sgf"]

bbcase : {3/4 (b2 + b6) - 1/4 (b1 + b5),\
          (b2 + b6)/2,\
          (b2 + b3 + b6 + b7)/4}
bbcond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi2 at b",bb,bbcase,bbcond,"rmasb.sgf"]

bccase : {3/4 (b6 + b10) - 1/4 (b5 + b9),\
          (b6 + b10)/2,\
          (b6 + b7 + b10 + b11)/4}
bccond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi2 at c",bc,bccase,bccond,"rmasb.sgf"]

bdcase : {3/4 (b6 + b10) - 1/4 (b7 + b11),\
          (b6 + b10)/2,\
          (b5 + b6 + b9 + b10)/4}
bdcond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casefort[3,"Psi2 at d",bd,bdcase,bdcond,"rmasb.sgf"]

mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
fort["Mass flux",'mas,"rmasb.sgf"]

Lpr["
      MASB = MAS

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmasbdec.sgf;* Rmasbdec.osf
!dele Rmasbdec.sgf;*
!@cgf Rmasbdec.sgf

<"decfile.in"

decfile["Rmasbdec.sgf"]
```

## File LMASB.IN

```
/* This session creates the subroutine which calculates the left hand side
contribution of the mass flux for an B face to the residual right hand side
for each of the equations:  x-momentum, y-momentum, and z momentum.*/

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy lmasb.sgf;* lmasb.osf
!dele lmasb.sgf;*
!@cgf lmasb.sgf

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmasbdec.sgf;* Lmasbdec.osf
!dele Lmasbdec.sgf;*
!@cgf Lmasbdec.sgf

filename : "lmasb.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMASB(DFXDM,DFYDM,DFZDM)
C     Left MASs flux calculation for an B-face.
C
C     This routine calculates contribution of the mass flux on
C     a B face to the x, y, and z momentum equations.
C     This routine is called by LMB so the variables
C     in common are already initialized.
C
      REAL DFXDM,DFYDM,DFZDM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMASBDEC.FOR'
      REAL DERI

",filename]


/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,\
  b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_a1[Const] : 1 ; _a2[Const] : 1 ; _a3[Const] : 1 ; _a4[Const] : 1
_a5[Const] : 1 ; _a6[Const] : 1 ; _a7[Const] : 1 ; _a8[Const] : 1
_a9[Const] : 1 ; _a10[Const] : 1 ; _a11[Const] : 1 ; _a12[Const] : 1
_b1[Const] : 1 ; _b2[Const] : 1 ; _b3[Const] : 1 ; _b4[Const] : 1
_b5[Const] : 1 ; _b6[Const] : 1 ; _b7[Const] : 1 ; _b8[Const] : 1
_b9[Const] : 1 ; _b10[Const] : 1 ; _b11[Const] : 1 ; _b12[Const] : 1

/* Algorithm */

aa : (a1 + a3 + a5 + a7)/4
deriv["Psi1 at a",'aa,"lmasb.sgf"]

ab : (a2 + a4 + a6 + a8)/4
deriv["Psi1 at b",'ab,"lmasb.sgf"]

ac : (a6 + a8 + a10 + a12)/4
deriv["Psi1 at c",'ac,"lmasb.sgf"]

ad : (a5 + a7 + a9 + a11)/4
deriv["Psi1 at d",'ad,"lmasb.sgf"]

bacase : {3/4 (b2 + b6) - 1/4 (b3 + b7),\
          (b2 + b6)/2,\
          (b1 + b2 + b5 + b6)/4}
bacond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi2 at a",ba,bacase,bacond,"lmasb.sgf"]

bbcase : {3/4 (b2 + b6) - 1/4 (b1 + b5),\
          (b2 + b6)/2,\
          (b2 + b3 + b6 + b7)/4}
bbcond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi2 at b",bb,bbcase,bbcond,"lmasb.sgf"]

bccase : {3/4 (b6 + b10) - 1/4 (b5 + b9),\
          (b6 + b10)/2,\
          (b6 + b7 + b10 + b11)/4}
```

```
bccond : {"IFTYP.EQ.2",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi2 at c",bc,bccase,bccond,"lmasb.sgf"]

bdcase : {3/4 (b6 + b10) - 1/4 (b7 + b11),\
          (b6 + b10)/2,\
          (b5 + b6 + b9 + b10)/4}
bdcond : {"IFTYP.EQ.1",\
          "IFTYP.EQ.3",\
          ".TRUE."}
casederiv[3,"Psi2 at d",bd,bdcase,bdcond,"lmasb.sgf"]

mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
deriv["Mass flux",'mas,"lmasb.sgf"]

/* Include in exprelist the dependence of mas on all the dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[mas,dii] \
]

/* Memory managment */
Lpr[Mem[]]
exprlist : Union[exprlist]
nel : Len[exprlist]
Gc[]
Lpr[Mem[]]

/* X - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      X - Momentum Equation
C      NCX must be set already.
C

C
C      Psi1 at 1
       CALL ADCOFS(DFXDM*MASD1,IA1+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 2
       CALL ADCOFS(DFXDM*MASD2,IA2+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 3
       CALL ADCOFS(DFXDM*MASD3,IA3+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 4
       CALL ADCOFS(DFXDM*MASD4,IA4+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 5
       CALL ADCOFS(DFXDM*MASD5,IA5+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 6
       CALL ADCOFS(DFXDM*MASD6,IA6+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 7
       CALL ADCOFS(DFXDM*MASD7,IA7+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 8
       CALL ADCOFS(DFXDM*MASD8,IA8+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 9
       CALL ADCOFS(DFXDM*MASD9,IA9+IPSI10,COFX,ICOLX,NCX)
```

```
C
C     Psi1 at 10
      CALL ADCOFS(DFXDM*MASD10,IA10+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 11
      CALL ADCOFS(DFXDM*MASD11,IA11+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 12
      CALL ADCOFS(DFXDM*MASD12,IA12+IPSI10,COFX,ICOLX,NCX)

C
C     Psi2 at 1
      CALL ADCOFS(DFXDM*MASD13,IB1+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 2
      CALL ADCOFS(DFXDM*MASD14,IB2+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 3
      CALL ADCOFS(DFXDM*MASD15,IB3+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 4
      CALL ADCOFS(DFXDM*MASD16,IB4+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 5
      CALL ADCOFS(DFXDM*MASD17,IB5+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 6
      CALL ADCOFS(DFXDM*MASD18,IB6+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 7
      CALL ADCOFS(DFXDM*MASD19,IB7+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 8
      CALL ADCOFS(DFXDM*MASD20,IB8+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 9
      CALL ADCOFS(DFXDM*MASD21,IB9+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 10
      CALL ADCOFS(DFXDM*MASD22,IB10+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 11
      CALL ADCOFS(DFXDM*MASD23,IB11+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 12
      CALL ADCOFS(DFXDM*MASD24,IB12+IPSI20,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     Psi1 at 1
      CALL ADCOFS(DFYDM*MASD1,IA1+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 2
      CALL ADCOFS(DFYDM*MASD2,IA2+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 3
```

```
      CALL ADCOFS(DFYDM*MASD3,IA3+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 4
      CALL ADCOFS(DFYDM*MASD4,IA4+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 5
      CALL ADCOFS(DFYDM*MASD5,IA5+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 6
      CALL ADCOFS(DFYDM*MASD6,IA6+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 7
      CALL ADCOFS(DFYDM*MASD7,IA7+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 8
      CALL ADCOFS(DFYDM*MASD8,IA8+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 9
      CALL ADCOFS(DFYDM*MASD9,IA9+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 10
      CALL ADCOFS(DFYDM*MASD10,IA10+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 11
      CALL ADCOFS(DFYDM*MASD11,IA11+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 12
      CALL ADCOFS(DFYDM*MASD12,IA12+IPSI10,COFY,ICOLY,NCY)

C
C     Psi2 at 1
      CALL ADCOFS(DFYDM*MASD13,IB1+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 2
      CALL ADCOFS(DFYDM*MASD14,IB2+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 3
      CALL ADCOFS(DFYDM*MASD15,IB3+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 4
      CALL ADCOFS(DFYDM*MASD16,IB4+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 5
      CALL ADCOFS(DFYDM*MASD17,IB5+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 6
      CALL ADCOFS(DFYDM*MASD18,IB6+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 7
      CALL ADCOFS(DFYDM*MASD19,IB7+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 8
      CALL ADCOFS(DFYDM*MASD20,IB8+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 9
      CALL ADCOFS(DFYDM*MASD21,IB9+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 10
      CALL ADCOFS(DFYDM*MASD22,IB10+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 11
      CALL ADCOFS(DFYDM*MASD23,IB11+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 12
      CALL ADCOFS(DFYDM*MASD24,IB12+IPSI20,COFY,ICOLY,NCY)
```

```
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi1 at 1
      CALL ADCOFS(DFZDM*MASD1,IA1+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 2
      CALL ADCOFS(DFZDM*MASD2,IA2+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 3
      CALL ADCOFS(DFZDM*MASD3,IA3+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 4
      CALL ADCOFS(DFZDM*MASD4,IA4+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 5
      CALL ADCOFS(DFZDM*MASD5,IA5+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 6
      CALL ADCOFS(DFZDM*MASD6,IA6+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 7
      CALL ADCOFS(DFZDM*MASD7,IA7+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 8
      CALL ADCOFS(DFZDM*MASD8,IA8+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 9
      CALL ADCOFS(DFZDM*MASD9,IA9+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 10
      CALL ADCOFS(DFZDM*MASD10,IA10+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 11
      CALL ADCOFS(DFZDM*MASD11,IA11+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 12
      CALL ADCOFS(DFZDM*MASD12,IA12+IPSI10,COFZ,ICOLZ,NCZ)

C
C     Psi2 at 1
      CALL ADCOFS(DFZDM*MASD13,IB1+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 2
      CALL ADCOFS(DFZDM*MASD14,IB2+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 3
      CALL ADCOFS(DFZDM*MASD15,IB3+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 4
      CALL ADCOFS(DFZDM*MASD16,IB4+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 5
      CALL ADCOFS(DFZDM*MASD17,IB5+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 6
      CALL ADCOFS(DFZDM*MASD18,IB6+IPSI20,COFZ,ICOLZ,NCZ)
```

```
C
C     Psi2 at 7
      CALL ADCOFS(DFZDM*MASD19,IB7+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 8
      CALL ADCOFS(DFZDM*MASD20,IB8+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 9
      CALL ADCOFS(DFZDM*MASD21,IB9+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 10
      CALL ADCOFS(DFZDM*MASD22,IB10+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 11
      CALL ADCOFS(DFZDM*MASD23,IB11+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 12
      CALL ADCOFS(DFZDM*MASD24,IB12+IPSI20,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["Lmasbdec.sgf"]
```

## File RMASS.IN

```
/* This session creates the subroutine which calculates the contribution
of the mass flux for an S face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy rmass.sgf;* rmass.osf
!dele rmass.sgf;*
!@cgf rmass.sgf

filename : "rmass.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMASS(MASS)
C     Right MASs flux calculation for an S-face.
C
C     This routine calculates the mass flux on an S face
C     This routine is called by RMS or LMS so the variables
C     in common are already initialized.
C
      REAL MASS

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMASSDEC.FOR'
```

```
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

aacase : {3/4 (a3 + a9) - 1/4 (a5 + a11),\
          (a3 + a9)/2,\
          (a1 + a3 + a7 + a9)/4}
aacond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi1 at a",aa,aacase,aacond,"rmass.sgf"]

abcase : {3/4 (a3 + a9) - 1/4 (a1 + a7),\
          (a3 + a9)/2,\
          (a3 + a5 + a9 + a11)/4}
abcond : {"IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi1 at b",ab,abcase,abcond,"rmass.sgf"]

accase : {3/4 (a4 + a10) - 1/4 (a2 + a8),\
          (a4 + a10)/2,\
          (a4 + a6 + a10 + a12)/4}
accond : {"IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi1 at c",ac,accase,accond,"rmass.sgf"]

adcase : {3/4 (a4 + a10) - 1/4 (a6 + a12),\
          (a4 + a10)/2,\
          (a2 + a4 + a8 + a10)/4}
adcond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi1 at d",ad,adcase,adcond,"rmass.sgf"]

bacase : {3/4 (b2 + b8) - 1/4 (b3 + b9),\
          (b2 + b8)/2,\
          (b1 + b2 + b7 + b8)/4}
bacond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi2 at a",ba,bacase,bacond,"rmass.sgf"]

bbcase : {3/4 (b5 + b11) - 1/4 (b6 + b12),\
          (b5 + b11)/2,\
          (b4 + b5 + b10 + b11)/4}
bbcond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi2 at b",bb,bbcase,bbcond,"rmass.sgf"]

bccase : {3/4 (b5 + b11) - 1/4 (b4 + b10),\
          (b5 + b11)/2,\
          (b5 + b6 + b11 + b12)/4}
bccond : {"IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi2 at c",bc,bccase,bccond,"rmass.sgf"]

bdcase : {3/4 (b2 + b8) - 1/4 (b1 + b7),\
          (b2 + b8)/2,\
          (b2 + b3 + b8 + b9)/4}
bdcond : {"IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[3,"Psi2 at d",bd,bdcase,bdcond,"rmass.sgf"]

mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
fort["Mass flux",'mas,"rmass.sgf"]

Lpr["
      MASS = MAS

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmassdec.sgf;* Rmassdec.osf
!dele Rmassdec.sgf;*
!@cgf Rmassdec.sgf

<"decfile.in"

decfile["Rmassdec.sgf"]
```

## File LMASSA.IN

```
/* This session creates the subroutine which calculates the contribution
of the mass flux for an S face to the left hand side for each
of the equations:  x-momentum, y-momentum, and z momentum.

This is LMASSA

Due to memory problems, the LMASS routine has been split into two parts:
lmassa and lmassb.  Lmassa will be the main driver and will have lmassb1
and lmassb2 included.  lmassb1 is the ba-bd calculations, and lmassb2 is the
mas derivatives for b1-b12.  */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy lmassa.sgf;* lmassa.osf
!dele lmassa.sgf;*
!@cgf lmassa.sgf

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy lmassadec.sgf;* lmassadec.osf
!dele lmassadec.sgf;*
!@cgf lmassadec.sgf

filename : "lmassa.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"derivsub.in"
<"casederiv2.in"

Lpr["
      SUBROUTINE LMASS(DFXDM,DFYDM,DFZDM)
```

```
C     Left MASs flux calculation for an S-face.
C
C     This routine calculates contribution of the mass flux on
C     an S face to the x, y, and z momentum equations.
C     This routine is called by LMS so the variables
C     in common are already initialized.
C
      REAL DFXDM,DFYDM,DFZDM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMASSADEC.FOR'
      INCLUDE 'LMASSBDEC.FOR'
      REAL DERI

",filename]


/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,\
  b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_a1[Const] : 1 ; _a2[Const] : 1 ; _a3[Const] : 1 ; _a4[Const] : 1
_a5[Const] : 1 ; _a6[Const] : 1 ; _a7[Const] : 1 ; _a8[Const] : 1
_a9[Const] : 1 ; _a10[Const] : 1 ; _a11[Const] : 1 ; _a12[Const] : 1
_b1[Const] : 1 ; _b2[Const] : 1 ; _b3[Const] : 1 ; _b4[Const] : 1
_b5[Const] : 1 ; _b6[Const] : 1 ; _b7[Const] : 1 ; _b8[Const] : 1
_b9[Const] : 1 ; _b10[Const] : 1 ; _b11[Const] : 1 ; _b12[Const] : 1

/* As part of splitting up the tasks into lmassa and lmassb,
   ba-bb must be declared constant. */
_ba[Const] : 1 ; _bb[Const] : 1 ; _bc[Const] : 1 ; _bd[Const] : 1

/* Algorithm */

aacase : {3/4 (a3 + a9) - 1/4 (a5 + a11),\
         (a3 + a9)/2,\
         (a1 + a3 + a7 + a9)/4}
aacond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
         "IFTYP.EQ.9",\
         ".TRUE."}
casederiv[3,"Psi1 at a",aa,aacase,aacond,"lmassa.sgf"]

abcase : {3/4 (a3 + a9) - 1/4 (a1 + a7),\
         (a3 + a9)/2,\
         (a3 + a5 + a9 + a11)/4}
abcond : {"IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
         "IFTYP.EQ.9",\
         ".TRUE."}
casederiv[3,"Psi1 at b",ab,abcase,abcond,"lmassa.sgf"]

accase : {3/4 (a4 + a10) - 1/4 (a2 + a8),\
         (a4 + a10)/2,\
         (a4 + a6 + a10 + a12)/4}
accond : {"IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
         "IFTYP.EQ.9",\
         ".TRUE."}
casederiv[3,"Psi1 at c",ac,accase,accond,"lmassa.sgf"]

adcase : {3/4 (a4 + a10) - 1/4 (a6 + a12),\
          (a4 + a10)/2,\
          (a2 + a4 + a8 + a10)/4}
adcond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi1 at d",ad,adcase,adcond,"lmassa.sgf"]

Lpr["
C     Include the calculation and derivatives of ba-bd.
      INCLUDE 'LMASSB1.FOR'
",filename]

/*
These are done in LMASSB and the results written to LMASSB1.
bacase : {3/4 (b2 + b8) - 1/4 (b3 + b9),\
          (b2 + b8)/2,\
          (b1 + b2 + b7 + b8)/4}
bacond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at a",ba,bacase,bacond,"lmass.sgf"]

bbcase : {3/4 (b5 + b11) - 1/4 (b6 + b12),\
          (b5 + b11)/2,\
          (b4 + b5 + b10 + b11)/4}
bbcond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at b",bb,bbcase,bbcond,"lmass.sgf"]

bccase : {3/4 (b5 + b11) - 1/4 (b4 + b10),\
          (b5 + b11)/2,\
          (b5 + b6 + b11 + b12)/4}
bccond : {"IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at c",bc,bccase,bccond,"lmass.sgf"]

bdcase : {3/4 (b2 + b8) - 1/4 (b1 + b7),\
          (b2 + b8)/2,\
          (b2 + b3 + b8 + b9)/4}
bdcond : {"IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at d",bd,bdcase,bdcond,"lmass.sgf"]
*/

/* Memory managment */
Lpr[Mem[]]
exprlist : Union[exprlist]
nel : Len[exprlist]
Gc[]
Lpr[Mem[]]

mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
derivsub["Mass flux",'mas,"lmassa.sgf",1,12]

Lpr["
C     Include the derivatives of mas for b1-b12.
      INCLUDE 'LMASSB2.FOR'
",filename]

/*
This is done in LMASSB and in file LMASSB2.
mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
derivsub["Mass flux",'mas,"lmass.sgf",13,niv]
```

```
*/

/* Include in exprlist the dependence of mas on the dependent variables used. */
Do[i,12,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[mas,dii] \
]

/* Memory managment */
Lpr[Mem[]]
exprlist : Union[exprlist]
nel : Len[exprlist]
Gc[]
Lpr[Mem[]]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi1 at 1
      CALL ADCOFS(DFXDM*MASD1,IA1+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 2
      CALL ADCOFS(DFXDM*MASD2,IA2+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 3
      CALL ADCOFS(DFXDM*MASD3,IA3+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 4
      CALL ADCOFS(DFXDM*MASD4,IA4+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 5
      CALL ADCOFS(DFXDM*MASD5,IA5+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 6
      CALL ADCOFS(DFXDM*MASD6,IA6+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 7
      CALL ADCOFS(DFXDM*MASD7,IA7+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 8
      CALL ADCOFS(DFXDM*MASD8,IA8+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 9
      CALL ADCOFS(DFXDM*MASD9,IA9+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 10
      CALL ADCOFS(DFXDM*MASD10,IA10+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 11
      CALL ADCOFS(DFXDM*MASD11,IA11+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 12
      CALL ADCOFS(DFXDM*MASD12,IA12+IPSI10,COFX,ICOLX,NCX)

C
C     Psi2 at 1
      CALL ADCOFS(DFXDM*MASD13,IB1+IPSI20,COFX,ICOLX,NCX)
C
```

```
C      Psi2 at 2
       CALL ADCOFS(DFXDM*MASD14,IB2+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 3
       CALL ADCOFS(DFXDM*MASD15,IB3+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 4
       CALL ADCOFS(DFXDM*MASD16,IB4+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 5
       CALL ADCOFS(DFXDM*MASD17,IB5+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 6
       CALL ADCOFS(DFXDM*MASD18,IB6+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 7
       CALL ADCOFS(DFXDM*MASD19,IB7+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 8
       CALL ADCOFS(DFXDM*MASD20,IB8+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 9
       CALL ADCOFS(DFXDM*MASD21,IB9+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 10
       CALL ADCOFS(DFXDM*MASD22,IB10+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 11
       CALL ADCOFS(DFXDM*MASD23,IB11+IPSI20,COFX,ICOLX,NCX)
C
C      Psi2 at 12
       CALL ADCOFS(DFXDM*MASD24,IB12+IPSI20,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      Y - Momentum Equation
C      NCY must be set already.
C

C
C      Psi1 at 1
       CALL ADCOFS(DFYDM*MASD1,IA1+IPSI10,COFY,ICOLY,NCY)
C
C      Psi1 at 2
       CALL ADCOFS(DFYDM*MASD2,IA2+IPSI10,COFY,ICOLY,NCY)
C
C      Psi1 at 3
       CALL ADCOFS(DFYDM*MASD3,IA3+IPSI10,COFY,ICOLY,NCY)
C
C      Psi1 at 4
       CALL ADCOFS(DFYDM*MASD4,IA4+IPSI10,COFY,ICOLY,NCY)
C
C      Psi1 at 5
       CALL ADCOFS(DFYDM*MASD5,IA5+IPSI10,COFY,ICOLY,NCY)
C
C      Psi1 at 6
       CALL ADCOFS(DFYDM*MASD6,IA6+IPSI10,COFY,ICOLY,NCY)
C
C      Psi1 at 7
       CALL ADCOFS(DFYDM*MASD7,IA7+IPSI10,COFY,ICOLY,NCY)
C
```

```
C     Psi1 at 8
      CALL ADCOFS(DFYDM*MASD8,IA8+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 9
      CALL ADCOFS(DFYDM*MASD9,IA9+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 10
      CALL ADCOFS(DFYDM*MASD10,IA10+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 11
      CALL ADCOFS(DFYDM*MASD11,IA11+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 12
      CALL ADCOFS(DFYDM*MASD12,IA12+IPSI10,COFY,ICOLY,NCY)

C
C     Psi2 at 1
      CALL ADCOFS(DFYDM*MASD13,IB1+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 2
      CALL ADCOFS(DFYDM*MASD14,IB2+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 3
      CALL ADCOFS(DFYDM*MASD15,IB3+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 4
      CALL ADCOFS(DFYDM*MASD16,IB4+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 5
      CALL ADCOFS(DFYDM*MASD17,IB5+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 6
      CALL ADCOFS(DFYDM*MASD18,IB6+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 7
      CALL ADCOFS(DFYDM*MASD19,IB7+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 8
      CALL ADCOFS(DFYDM*MASD20,IB8+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 9
      CALL ADCOFS(DFYDM*MASD21,IB9+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 10
      CALL ADCOFS(DFYDM*MASD22,IB10+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 11
      CALL ADCOFS(DFYDM*MASD23,IB11+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 12
      CALL ADCOFS(DFYDM*MASD24,IB12+IPSI20,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi1 at 1
      CALL ADCOFS(DFZDM*MASD1,IA1+IPSI10,COFZ,ICOLZ,NCZ)
```

```
C
C     Psi1 at 2
      CALL ADCOFS(DFZDM*MASD2,IA2+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 3
      CALL ADCOFS(DFZDM*MASD3,IA3+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 4
      CALL ADCOFS(DFZDM*MASD4,IA4+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 5
      CALL ADCOFS(DFZDM*MASD5,IA5+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 6
      CALL ADCOFS(DFZDM*MASD6,IA6+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 7
      CALL ADCOFS(DFZDM*MASD7,IA7+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 8
      CALL ADCOFS(DFZDM*MASD8,IA8+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 9
      CALL ADCOFS(DFZDM*MASD9,IA9+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 10
      CALL ADCOFS(DFZDM*MASD10,IA10+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 11
      CALL ADCOFS(DFZDM*MASD11,IA11+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 12
      CALL ADCOFS(DFZDM*MASD12,IA12+IPSI10,COFZ,ICOLZ,NCZ)

C
C     Psi2 at 1
      CALL ADCOFS(DFZDM*MASD13,IB1+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 2
      CALL ADCOFS(DFZDM*MASD14,IB2+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 3
      CALL ADCOFS(DFZDM*MASD15,IB3+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 4
      CALL ADCOFS(DFZDM*MASD16,IB4+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 5
      CALL ADCOFS(DFZDM*MASD17,IB5+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 6
      CALL ADCOFS(DFZDM*MASD18,IB6+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 7
      CALL ADCOFS(DFZDM*MASD19,IB7+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 8
      CALL ADCOFS(DFZDM*MASD20,IB8+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 9
      CALL ADCOFS(DFZDM*MASD21,IB9+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 10
      CALL ADCOFS(DFZDM*MASD22,IB10+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 11
```

```
      CALL ADCOFS(DFZDM*MASD23,IB11+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 12
      CALL ADCOFS(DFZDM*MASD24,IB12+IPSI20,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["lmassadec.sgf"]
```

## File LMASSB.IN

```
/* This session creates the subroutine which calculates the contribution
of the mass flux for an S face to the left hand side for each
of the equations:  x-momentum, y-momentum, and z momentum.

This is LMASSB

Due to memory problems, the LMASS routine has been split into two parts:
lmassa and lmassb.  Lmassa will be the main driver and will have lmassb1
and lmassb2 included.  lmassb1 is the ba-bd calculations, and lmassb2 is the
mas derivatives for b1-b12.  */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy lmassb1.sgf;* lmassb1.osf
!dele lmassb1.sgf;*
!@cgf lmassb1.sgf

!copy lmassb2.sgf;* lmassb2.osf
!dele lmassb2.sgf;*
!@cgf lmassb2.sgf

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy lmassbdec.sgf;* lmassbdec.osf
!dele lmassbdec.sgf;*
!@cgf lmassbdec.sgf

/* Load in the fort and deriv routines. */
<"fort.in"
<"derivsub.in"
<"casederiv2.in"

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,\
  b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_a1[Const] : 1 ; _a2[Const] : 1 ; _a3[Const] : 1 ; _a4[Const] : 1
_a5[Const] : 1 ; _a6[Const] : 1 ; _a7[Const] : 1 ; _a8[Const] : 1
```

```
_a9[Const] : 1 ; _a10[Const] : 1 ; _a11[Const] : 1 ; _a12[Const] : 1
_b1[Const] : 1 ; _b2[Const] : 1 ; _b3[Const] : 1 ; _b4[Const] : 1
_b5[Const] : 1 ; _b6[Const] : 1 ; _b7[Const] : 1 ; _b8[Const] : 1
_b9[Const] : 1 ; _b10[Const] : 1 ; _b11[Const] : 1 ; _b12[Const] : 1

/* As part of splitting up the tasks into lmassa and lmassb,
   aa-ab must be declared constant. */
_aa[Const] : 1 ; _ab[Const] : 1 ; _ac[Const] : 1 ; _ad[Const] : 1

/* Algorithm */

/* This are calculted in LMASSA.
aacase : {3/4 (a3 + a9) - 1/4 (a5 + a11),\
          (a3 + a9)/2,\
          (a1 + a3 + a7 + a9)/4}
aacond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi1 at a",aa,aacase,aacond,"lmassa.sgf"]

abcase : {3/4 (a3 + a9) - 1/4 (a1 + a7),\
          (a3 + a9)/2,\
          (a3 + a5 + a9 + a11)/4}
abcond : {"IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi1 at b",ab,abcase,abcond,"lmassa.sgf"]

accase : {3/4 (a4 + a10) - 1/4 (a2 + a8),\
          (a4 + a10)/2,\
          (a4 + a6 + a10 + a12)/4}
accond : {"IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi1 at c",ac,accase,accond,"lmassa.sgf"]

adcase : {3/4 (a4 + a10) - 1/4 (a6 + a12),\
          (a4 + a10)/2,\
          (a2 + a4 + a8 + a10)/4}
adcond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi1 at d",ad,adcase,adcond,"lmassa.sgf"]
*/

bacase : {3/4 (b2 + b8) - 1/4 (b3 + b9),\
          (b2 + b8)/2,\
          (b1 + b2 + b7 + b8)/4}
bacond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at a",ba,bacase,bacond,"lmassb1.sgf"]

bbcase : {3/4 (b5 + b11) - 1/4 (b6 + b12),\
          (b5 + b11)/2,\
          (b4 + b5 + b10 + b11)/4}
bbcond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at b",bb,bbcase,bbcond,"lmassb1.sgf"]

bccase : {3/4 (b5 + b11) - 1/4 (b4 + b10),\
          (b5 + b11)/2,\
          (b5 + b6 + b11 + b12)/4}
bccond : {"IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at c",bc,bccase,bccond,"lmassb1.sgf"]

bdcase : {3/4 (b2 + b8) - 1/4 (b1 + b7),\
          (b2 + b8)/2,\
          (b2 + b3 + b8 + b9)/4}
bdcond : {"IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[3,"Psi2 at d",bd,bdcase,bdcond,"lmassb1.sgf"]

/* Memory managment */
Lpr[Mem[]]
exprlist : Union[exprlist]
nel : Len[exprlist]
Gc[]
Lpr[Mem[]]

/* This is done in LMASSA.
mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
derivsub["Mass flux",'mas,"lmassa.sgf",1,12]
*/

mas : ( (bb+ba)(ab-aa) + (bc+bb)(ac-ab) + (bd+bc)(ad-ac) + (ba+bd)(aa-ad) )/2
derivsub["Mass flux",'mas,"lmassb2.sgf",13,niv]

/* Include in exprlist the dependence of mas on the dependent variables used. */
Do[i,13,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[mas,dii] \
]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["lmassbdec.sgf"]
```

## File CD1A.IN

```
/* This session creates the subroutine which calculates the Psi1 derivatives
in xi1,xi2, and xi3 on an A face.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy cd1a.sgf;* cd1a.osf
!dele cd1a.sgf;*
!@cgf cd1a.sgf

filename : "cd1a.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RCD1A(SI11,SI12,SI13)
C     Right Curvillinear Derivatives of psi1 calculation for an A-face.
C
C     This routine calculates the psi1 derivatives
C     with respect to xi1,xi2, and xi3 on an A face.
C     This routine is called by RMA or LMA so the variables
C     in common are already initialized.
C
      REAL SI11,SI12,SI13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RCD1ADEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

da1 : (a7 - a5)/2
fort["Dpsi1/Dxi1",'da1,"cd1a.sgf"]

da2case : {(a8 - a4)/2,\
           (-3a6 + 4a8 - a4)/2,\
           (3a6 - 4a4 + a8)/2,\
           0}
da2cond : {"IFTYP.EQ.0",\
          "IFTYP.EQ.1",\
          "IFTYP.EQ.2",\
          "IFTYP.EQ.3"}
casefort[4,"Dpsi1/Dxi2",da2,da2case,da2cond,"cd1a.sgf"]

da3 : (a10 - a2)/2
fort["Dpsi1/Dxi3",'da3,"cd1a.sgf"]

Lpr["
      SI11 = DA1
      SI12 = DA2
      SI13 = DA3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rcd1adec.sgf;* Rcd1adec.osf
!dele Rcd1adec.sgf;*
!@cgf Rcd1adec.sgf

<"decfile.in"

decfile["Rcd1adec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "cd1a.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"
```

```
Lpr["
      SUBROUTINE LCD1A(DFXD11,DFXD12,DFXD13,
     1 DFYD11,DFYD12,DFYD13,
     1 DFZD11,DFZD12,DFZD13)
C     Left Curvillinear Derivative for psi1 calculation for an A-face.
C
C     This routine calculates the contribution of the psi1 derivatives
C     with respect to xi1,xi2, and xi3 on an A face to the x, y,
C     and z momentum equations.
C     This routine is called by LMA so the variables
C     in common are already initialized.
C
      REAL DFXD11,DFXD12,DFXD13
      REAL DFYD11,DFYD12,DFYD13
      REAL DFZD11,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LCD1ADEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_a1[Const] : 1 ; _a2[Const] : 1 ; _a3[Const] : 1 ; _a4[Const] : 1
_a5[Const] : 1 ; _a6[Const] : 1 ; _a7[Const] : 1 ; _a8[Const] : 1
_a9[Const] : 1 ; _a10[Const] : 1 ; _a11[Const] : 1 ; _a12[Const] : 1

/* Algorithm */

da1 : (a7 - a5)/2
deriv["Dpsi1/Dxi1",'da1,"cd1a.sgf"]

da2case : {(a8 - a4)/2,\
           (-3a6 + 4a8 - a4)/2,\
           (3a6 - 4a4 + a8)/2,\
           0}
da2cond : {"IFTYP.EQ.0",\
          "IFTYP.EQ.1",\
          "IFTYP.EQ.2",\
          "IFTYP.EQ.3"}
casederiv[4,"Dpsi1/Dxi2",da2,da2case,da2cond,"cd1a.sgf"]

da3 : (a10 - a2)/2
deriv["Dpsi1/Dxi3",'da3,"cd1a.sgf"]

/* Include in exprelist the dependence of da1, da2, and da3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[da1,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[da2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[da3,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi1 at 1
      COEF = DFXD11*DA1D1 + DFXD12*DA2D1 + DFXD13*DA3D1
      CALL ADCOFS(COEF,IA1+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 2
      COEF = DFXD11*DA1D2 + DFXD12*DA2D2 + DFXD13*DA3D2
      CALL ADCOFS(COEF,IA2+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 3
      COEF = DFXD11*DA1D3 + DFXD12*DA2D3 + DFXD13*DA3D3
      CALL ADCOFS(COEF,IA3+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 4
      COEF = DFXD11*DA1D4 + DFXD12*DA2D4 + DFXD13*DA3D4
      CALL ADCOFS(COEF,IA4+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 5
      COEF = DFXD11*DA1D5 + DFXD12*DA2D5 + DFXD13*DA3D5
      CALL ADCOFS(COEF,IA5+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 6
      COEF = DFXD11*DA1D6 + DFXD12*DA2D6 + DFXD13*DA3D6
      CALL ADCOFS(COEF,IA6+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 7
      COEF = DFXD11*DA1D7 + DFXD12*DA2D7 + DFXD13*DA3D7
      CALL ADCOFS(COEF,IA7+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 8
      COEF = DFXD11*DA1D8 + DFXD12*DA2D8 + DFXD13*DA3D8
      CALL ADCOFS(COEF,IA8+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 9
      COEF = DFXD11*DA1D9 + DFXD12*DA2D9 + DFXD13*DA3D9
      CALL ADCOFS(COEF,IA9+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 10
      COEF = DFXD11*DA1D10 + DFXD12*DA2D10 + DFXD13*DA3D10
      CALL ADCOFS(COEF,IA10+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 11
      COEF = DFXD11*DA1D11 + DFXD12*DA2D11 + DFXD13*DA3D11
      CALL ADCOFS(COEF,IA11+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 12
      COEF = DFXD11*DA1D12 + DFXD12*DA2D12 + DFXD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
```

```
C       Y - Momentum Equation
C       NCY must be set already.
C

C
C       Psi1 at 1
        COEF = DFYD11*DA1D1 + DFYD12*DA2D1 + DFYD13*DA3D1
        CALL ADCOFS(COEF,IA1+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 2
        COEF = DFYD11*DA1D2 + DFYD12*DA2D2 + DFYD13*DA3D2
        CALL ADCOFS(COEF,IA2+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 3
        COEF = DFYD11*DA1D3 + DFYD12*DA2D3 + DFYD13*DA3D3
        CALL ADCOFS(COEF,IA3+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 4
        COEF = DFYD11*DA1D4 + DFYD12*DA2D4 + DFYD13*DA3D4
        CALL ADCOFS(COEF,IA4+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 5
        COEF = DFYD11*DA1D5 + DFYD12*DA2D5 + DFYD13*DA3D5
        CALL ADCOFS(COEF,IA5+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 6
        COEF = DFYD11*DA1D6 + DFYD12*DA2D6 + DFYD13*DA3D6
        CALL ADCOFS(COEF,IA6+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 7
        COEF = DFYD11*DA1D7 + DFYD12*DA2D7 + DFYD13*DA3D7
        CALL ADCOFS(COEF,IA7+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 8
        COEF = DFYD11*DA1D8 + DFYD12*DA2D8 + DFYD13*DA3D8
        CALL ADCOFS(COEF,IA8+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 9
        COEF = DFYD11*DA1D9 + DFYD12*DA2D9 + DFYD13*DA3D9
        CALL ADCOFS(COEF,IA9+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 10
        COEF = DFYD11*DA1D10 + DFYD12*DA2D10 + DFYD13*DA3D10
        CALL ADCOFS(COEF,IA10+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 11
        COEF = DFYD11*DA1D11 + DFYD12*DA2D11 + DFYD13*DA3D11
        CALL ADCOFS(COEF,IA11+IPSI10,COFY,ICOLY,NCY)
C
C       Psi1 at 12
        COEF = DFYD11*DA1D12 + DFYD12*DA2D12 + DFYD13*DA3D12
        CALL ADCOFS(COEF,IA12+IPSI10,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C       Put Jacobians in correct location in matrix.
C       Z - Momentum Equation
C       NCZ must be set already.
C

C
C       Psi1 at 1
        COEF = DFZD11*DA1D1 + DFZD12*DA2D1 + DFZD13*DA3D1
```

```
      CALL ADCOFS(COEF,IA1+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 2
      COEF = DFZD11*DA1D2 + DFZD12*DA2D2 + DFZD13*DA3D2
      CALL ADCOFS(COEF,IA2+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 3
      COEF = DFZD11*DA1D3 + DFZD12*DA2D3 + DFZD13*DA3D3
      CALL ADCOFS(COEF,IA3+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 4
      COEF = DFZD11*DA1D4 + DFZD12*DA2D4 + DFZD13*DA3D4
      CALL ADCOFS(COEF,IA4+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 5
      COEF = DFZD11*DA1D5 + DFZD12*DA2D5 + DFZD13*DA3D5
      CALL ADCOFS(COEF,IA5+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 6
      COEF = DFZD11*DA1D6 + DFZD12*DA2D6 + DFZD13*DA3D6
      CALL ADCOFS(COEF,IA6+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 7
      COEF = DFZD11*DA1D7 + DFZD12*DA2D7 + DFZD13*DA3D7
      CALL ADCOFS(COEF,IA7+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 8
      COEF = DFZD11*DA1D8 + DFZD12*DA2D8 + DFZD13*DA3D8
      CALL ADCOFS(COEF,IA8+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 9
      COEF = DFZD11*DA1D9 + DFZD12*DA2D9 + DFZD13*DA3D9
      CALL ADCOFS(COEF,IA9+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 10
      COEF = DFZD11*DA1D10 + DFZD12*DA2D10 + DFZD13*DA3D10
      CALL ADCOFS(COEF,IA10+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 11
      COEF = DFZD11*DA1D11 + DFZD12*DA2D11 + DFZD13*DA3D11
      CALL ADCOFS(COEF,IA11+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 12
      COEF = DFZD11*DA1D12 + DFZD12*DA2D12 + DFZD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lcd1adec.sgf;* Lcd1adec.osf
!dele Lcd1adec.sgf;*
!@cgf Lcd1adec.sgf

<"decfile.in"

decfile["Lcd1adec.sgf"]
```

## File CD1B.IN

```
/* This session creates the subroutine which calculates the Psi1 derivatives
in xi1,xi2, and xi3 on a B face.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy cd1b.sgf;* cd1b.osf
!dele cd1b.sgf;*
!@cgf cd1b.sgf

filename : "cd1b.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RCD1B(SI11,SI12,SI13)
C     Right Curvillinear Derivatives of psi1 calculation for an B-face.
C
C     This routine calculates the psi1 derivatives
C     with respect to xi1,xi2, and xi3 on an B face.
C     This routine is called by RMB or LMB so the variables
C     in common are already initialized.
C
      REAL SI11,SI12,SI13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RCD1BDEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

da1 : ((a6 + a8) - (a5 + a7))/2
fort["Dpsi1/Dxi1",'da1,"cd1b.sgf"]

da2 : ((a7 + a8) - (a5 + a6))/2
fort["Dpsi1/Dxi2",'da2,"cd1b.sgf"]

da3 : ((a9+a10+a11+a12) - (a1+a2+a3+a4))/8
fort["Dpsi1/Dxi3",'da3,"cd1b.sgf"]

Lpr["
      SI11 = DA1
      SI12 = DA2
      SI13 = DA3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rcd1bdec.sgf;* Rcd1bdec.osf
!dele Rcd1bdec.sgf;*
!@cgf Rcd1bdec.sgf

<"decfile.in"

decfile["Rcd1bdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "cd1b.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LCD1B(DFXD11,DFXD12,DFXD13,
     1 DFYD11,DFYD12,DFYD13,
     1 DFZD11,DFZD12,DFZD13)
C     Left Curvillinear Derivative for psi1 calculation for an B-face.
C
C     This routine calculates the contribution of the psi1 derivatives
C     with respect to xi1,xi2, and xi3 on a B face to the x, y,
C     and z momentum equations.
C     This routine is called by LMB so the variables
C     in common are already initialized.
C
      REAL DFXD11,DFXD12,DFXD13
      REAL DFYD11,DFYD12,DFYD13
      REAL DFZD11,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LCD1BDEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_a1[Const] : 1 ; _a2[Const] : 1 ; _a3[Const] : 1 ; _a4[Const] : 1
_a5[Const] : 1 ; _a6[Const] : 1 ; _a7[Const] : 1 ; _a8[Const] : 1
_a9[Const] : 1 ; _a10[Const] : 1 ; _a11[Const] : 1 ; _a12[Const] : 1

/* Algorithm */

da1 : ((a6 + a8) - (a5 + a7))/2
deriv["Dpsi1/Dxi1",'da1,"cd1b.sgf"]

da2 : ((a7 + a8) - (a5 + a6))/2
deriv["Dpsi1/Dxi2",'da2,"cd1b.sgf"]

da3 : ((a9+a10+a11+a12) - (a1+a2+a3+a4))/8
deriv["Dpsi1/Dxi3",'da3,"cd1b.sgf"]
```

```
/* Include in exprelist the dependence of da1, da2, and da3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[da1,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[da2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[da3,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      X - Momentum Equation
C      NCX must be set already.
C

C
C      Psi1 at 1
       COEF = DFXD11*DA1D1 + DFXD12*DA2D1 + DFXD13*DA3D1
       CALL ADCOFS(COEF,IA1+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 2
       COEF = DFXD11*DA1D2 + DFXD12*DA2D2 + DFXD13*DA3D2
       CALL ADCOFS(COEF,IA2+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 3
       COEF = DFXD11*DA1D3 + DFXD12*DA2D3 + DFXD13*DA3D3
       CALL ADCOFS(COEF,IA3+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 4
       COEF = DFXD11*DA1D4 + DFXD12*DA2D4 + DFXD13*DA3D4
       CALL ADCOFS(COEF,IA4+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 5
       COEF = DFXD11*DA1D5 + DFXD12*DA2D5 + DFXD13*DA3D5
       CALL ADCOFS(COEF,IA5+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 6
       COEF = DFXD11*DA1D6 + DFXD12*DA2D6 + DFXD13*DA3D6
       CALL ADCOFS(COEF,IA6+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 7
       COEF = DFXD11*DA1D7 + DFXD12*DA2D7 + DFXD13*DA3D7
       CALL ADCOFS(COEF,IA7+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 8
       COEF = DFXD11*DA1D8 + DFXD12*DA2D8 + DFXD13*DA3D8
       CALL ADCOFS(COEF,IA8+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 9
       COEF = DFXD11*DA1D9 + DFXD12*DA2D9 + DFXD13*DA3D9
       CALL ADCOFS(COEF,IA9+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 10
       COEF = DFXD11*DA1D10 + DFXD12*DA2D10 + DFXD13*DA3D10
       CALL ADCOFS(COEF,IA10+IPSI10,COFX,ICOLX,NCX)
C
C      Psi1 at 11
       COEF = DFXD11*DA1D11 + DFXD12*DA2D11 + DFXD13*DA3D11
       CALL ADCOFS(COEF,IA11+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 12
      COEF = DFXD11*DA1D12 + DFXD12*DA2D12 + DFXD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     Psi1 at 1
      COEF = DFYD11*DA1D1 + DFYD12*DA2D1 + DFYD13*DA3D1
      CALL ADCOFS(COEF,IA1+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 2
      COEF = DFYD11*DA1D2 + DFYD12*DA2D2 + DFYD13*DA3D2
      CALL ADCOFS(COEF,IA2+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 3
      COEF = DFYD11*DA1D3 + DFYD12*DA2D3 + DFYD13*DA3D3
      CALL ADCOFS(COEF,IA3+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 4
      COEF = DFYD11*DA1D4 + DFYD12*DA2D4 + DFYD13*DA3D4
      CALL ADCOFS(COEF,IA4+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 5
      COEF = DFYD11*DA1D5 + DFYD12*DA2D5 + DFYD13*DA3D5
      CALL ADCOFS(COEF,IA5+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 6
      COEF = DFYD11*DA1D6 + DFYD12*DA2D6 + DFYD13*DA3D6
      CALL ADCOFS(COEF,IA6+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 7
      COEF = DFYD11*DA1D7 + DFYD12*DA2D7 + DFYD13*DA3D7
      CALL ADCOFS(COEF,IA7+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 8
      COEF = DFYD11*DA1D8 + DFYD12*DA2D8 + DFYD13*DA3D8
      CALL ADCOFS(COEF,IA8+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 9
      COEF = DFYD11*DA1D9 + DFYD12*DA2D9 + DFYD13*DA3D9
      CALL ADCOFS(COEF,IA9+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 10
      COEF = DFYD11*DA1D10 + DFYD12*DA2D10 + DFYD13*DA3D10
      CALL ADCOFS(COEF,IA10+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 11
      COEF = DFYD11*DA1D11 + DFYD12*DA2D11 + DFYD13*DA3D11
      CALL ADCOFS(COEF,IA11+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 12
      COEF = DFYD11*DA1D12 + DFYD12*DA2D12 + DFYD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */
```

```
Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi1 at 1
      COEF = DFZD11*DA1D1 + DFZD12*DA2D1 + DFZD13*DA3D1
      CALL ADCOFS(COEF,IA1+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 2
      COEF = DFZD11*DA1D2 + DFZD12*DA2D2 + DFZD13*DA3D2
      CALL ADCOFS(COEF,IA2+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 3
      COEF = DFZD11*DA1D3 + DFZD12*DA2D3 + DFZD13*DA3D3
      CALL ADCOFS(COEF,IA3+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 4
      COEF = DFZD11*DA1D4 + DFZD12*DA2D4 + DFZD13*DA3D4
      CALL ADCOFS(COEF,IA4+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 5
      COEF = DFZD11*DA1D5 + DFZD12*DA2D5 + DFZD13*DA3D5
      CALL ADCOFS(COEF,IA5+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 6
      COEF = DFZD11*DA1D6 + DFZD12*DA2D6 + DFZD13*DA3D6
      CALL ADCOFS(COEF,IA6+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 7
      COEF = DFZD11*DA1D7 + DFZD12*DA2D7 + DFZD13*DA3D7
      CALL ADCOFS(COEF,IA7+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 8
      COEF = DFZD11*DA1D8 + DFZD12*DA2D8 + DFZD13*DA3D8
      CALL ADCOFS(COEF,IA8+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 9
      COEF = DFZD11*DA1D9 + DFZD12*DA2D9 + DFZD13*DA3D9
      CALL ADCOFS(COEF,IA9+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 10
      COEF = DFZD11*DA1D10 + DFZD12*DA2D10 + DFZD13*DA3D10
      CALL ADCOFS(COEF,IA10+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 11
      COEF = DFZD11*DA1D11 + DFZD12*DA2D11 + DFZD13*DA3D11
      CALL ADCOFS(COEF,IA11+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 12
      COEF = DFZD11*DA1D12 + DFZD12*DA2D12 + DFZD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lcd1bdec.sgf;* Lcd1bdec.osf
!dele Lcd1bdec.sgf;*
!@cgf Lcd1bdec.sgf

<"decfile.in"

decfile["Lcd1bdec.sgf"]
```

## File CD1S.IN

```
/* This session creates the subroutine which calculates the Psi1 derivatives
in xi1,xi2, and xi3 on an S face.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy cd1s.sgf;* cd1s.osf
!dele cd1s.sgf;*
!@cgf cd1s.sgf

filename : "cd1s.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RCD1S(SI11,SI12,SI13)
C     Right Curvillinear Derivatives of psi1 calculation for an S-face.
C
C     This routine calculates the psi1 derivatives
C     with respect to xi1,xi2, and xi3 on an S face.
C     This routine is called by RMS or LMS so the variables
C     in common are already initialized.
C
      REAL SI11,SI12,SI13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RCD1SDEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

da1 : ((a4 + a10) - (a3 + a9))/2
fort["Dpsi1/Dxi1",'da1,"cd1s.sgf"]

da2case : {(-3(a3+a4+a9+a10) + 4(a5+a6+a11+a12) - (a1+a2+a7+a8))/8,\
           (3(a3+a4+a9+a10) - 4(a1+a2+a7+a8) + (a5+a6+a11+a12))/8,\
           0,\
           ((a5+a6+a11+a12) - (a1+a2+a7+a8))/8}
da2cond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
          "IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[4,"Dpsi1/Dxi2",da2,da2case,da2cond,"cd1s.sgf"]

da3 : ((a9 + a10) - (a3 + a4))/2
fort["Dpsi1/Dxi3",'da3,"cd1s.sgf"]
```

```
Lpr["
      SI11 = DA1
      SI12 = DA2
      SI13 = DA3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rcd1sdec.sgf;* Rcd1sdec.osf
!dele Rcd1sdec.sgf;*
!@cgf Rcd1sdec.sgf

<"decfile.in"

decfile["Rcd1sdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "cd1s.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LCD1S(DFXD11,DFXD12,DFXD13,
     1 DFYD11,DFYD12,DFYD13,
     1 DFZD11,DFZD12,DFZD13)
C     Left Curvillinear Derivative for psi1 calculation for an S-face.
C
C     This routine calculates the contribution of the psi1 derivatives
C     with respect to xi1,xi2, and xi3 on an S face to the x, y,
C     and z momentum equations.
C     This routine is called by LMS so the variables
C     in common are already initialized.
C
      REAL DFXD11,DFXD12,DFXD13
      REAL DFYD11,DFYD12,DFYD13
      REAL DFZD11,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LCD1SDEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12}
```

```
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_a1[Const] : 1 ; _a2[Const] : 1 ; _a3[Const] : 1 ; _a4[Const] : 1
_a5[Const] : 1 ; _a6[Const] : 1 ; _a7[Const] : 1 ; _a8[Const] : 1
_a9[Const] : 1 ; _a10[Const] : 1 ; _a11[Const] : 1 ; _a12[Const] : 1

/* Algorithm */

da1 : ((a4 + a10) - (a3 + a9))/2
deriv["Dpsi1/Dxi1",'da1,"cd1s.sgf"]

da2case : {(-3(a3+a4+a9+a10) + 4(a5+a6+a11+a12) - (a1+a2+a7+a8))/8,\
           (3(a3+a4+a9+a10) - 4(a1+a2+a7+a8) + (a5+a6+a11+a12))/8,\
           0,\
           ((a5+a6+a11+a12) - (a1+a2+a7+a8))/8}
da2cond : {"IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6",\
          "IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[4,"Dpsi1/Dxi2",da2,da2case,da2cond,"cd1s.sgf"]

da3 : ((a9 + a10) - (a3 + a4))/2
deriv["Dpsi1/Dxi3",'da3,"cd1s.sgf"]

/* Include in exprelist the dependence of da1, da2, and da3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[da1,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[da2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[da3,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi1 at 1
      COEF = DFXD11*DA1D1 + DFXD12*DA2D1 + DFXD13*DA3D1
      CALL ADCOFS(COEF,IA1+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 2
      COEF = DFXD11*DA1D2 + DFXD12*DA2D2 + DFXD13*DA3D2
      CALL ADCOFS(COEF,IA2+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 3
      COEF = DFXD11*DA1D3 + DFXD12*DA2D3 + DFXD13*DA3D3
      CALL ADCOFS(COEF,IA3+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 4
      COEF = DFXD11*DA1D4 + DFXD12*DA2D4 + DFXD13*DA3D4
      CALL ADCOFS(COEF,IA4+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 5
      COEF = DFXD11*DA1D5 + DFXD12*DA2D5 + DFXD13*DA3D5
      CALL ADCOFS(COEF,IA5+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 6
      COEF = DFXD11*DA1D6 + DFXD12*DA2D6 + DFXD13*DA3D6
      CALL ADCOFS(COEF,IA6+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 7
      COEF = DFXD11*DA1D7 + DFXD12*DA2D7 + DFXD13*DA3D7
      CALL ADCOFS(COEF,IA7+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 8
      COEF = DFXD11*DA1D8 + DFXD12*DA2D8 + DFXD13*DA3D8
      CALL ADCOFS(COEF,IA8+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 9
      COEF = DFXD11*DA1D9 + DFXD12*DA2D9 + DFXD13*DA3D9
      CALL ADCOFS(COEF,IA9+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 10
      COEF = DFXD11*DA1D10 + DFXD12*DA2D10 + DFXD13*DA3D10
      CALL ADCOFS(COEF,IA10+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 11
      COEF = DFXD11*DA1D11 + DFXD12*DA2D11 + DFXD13*DA3D11
      CALL ADCOFS(COEF,IA11+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 12
      COEF = DFXD11*DA1D12 + DFXD12*DA2D12 + DFXD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     Psi1 at 1
      COEF = DFYD11*DA1D1 + DFYD12*DA2D1 + DFYD13*DA3D1
      CALL ADCOFS(COEF,IA1+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 2
      COEF = DFYD11*DA1D2 + DFYD12*DA2D2 + DFYD13*DA3D2
      CALL ADCOFS(COEF,IA2+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 3
      COEF = DFYD11*DA1D3 + DFYD12*DA2D3 + DFYD13*DA3D3
      CALL ADCOFS(COEF,IA3+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 4
      COEF = DFYD11*DA1D4 + DFYD12*DA2D4 + DFYD13*DA3D4
      CALL ADCOFS(COEF,IA4+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 5
      COEF = DFYD11*DA1D5 + DFYD12*DA2D5 + DFYD13*DA3D5
      CALL ADCOFS(COEF,IA5+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 6
      COEF = DFYD11*DA1D6 + DFYD12*DA2D6 + DFYD13*DA3D6
      CALL ADCOFS(COEF,IA6+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 7
```

```
      COEF = DFYD11*DA1D7 + DFYD12*DA2D7 + DFYD13*DA3D7
      CALL ADCOFS(COEF,IA7+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 8
      COEF = DFYD11*DA1D8 + DFYD12*DA2D8 + DFYD13*DA3D8
      CALL ADCOFS(COEF,IA8+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 9
      COEF = DFYD11*DA1D9 + DFYD12*DA2D9 + DFYD13*DA3D9
      CALL ADCOFS(COEF,IA9+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 10
      COEF = DFYD11*DA1D10 + DFYD12*DA2D10 + DFYD13*DA3D10
      CALL ADCOFS(COEF,IA10+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 11
      COEF = DFYD11*DA1D11 + DFYD12*DA2D11 + DFYD13*DA3D11
      CALL ADCOFS(COEF,IA11+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 12
      COEF = DFYD11*DA1D12 + DFYD12*DA2D12 + DFYD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi1 at 1
      COEF = DFZD11*DA1D1 + DFZD12*DA2D1 + DFZD13*DA3D1
      CALL ADCOFS(COEF,IA1+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 2
      COEF = DFZD11*DA1D2 + DFZD12*DA2D2 + DFZD13*DA3D2
      CALL ADCOFS(COEF,IA2+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 3
      COEF = DFZD11*DA1D3 + DFZD12*DA2D3 + DFZD13*DA3D3
      CALL ADCOFS(COEF,IA3+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 4
      COEF = DFZD11*DA1D4 + DFZD12*DA2D4 + DFZD13*DA3D4
      CALL ADCOFS(COEF,IA4+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 5
      COEF = DFZD11*DA1D5 + DFZD12*DA2D5 + DFZD13*DA3D5
      CALL ADCOFS(COEF,IA5+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 6
      COEF = DFZD11*DA1D6 + DFZD12*DA2D6 + DFZD13*DA3D6
      CALL ADCOFS(COEF,IA6+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 7
      COEF = DFZD11*DA1D7 + DFZD12*DA2D7 + DFZD13*DA3D7
      CALL ADCOFS(COEF,IA7+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 8
      COEF = DFZD11*DA1D8 + DFZD12*DA2D8 + DFZD13*DA3D8
      CALL ADCOFS(COEF,IA8+IPSI10,COFZ,ICOLZ,NCZ)
C
```

```
C     Psi1 at 9
      COEF = DFZD11*DA1D9 + DFZD12*DA2D9 + DFZD13*DA3D9
      CALL ADCOFS(COEF,IA9+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 10
      COEF = DFZD11*DA1D10 + DFZD12*DA2D10 + DFZD13*DA3D10
      CALL ADCOFS(COEF,IA10+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 11
      COEF = DFZD11*DA1D11 + DFZD12*DA2D11 + DFZD13*DA3D11
      CALL ADCOFS(COEF,IA11+IPSI10,COFZ,ICOLZ,NCZ)
C
C     Psi1 at 12
      COEF = DFZD11*DA1D12 + DFZD12*DA2D12 + DFZD13*DA3D12
      CALL ADCOFS(COEF,IA12+IPSI10,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lcd1sdec.sgf;* Lcd1sdec.osf
!dele Lcd1sdec.sgf;*
!@cgf Lcd1sdec.sgf

<"decfile.in"

decfile["Lcd1sdec.sgf"]
```

## File CD2A.IN

```
/* This session creates the subroutine which calculates the Psi2 derivatives
in xi1,xi2, and xi3 on an A face.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy cd2a.sgf;* cd2a.osf
!dele cd2a.sgf;*
!@cgf cd2a.sgf

filename : "cd2a.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RCD2A(SI21,SI22,SI23)
C     Right Curvillinear Derivatives of psi2 calculation for an A-face.
C
C     This routine calculates the psi2 derivatives
C     with respect to xi1,xi2, and xi3 on an A face.
C     This routine is called by RMA or LMA so the variables
C     in common are already initialized.
C
      REAL SI21,SI22,SI23

      INCLUDE 'FACE.FOR'
```

```
C     These declarations are generated by SMP:
      INCLUDE 'RCD2ADEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

db1 : ((b6 + b8) - (b5 + b7))/2
fort["Dpsi2/Dxi1",'db1,"cd2a.sgf"]

db2 : ((b7 + b8) - (b5 + b6))/2
fort["Dpsi2/Dxi2",'db2,"cd2a.sgf"]

db3 : ((b9 + b10 + b11 + b12) - (b1 + b2 + b3 + b4))/8
fort["Dpsi2/Dxi3",'db3,"cd2a.sgf"]

Lpr["
      SI21 = DB1
      SI22 = DB2
      SI23 = DB3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rcd2adec.sgf;* Rcd2adec.osf
!dele Rcd2adec.sgf;*
!@cgf Rcd2adec.sgf

<"decfile.in"

decfile["Rcd2adec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "cd2a.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LCD2A(DFXD21,DFXD22,DFXD23,
     1 DFYD21,DFYD22,DFYD23,
     1 DFZD21,DFZD22,DFZD23)
C     Left Curvillinear Derivative for psi2 calculation for an A-face.
C
C     This routine calculates the contribution of the psi2 derivatives
C     with respect to xi1,xi2, and xi3 on an A face to the x, y,
C     and z momentum equations.
C     This routine is called by LMA so the variables
C     in common are already initialized.
```

```
C
      REAL DFXD21,DFXD22,DFXD23
      REAL DFYD21,DFYD22,DFYD23
      REAL DFZD21,DFZD22,DFZD23

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LCD2ADEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_b1[Const] : 1 ; _b2[Const] : 1 ; _b3[Const] : 1 ; _b4[Const] : 1
_b5[Const] : 1 ; _b6[Const] : 1 ; _b7[Const] : 1 ; _b8[Const] : 1
_b9[Const] : 1 ; _b10[Const] : 1 ; _b11[Const] : 1 ; _b12[Const] : 1

/* Algorithm */

db1 : ((b6 + b8) - (b5 + b7))/2
deriv["Dpsi2/Dxi1",'db1,"cd2a.sgf"]

db2 : ((b7 + b8) - (b5 + b6))/2
deriv["Dpsi2/Dxi2",'db2,"cd2a.sgf"]

db3 : ((b9 + b10 + b11 + b12) - (b1 + b2 + b3 + b4))/8
deriv["Dpsi2/Dxi3",'db3,"cd2a.sgf"]

/* Include in exprelist the dependence of db1, db2, and db3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[db1,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[db2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[db3,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi2 at 1
      COEF = DFXD21*DB1D1 + DFXD22*DB2D1 + DFXD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 2
      COEF = DFXD21*DB1D2 + DFXD22*DB2D2 + DFXD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFX,ICOLX,NCX)
```

```
C
C     Psi2 at 3
      COEF = DFXD21*DB1D3 + DFXD22*DB2D3 + DFXD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 4
      COEF = DFXD21*DB1D4 + DFXD22*DB2D4 + DFXD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 5
      COEF = DFXD21*DB1D5 + DFXD22*DB2D5 + DFXD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 6
      COEF = DFXD21*DB1D6 + DFXD22*DB2D6 + DFXD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 7
      COEF = DFXD21*DB1D7 + DFXD22*DB2D7 + DFXD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 8
      COEF = DFXD21*DB1D8 + DFXD22*DB2D8 + DFXD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 9
      COEF = DFXD21*DB1D9 + DFXD22*DB2D9 + DFXD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 10
      COEF = DFXD21*DB1D10 + DFXD22*DB2D10 + DFXD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 11
      COEF = DFXD21*DB1D11 + DFXD22*DB2D11 + DFXD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 12
      COEF = DFXD21*DB1D12 + DFXD22*DB2D12 + DFXD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C


C
C     Psi2 at 1
      COEF = DFYD21*DB1D1 + DFYD22*DB2D1 + DFYD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 2
      COEF = DFYD21*DB1D2 + DFYD22*DB2D2 + DFYD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 3
      COEF = DFYD21*DB1D3 + DFYD22*DB2D3 + DFYD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 4
      COEF = DFYD21*DB1D4 + DFYD22*DB2D4 + DFYD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 5
      COEF = DFYD21*DB1D5 + DFYD22*DB2D5 + DFYD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 6
      COEF = DFYD21*DB1D6 + DFYD22*DB2D6 + DFYD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 7
      COEF = DFYD21*DB1D7 + DFYD22*DB2D7 + DFYD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 8
      COEF = DFYD21*DB1D8 + DFYD22*DB2D8 + DFYD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 9
      COEF = DFYD21*DB1D9 + DFYD22*DB2D9 + DFYD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 10
      COEF = DFYD21*DB1D10 + DFYD22*DB2D10 + DFYD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 11
      COEF = DFYD21*DB1D11 + DFYD22*DB2D11 + DFYD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 12
      COEF = DFYD21*DB1D12 + DFYD22*DB2D12 + DFYD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi2 at 1
      COEF = DFZD21*DB1D1 + DFZD22*DB2D1 + DFZD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 2
      COEF = DFZD21*DB1D2 + DFZD22*DB2D2 + DFZD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 3
      COEF = DFZD21*DB1D3 + DFZD22*DB2D3 + DFZD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 4
      COEF = DFZD21*DB1D4 + DFZD22*DB2D4 + DFZD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 5
      COEF = DFZD21*DB1D5 + DFZD22*DB2D5 + DFZD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 6
```

```
      COEF = DFZD21*DB1D6 + DFZD22*DB2D6 + DFZD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 7
      COEF = DFZD21*DB1D7 + DFZD22*DB2D7 + DFZD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 8
      COEF = DFZD21*DB1D8 + DFZD22*DB2D8 + DFZD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 9
      COEF = DFZD21*DB1D9 + DFZD22*DB2D9 + DFZD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 10
      COEF = DFZD21*DB1D10 + DFZD22*DB2D10 + DFZD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 11
      COEF = DFZD21*DB1D11 + DFZD22*DB2D11 + DFZD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 12
      COEF = DFZD21*DB1D12 + DFZD22*DB2D12 + DFZD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lcd2adec.sgf;* Lcd2adec.osf
!dele Lcd2adec.sgf;*
!@cgf Lcd2adec.sgf

<"decfile.in"

decfile["Lcd2adec.sgf"]
```

## File CD2B.IN

```
/* This session creates the subroutine which calculates the Psi2 derivatives
in xi1,xi2, and xi3 on a B face.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy cd2b.sgf;* cd2b.osf
!dele cd2b.sgf;*
!@cgf cd2b.sgf

filename : "cd2b.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
```

```
      SUBROUTINE RCD2B(SI21,SI22,SI23)
C     Right Curvillinear Derivatives of psi2 calculation for a B-face.
C
C     This routine calculates the psi2 derivatives
C     with respect to xi1,xi2, and xi3 on an B face.
C     This routine is called by RMB or LMB so the variables
C     in common are already initialized.
C
      REAL SI21,SI22,SI23

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RCD2BDEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

db1case : {(b7 - b5)/2,\
           (-3b6 + 4b7 - b5)/2,\
           (3b6 - 4b5 + b7)/2,\
           0}
db1cond : {"IFTYP.EQ.0",\
           "IFTYP.EQ.1",\
           "IFTYP.EQ.2",\
           "IFTYP.EQ.3"}
casefort[4,"Dpsi2/Dxi1",db1,db1case,db1cond,"cd2b.sgf"]

db2 : (b8 - b4)/2
fort["Dpsi2/Dxi2",'db2,"cd2b.sgf"]

db3 : (b10 - b2)/2
fort["Dpsi2/Dxi3",'db3,"cd2b.sgf"]

Lpr["
      SI21 = DB1
      SI22 = DB2
      SI23 = DB3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rcd2bdec.sgf;* Rcd2bdec.osf
!dele Rcd2bdec.sgf;*
!@cgf Rcd2bdec.sgf

<"decfile.in"

decfile["Rcd2bdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]
```

```
filename : "cd2b.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LCD2B(DFXD21,DFXD22,DFXD23,
     1 DFYD21,DFYD22,DFYD23,
     1 DFZD21,DFZD22,DFZD23)
C     Left Curvillinear Derivative for psi2 calculation for a B-face.
C
C     This routine calculates the contribution of the psi2 derivatives
C     with respect to xi1,xi2, and xi3 on an B face to the x, y,
C     and z momentum equations.
C     This routine is called by LMB so the variables
C     in common are already initialized.
C
      REAL DFXD21,DFXD22,DFXD23
      REAL DFYD21,DFYD22,DFYD23
      REAL DFZD21,DFZD22,DFZD23

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LCD2BDEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_b1[Const] : 1 ; _b2[Const] : 1 ; _b3[Const] : 1 ; _b4[Const] : 1
_b5[Const] : 1 ; _b6[Const] : 1 ; _b7[Const] : 1 ; _b8[Const] : 1
_b9[Const] : 1 ; _b10[Const] : 1 ; _b11[Const] : 1 ; _b12[Const] : 1

/* Algorithm */

db1case : {(b7 - b5)/2,\
           (-3b6 + 4b7 - b5)/2,\
           (3b6 - 4b5 + b7)/2,\
           0}
db1cond : {"IFTYP.EQ.0",\
          "IFTYP.EQ.1",\
          "IFTYP.EQ.2",\
          "IFTYP.EQ.3"}
casederiv[4,"Dpsi2/Dxi1",db1,db1case,db1cond,"cd2b.sgf"]

db2 : (b8 - b4)/2
deriv["Dpsi2/Dxi2",'db2,"cd2b.sgf"]

db3 : (b10 - b2)/2
deriv["Dpsi2/Dxi3",'db3,"cd2b.sgf"]

/* Include in exprelist the dependence of db1, db2, and db3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[db1,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[db2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[db3,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi2 at 1
      COEF = DFXD21*DB1D1 + DFXD22*DB2D1 + DFXD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 2
      COEF = DFXD21*DB1D2 + DFXD22*DB2D2 + DFXD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 3
      COEF = DFXD21*DB1D3 + DFXD22*DB2D3 + DFXD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 4
      COEF = DFXD21*DB1D4 + DFXD22*DB2D4 + DFXD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 5
      COEF = DFXD21*DB1D5 + DFXD22*DB2D5 + DFXD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 6
      COEF = DFXD21*DB1D6 + DFXD22*DB2D6 + DFXD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 7
      COEF = DFXD21*DB1D7 + DFXD22*DB2D7 + DFXD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 8
      COEF = DFXD21*DB1D8 + DFXD22*DB2D8 + DFXD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 9
      COEF = DFXD21*DB1D9 + DFXD22*DB2D9 + DFXD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 10
      COEF = DFXD21*DB1D10 + DFXD22*DB2D10 + DFXD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 11
      COEF = DFXD21*DB1D11 + DFXD22*DB2D11 + DFXD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 12
      COEF = DFXD21*DB1D12 + DFXD22*DB2D12 + DFXD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFX,ICOLX,NCX)
",filename]
```

```
/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     Psi2 at 1
      COEF = DFYD21*DB1D1 + DFYD22*DB2D1 + DFYD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 2
      COEF = DFYD21*DB1D2 + DFYD22*DB2D2 + DFYD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 3
      COEF = DFYD21*DB1D3 + DFYD22*DB2D3 + DFYD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 4
      COEF = DFYD21*DB1D4 + DFYD22*DB2D4 + DFYD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 5
      COEF = DFYD21*DB1D5 + DFYD22*DB2D5 + DFYD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 6
      COEF = DFYD21*DB1D6 + DFYD22*DB2D6 + DFYD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 7
      COEF = DFYD21*DB1D7 + DFYD22*DB2D7 + DFYD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 8
      COEF = DFYD21*DB1D8 + DFYD22*DB2D8 + DFYD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 9
      COEF = DFYD21*DB1D9 + DFYD22*DB2D9 + DFYD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 10
      COEF = DFYD21*DB1D10 + DFYD22*DB2D10 + DFYD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 11
      COEF = DFYD21*DB1D11 + DFYD22*DB2D11 + DFYD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 12
      COEF = DFYD21*DB1D12 + DFYD22*DB2D12 + DFYD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi2 at 1
      COEF = DFZD21*DB1D1 + DFZD22*DB2D1 + DFZD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 2
      COEF = DFZD21*DB1D2 + DFZD22*DB2D2 + DFZD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 3
      COEF = DFZD21*DB1D3 + DFZD22*DB2D3 + DFZD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 4
      COEF = DFZD21*DB1D4 + DFZD22*DB2D4 + DFZD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 5
      COEF = DFZD21*DB1D5 + DFZD22*DB2D5 + DFZD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 6
      COEF = DFZD21*DB1D6 + DFZD22*DB2D6 + DFZD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 7
      COEF = DFZD21*DB1D7 + DFZD22*DB2D7 + DFZD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 8
      COEF = DFZD21*DB1D8 + DFZD22*DB2D8 + DFZD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 9
      COEF = DFZD21*DB1D9 + DFZD22*DB2D9 + DFZD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 10
      COEF = DFZD21*DB1D10 + DFZD22*DB2D10 + DFZD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 11
      COEF = DFZD21*DB1D11 + DFZD22*DB2D11 + DFZD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 12
      COEF = DFZD21*DB1D12 + DFZD22*DB2D12 + DFZD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lcd2bdec.sgf;* Lcd2bdec.osf
!dele Lcd2bdec.sgf;*
!@cgf Lcd2bdec.sgf
```

```
<"decfile.in"

decfile["Lcd2bdec.sgf"]
```

## File CD2S.IN

```
/* This session creates the subroutine which calculates the Psi2 derivatives
in xi1,xi2, and xi3 on an S face.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy cd2s.sgf;* cd2s.osf
!dele cd2s.sgf;*
!@cgf cd2s.sgf

filename : "cd2s.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RCD2S(SI21,SI22,SI23)
C     Right Curvillinear Derivatives of psi2 calculation for an S-face.
C
C     This routine calculates the psi2 derivatives
C     with respect to xi1,xi2, and xi3 on an S face.
C     This routine is called by RMS or LMS so the variables
C     in common are already initialized.
C
      REAL SI21,SI22,SI23

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RCD2SDEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

db1case : {(-3(b2+b5+b8+b11) + 4(b3+b6+b9+b12) - (b1+b4+b7+b10))/8,\
           (3(b2+b5+b8+b11) - 4(b1+b4+b7+b10) + (b3+b6+b9+b12))/8,\
           0,\
           ((b3+b6+b9+b12) - (b1+b4+b7+b10))/8}
db1cond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casefort[4,"Dpsi2/Dxi1",db1,db1case,db1cond,"cd2s.sgf"]

db2 : ((b5 + b11) - (b2 + b8))/2
fort["Dpsi2/Dxi2",'db2,"cd2s.sgf"]

db3 : ((b8 + b11) - (b2 + b5))/2
fort["Dpsi2/Dxi3",'db3,"cd2s.sgf"]

Lpr["
      SI21 = DB1
      SI22 = DB2
      SI23 = DB3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rcd2sdec.sgf;* Rcd2sdec.osf
!dele Rcd2sdec.sgf;*
!@cgf Rcd2sdec.sgf

<"decfile.in"

decfile["Rcd2sdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "cd2s.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LCD2S(DFXD21,DFXD22,DFXD23,
     1 DFYD21,DFYD22,DFYD23,
     1 DFZD21,DFZD22,DFZD23)
C     Left Curvillinear Derivative for psi2 calculation for an S-face.
C
C     This routine calculates the contribution of the psi2 derivatives
C     with respect to xi1,xi2, and xi3 on an S face to the x, y,
C     and z momentum equations.
C     This routine is called by LMS so the variables
C     in common are already initialized.
C
      REAL DFXD21,DFXD22,DFXD23
      REAL DFYD21,DFYD22,DFYD23
      REAL DFZD21,DFZD22,DFZD23

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LCD2SDEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_b1[Const] : 1 ; _b2[Const] : 1 ; _b3[Const] : 1 ; _b4[Const] : 1
```

```
_b5[Const] : 1 ; _b6[Const] : 1 ; _b7[Const] : 1 ; _b8[Const] : 1
_b9[Const] : 1 ; _b10[Const] : 1 ; _b11[Const] : 1 ; _b12[Const] : 1

/* Algorithm */

db1case : {(-3(b2+b5+b8+b11) + 4(b3+b6+b9+b12) - (b1+b4+b7+b10))/8,\
           (3(b2+b5+b8+b11) - 4(b1+b4+b7+b10) + (b3+b6+b9+b12))/8,\
           0,\
           ((b3+b6+b9+b12) - (b1+b4+b7+b10))/8}
db1cond : {"IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7",\
          "IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8",\
          "IFTYP.EQ.9",\
          ".TRUE."}
casederiv[4,"Dpsi2/Dxi1",db1,db1case,db1cond,"cd2s.sgf"]

db2 : ((b5 + b11) - (b2 + b8))/2
deriv["Dpsi2/Dxi2",'db2,"cd2s.sgf"]

db3 : ((b8 + b11) - (b2 + b5))/2
deriv["Dpsi2/Dxi3",'db3,"cd2s.sgf"]

/* Include in exprelist the dependence of db1, db2, and db3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[db1,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[db2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[db3,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi2 at 1
      COEF = DFXD21*DB1D1 + DFXD22*DB2D1 + DFXD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 2
      COEF = DFXD21*DB1D2 + DFXD22*DB2D2 + DFXD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 3
      COEF = DFXD21*DB1D3 + DFXD22*DB2D3 + DFXD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 4
      COEF = DFXD21*DB1D4 + DFXD22*DB2D4 + DFXD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 5
      COEF = DFXD21*DB1D5 + DFXD22*DB2D5 + DFXD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 6
      COEF = DFXD21*DB1D6 + DFXD22*DB2D6 + DFXD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFX,ICOLX,NCX)
```

```
C
C     Psi2 at 7
      COEF = DFXD21*DB1D7 + DFXD22*DB2D7 + DFXD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 8
      COEF = DFXD21*DB1D8 + DFXD22*DB2D8 + DFXD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 9
      COEF = DFXD21*DB1D9 + DFXD22*DB2D9 + DFXD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 10
      COEF = DFXD21*DB1D10 + DFXD22*DB2D10 + DFXD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 11
      COEF = DFXD21*DB1D11 + DFXD22*DB2D11 + DFXD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 12
      COEF = DFXD21*DB1D12 + DFXD22*DB2D12 + DFXD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     Psi2 at 1
      COEF = DFYD21*DB1D1 + DFYD22*DB2D1 + DFYD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 2
      COEF = DFYD21*DB1D2 + DFYD22*DB2D2 + DFYD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 3
      COEF = DFYD21*DB1D3 + DFYD22*DB2D3 + DFYD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 4
      COEF = DFYD21*DB1D4 + DFYD22*DB2D4 + DFYD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 5
      COEF = DFYD21*DB1D5 + DFYD22*DB2D5 + DFYD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 6
      COEF = DFYD21*DB1D6 + DFYD22*DB2D6 + DFYD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 7
      COEF = DFYD21*DB1D7 + DFYD22*DB2D7 + DFYD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 8
      COEF = DFYD21*DB1D8 + DFYD22*DB2D8 + DFYD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 9
      COEF = DFYD21*DB1D9 + DFYD22*DB2D9 + DFYD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 10
      COEF = DFYD21*DB1D10 + DFYD22*DB2D10 + DFYD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 11
      COEF = DFYD21*DB1D11 + DFYD22*DB2D11 + DFYD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 12
      COEF = DFYD21*DB1D12 + DFYD22*DB2D12 + DFYD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     Psi2 at 1
      COEF = DFZD21*DB1D1 + DFZD22*DB2D1 + DFZD23*DB3D1
      CALL ADCOFS(COEF,IB1+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 2
      COEF = DFZD21*DB1D2 + DFZD22*DB2D2 + DFZD23*DB3D2
      CALL ADCOFS(COEF,IB2+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 3
      COEF = DFZD21*DB1D3 + DFZD22*DB2D3 + DFZD23*DB3D3
      CALL ADCOFS(COEF,IB3+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 4
      COEF = DFZD21*DB1D4 + DFZD22*DB2D4 + DFZD23*DB3D4
      CALL ADCOFS(COEF,IB4+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 5
      COEF = DFZD21*DB1D5 + DFZD22*DB2D5 + DFZD23*DB3D5
      CALL ADCOFS(COEF,IB5+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 6
      COEF = DFZD21*DB1D6 + DFZD22*DB2D6 + DFZD23*DB3D6
      CALL ADCOFS(COEF,IB6+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 7
      COEF = DFZD21*DB1D7 + DFZD22*DB2D7 + DFZD23*DB3D7
      CALL ADCOFS(COEF,IB7+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 8
      COEF = DFZD21*DB1D8 + DFZD22*DB2D8 + DFZD23*DB3D8
      CALL ADCOFS(COEF,IB8+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 9
      COEF = DFZD21*DB1D9 + DFZD22*DB2D9 + DFZD23*DB3D9
      CALL ADCOFS(COEF,IB9+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 10
```

```
      COEF = DFZD21*DB1D10 + DFZD22*DB2D10 + DFZD23*DB3D10
      CALL ADCOFS(COEF,IB10+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 11
      COEF = DFZD21*DB1D11 + DFZD22*DB2D11 + DFZD23*DB3D11
      CALL ADCOFS(COEF,IB11+IPSI20,COFZ,ICOLZ,NCZ)
C
C     Psi2 at 12
      COEF = DFZD21*DB1D12 + DFZD22*DB2D12 + DFZD23*DB3D12
      CALL ADCOFS(COEF,IB12+IPSI20,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lcd2sdec.sgf;* Lcd2sdec.osf
!dele Lcd2sdec.sgf;*
!@cgf Lcd2sdec.sgf

<"decfile.in"

decfile["Lcd2sdec.sgf"]
```

## File ANP.IN

```
/* This session creates the subroutine which calculates the Area Normals
for planer grids.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy anp.sgf;* anp.osf
!dele anp.sgf;*
!@cgf anp.sgf

filename : "anp.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RANP(AXV,AYV,AZV)
C     Right Area Normal Planer.
C
C     This routine calculates the area normals for a current face for a
C     planer grid.
C     The variables in common are already initialized for the current face.
C
      REAL AXV,AYV,AZV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RANPDEC.FOR'
C     Declare and set the moving grid variables to 0.
      REAL DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC
      DATA DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC/0.,0.,0.,0.,0.,0./
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

x1av : x1ac + dx1a + d1dlea dslead
fort["X1 at a.",'x1av,"anp.sgf"]

x1bv : x1bc + dx1b + d1dleb dslebc
fort["X1 at b.",'x1bv,"anp.sgf"]

x1cv : x1cc + dx1c + d1dlec dslebc
fort["X1 at c.",'x1cv,"anp.sgf"]

x1dv : x1dc + dx1d + d1dled dslead
fort["X1 at d.",'x1dv,"anp.sgf"]

x2av : x2ac + d2dlea dslead
fort["X2 at a.",'x2av,"anp.sgf"]

x2bv : x2bc + d2dleb dslebc
fort["X2 at b.",'x2bv,"anp.sgf"]

x2cv : x2cc + d2dlec dslebc
fort["X2 at c.",'x2cv,"anp.sgf"]

x2dv : x2dc + d2dled dslead
fort["X2 at d.",'x2dv,"anp.sgf"]

za : x3ac + d3dlea dslead
fort["Z or x3 at a.",'za,"anp.sgf"]

zb : x3bc + d3dleb dslebc
fort["Z or x3 at b.",'zb,"anp.sgf"]

zc : x3cc + d3dlec dslebc
fort["Z or x3 at c.",'zc,"anp.sgf"]

zd : x3dc + d3dled dslead
fort["Z or x3 at d.",'zd,"anp.sgf"]

xa : x1av
fort["X at a calculated.",'xa,"anp.sgf"]

xb : x1bv
fort["X at b calculated.",'xb,"anp.sgf"]

xc : x1cv
fort["X at c calculated.",'xc,"anp.sgf"]

xd : x1dv
fort["X at d calculated.",'xd,"anp.sgf"]

ya : x2av
fort["Y at a calculated.",'ya,"anp.sgf"]

yb : x2bv
fort["Y at b calculated.",'yb,"anp.sgf"]

yc : x2cv
fort["Y at c calculated.",'yc,"anp.sgf"]

yd : x2dv
fort["Y at d calculated.",'yd,"anp.sgf"]
```

```
dx1 : ((xc + xd) - (xa + xb))/2
fort["Delta x for vector 1",'dx1,"anp.sgf"]

dy1 : ((yc + yd) - (ya + yb))/2
fort["Delta y for vector 1",'dy1,"anp.sgf"]

dz1 : ((zc +  zd) - (za + zb))/2
fort["Delta z for vector 1",'dz1,"anp.sgf"]

dx2 : ((xb + xc) - (xa + xd))/2
fort["Delta x for vector 2",'dx2,"anp.sgf"]

dy2 : ((yb + yc) - (ya + yd))/2
fort["Delta y for vector 2",'dy2,"anp.sgf"]

dz2 : ((zb + zc) - (za + zd))/2
fort["Delta z for vector 2",'dz2,"anp.sgf"]

ax : dy1 dz2 - dy2 dz1
fort["X component of normal.",'ax,"anp.sgf"]

ay : dx2 dz1 - dx1 dz2
fort["Y component of normal.",'ay,"anp.sgf"]

az : dx1 dy2 - dx2 dy1
fort["Z component of normal.",'az,"anp.sgf"]

Lpr["
      AXV = AX
      AYV = AY
      AZV = AZ
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Ranpdec.sgf;* Ranpdec.osf
!dele Ranpdec.sgf;*
!@cgf Ranpdec.sgf

<"decfile.in"

decfile["Ranpdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lanpdec.sgf;* Lanpdec.osf
!dele Lanpdec.sgf;*
!@cgf Lanpdec.sgf

filename : "anp.sgf"
```

```
/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LANP(DFXDAX,DFXDAY,DFXDAZ,
     1 DFYDAX,DFYDAY,DFYDAZ,
     1 DFZDAX,DFZDAY,DFZDAZ)
C     Left Area Normal Planer.
C
C     This routine calculates the contribution of the area normals for
C     the current face for a planer grid to the left hand sides of the x, y, and
C     z momentum eqs.
C     The variables in common are already initialized for the current face.
C
      REAL DFXDAX,DFXDAY,DFXDAZ
      REAL DFYDAX,DFYDAY,DFYDAZ
      REAL DFZDAX,DFZDAY,DFZDAZ

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LANPDEC.FOR'
C     Declare and set the moving grid variables to 0.
      REAL DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC
      DATA DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC/0.,0.,0.,0.,0.,0./

      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx1a,dx1b,dx1c,dx1d,dslead,dslebc}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx1a[Const] : 1 ; _dx1b[Const] : 1 ; _dx1c[Const] : 1 ; _dx1d[Const] : 1
_dslead[Const] : 1 ; _dslebc[Const] : 1
_x1ac[Const] : 1 ; _x1bc[Const] : 1 ; _x1cc[Const] : 1 ; _x1dc[Const] : 1
_x2ac[Const] : 1 ; _x2bc[Const] : 1 ; _x2cc[Const] : 1 ; _x2dc[Const] : 1
_x3ac[Const] : 1 ; _x3bc[Const] : 1 ; _x3cc[Const] : 1 ; _x3dc[Const] : 1
_d1dlea[Const] : 1 ; _d1dleb[Const] : 1
_d1dlec[Const] : 1 ; _d1dled[Const] : 1
_d2dlea[Const] : 1 ; _d2dleb[Const] : 1
_d2dlec[Const] : 1 ; _d2dled[Const] : 1
_d3dlea[Const] : 1 ; _d3dleb[Const] : 1
_d3dlec[Const] : 1 ; _d3dled[Const] : 1

/* Algorithm */

x1av : x1ac + dx1a + d1dlea dslead
deriv["X1 at a.",'x1av,"anp.sgf"]

x1bv : x1bc + dx1b + d1dleb dslebc
deriv["X1 at b.",'x1bv,"anp.sgf"]

x1cv : x1cc + dx1c + d1dlec dslebc
deriv["X1 at c.",'x1cv,"anp.sgf"]

x1dv : x1dc + dx1d + d1dled dslead
```

```
deriv["X1 at d.",'x1dv,"anp.sgf"]

x2av : x2ac + d2dlea dslead
deriv["X2 at a.",'x2av,"anp.sgf"]

x2bv : x2bc + d2dleb dslebc
deriv["X2 at b.",'x2bv,"anp.sgf"]

x2cv : x2cc + d2dlec dslebc
deriv["X2 at c.",'x2cv,"anp.sgf"]

x2dv : x2dc + d2dled dslead
deriv["X2 at d.",'x2dv,"anp.sgf"]

za : x3ac + d3dlea dslead
deriv["Z or x3 at a.",'za,"anp.sgf"]

zb : x3bc + d3dleb dslebc
deriv["Z or x3 at b.",'zb,"anp.sgf"]

zc : x3cc + d3dlec dslebc
deriv["Z or x3 at c.",'zc,"anp.sgf"]

zd : x3dc + d3dled dslead
deriv["Z or x3 at d.",'zd,"anp.sgf"]

xa : x1av
deriv["X at a calculated.",'xa,"anp.sgf"]

xb : x1bv
deriv["X at b calculated.",'xb,"anp.sgf"]

xc : x1cv
deriv["X at c calculated.",'xc,"anp.sgf"]

xd : x1dv
deriv["X at d calculated.",'xd,"anp.sgf"]

ya : x2av
deriv["Y at a calculated.",'ya,"anp.sgf"]

yb : x2bv
deriv["Y at b calculated.",'yb,"anp.sgf"]

yc : x2cv
deriv["Y at c calculated.",'yc,"anp.sgf"]

yd : x2dv
deriv["Y at d calculated.",'yd,"anp.sgf"]

dx1 : ((xc + xd) - (xa + xb))/2
deriv["Delta x for vector 1",'dx1,"anp.sgf"]

dy1 : ((yc + yd) - (ya + yb))/2
deriv["Delta y for vector 1",'dy1,"anp.sgf"]

dz1 : ((zc +  zd) - (za + zb))/2
deriv["Delta z for vector 1",'dz1,"anp.sgf"]

dx2 : ((xb + xc) - (xa + xd))/2
deriv["Delta x for vector 2",'dx2,"anp.sgf"]

dy2 : ((yb + yc) - (ya + yd))/2
deriv["Delta y for vector 2",'dy2,"anp.sgf"]

dz2 : ((zb + zc) - (za + zd))/2
```

```
deriv["Delta z for vector 2",'dz2,"anp.sgf"]

ax : dy1 dz2 - dy2 dz1
deriv["X component of normal.",'ax,"anp.sgf"]

ay : dx2 dz1 - dx1 dz2
deriv["Y component of normal.",'ay,"anp.sgf"]

az : dx1 dy2 - dx2 dy1
deriv["Z component of normal.",'az,"anp.sgf"]

/* Include in exprelist the dependence of ax, ay and az on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ax,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ay,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[az,dii] \
]

Lpr["
C     The dependence on delta x1 at volume nodes.
      CALL LDX1VN(LA,DFXDAX*AXD1+DFXDAY*AYD1+DFXDAZ*AZD1,
     1              DFYDAX*AXD1+DFYDAY*AYD1+DFYDAZ*AZD1,
     1              DFZDAX*AXD1+DFZDAY*AYD1+DFZDAZ*AZD1)

      CALL LDX1VN(LB,DFXDAX*AXD2+DFXDAY*AYD2+DFXDAZ*AZD2,
     1              DFYDAX*AXD2+DFYDAY*AYD2+DFYDAZ*AZD2,
     1              DFZDAX*AXD2+DFZDAY*AYD2+DFZDAZ*AZD2)

      CALL LDX1VN(LC,DFXDAX*AXD3+DFXDAY*AYD3+DFXDAZ*AZD3,
     1              DFYDAX*AXD3+DFYDAY*AYD3+DFYDAZ*AZD3,
     1              DFZDAX*AXD3+DFZDAY*AYD3+DFZDAZ*AZD3)

      CALL LDX1VN(LD,DFXDAX*AXD4+DFXDAY*AYD4+DFXDAZ*AZD4,
     1              DFYDAX*AXD4+DFYDAY*AYD4+DFYDAZ*AZD4,
     1              DFZDAX*AXD4+DFZDAY*AYD4+DFZDAZ*AZD4)

C     The dependence on delta sble for a and d.
      CALL LSBLEN(JSLEAD,NJJ,DFXDAX*AXD5+DFXDAY*AYD5+DFXDAZ*AZD5,
     1                       DFYDAX*AXD5+DFYDAY*AYD5+DFYDAZ*AZD5,
     1                       DFZDAX*AXD5+DFZDAY*AYD5+DFZDAZ*AZD5)

C     The dependence on delta sble for b and c.
      CALL LSBLEN(JSLEBC,NJJ,DFXDAX*AXD6+DFXDAY*AYD6+DFXDAZ*AZD6,
     1                       DFYDAX*AXD6+DFYDAY*AYD6+DFYDAZ*AZD6,
     1                       DFZDAX*AXD6+DFZDAY*AYD6+DFZDAZ*AZD6)

",filename]

Lpr["

      RETURN
      END
",filename]

<"decfile.in"

decfile["Lanpdec.sgf"]
```

## File ANC.IN

```
/* This session creates the subroutine which calculates the Area Normals
for cylindrical grids.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy anc.sgf;* anc.osf
!dele anc.sgf;*
!@cgf anc.sgf

filename : "anc.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RANC(AXV,AYV,AZV)
C     Right Area Normal Cylindrical.
C
C     This routine calculates the area normals for a current face for a
C     cylindrical grid.
C     The variables in common are already initialized for the current face.
C
      REAL AXV,AYV,AZV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RANCDEC.FOR'
C     Declare and set the moving grid variables to 0.
      REAL DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC
      DATA DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC/0.,0.,0.,0.,0.,0./

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

x1av : x1ac + dx1a + d1dlea dslead
fort["X1 at a.",'x1av,"anc.sgf"]

x1bv : x1bc + dx1b + d1dleb dslebc
fort["X1 at b.",'x1bv,"anc.sgf"]

x1cv : x1cc + dx1c + d1dlec dslebc
fort["X1 at c.",'x1cv,"anc.sgf"]

x1dv : x1dc + dx1d + d1dled dslead
fort["X1 at d.",'x1dv,"anc.sgf"]

x2av : x2ac + d2dlea dslead
fort["X2 at a.",'x2av,"anc.sgf"]

x2bv : x2bc + d2dleb dslebc
fort["X2 at b.",'x2bv,"anc.sgf"]

x2cv : x2cc + d2dlec dslebc
fort["X2 at c.",'x2cv,"anc.sgf"]

x2dv : x2dc + d2dled dslead
fort["X2 at d.",'x2dv,"anc.sgf"]
```

```
za : x3ac + d3dlea dslead
fort["Z or x3 at a.",'za,"anc.sgf"]

zb : x3bc + d3dleb dslebc
fort["Z or x3 at b.",'zb,"anc.sgf"]

zc : x3cc + d3dlec dslebc
fort["Z or x3 at c.",'zc,"anc.sgf"]

zd : x3dc + d3dled dslead
fort["Z or x3 at d.",'zd,"anc.sgf"]

xa : x2av Sin[x1av]
fort["X at a calculated.",'xa,"anc.sgf"]

xb : x2bv Sin[x1bv]
fort["X at b calculated.",'xb,"anc.sgf"]

xc : x2cv Sin[x1cv]
fort["X at c calculated.",'xc,"anc.sgf"]

xd : x2dv Sin[x1dv]
fort["X at d calculated.",'xd,"anc.sgf"]

ya : x2av Cos[x1av]
fort["Y at a calculated.",'ya,"anc.sgf"]

yb : x2bv Cos[x1bv]
fort["Y at b calculated.",'yb,"anc.sgf"]

yc : x2cv Cos[x1cv]
fort["Y at c calculated.",'yc,"anc.sgf"]

yd : x2dv Cos[x1dv]
fort["Y at d calculated.",'yd,"anc.sgf"]

dx1 : ((xc + xd) - (xa + xb))/2
fort["Delta x for vector 1",'dx1,"anc.sgf"]

dy1 : ((yc + yd) - (ya + yb))/2
fort["Delta y for vector 1",'dy1,"anc.sgf"]

dz1 : ((zc +  zd) - (za + zb))/2
fort["Delta z for vector 1",'dz1,"anc.sgf"]

dx2 : ((xb + xc) - (xa + xd))/2
fort["Delta x for vector 2",'dx2,"anc.sgf"]

dy2 : ((yb + yc) - (ya + yd))/2
fort["Delta y for vector 2",'dy2,"anc.sgf"]

dz2 : ((zb + zc) - (za + zd))/2
fort["Delta z for vector 2",'dz2,"anc.sgf"]

ax : dy1 dz2 - dy2 dz1
fort["X component of normal.",'ax,"anc.sgf"]

ay : dx2 dz1 - dx1 dz2
fort["Y component of normal.",'ay,"anc.sgf"]

az : dx1 dy2 - dx2 dy1
fort["Z component of normal.",'az,"anc.sgf"]

Lpr["
      AXV = AX
      AYV = AY
      AZV = AZ
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rancdec.sgf;* Rancdec.osf
!dele Rancdec.sgf;*
!@cgf Rancdec.sgf

<"decfile.in"

decfile["Rancdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lancdec.sgf;* Lancdec.osf
!dele Lancdec.sgf;*
!@cgf Lancdec.sgf

filename : "anc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LANC(DFXDAX,DFXDAY,DFXDAZ,
     1 DFYDAX,DFYDAY,DFYDAZ,
     1 DFZDAX,DFZDAY,DFZDAZ)
C     Left Area Normal Cylindrical.
C
C     This routine calculates the contribution of the area normals for
C     the current face for a cylindrical grid to the left hand sides of
C     the x, y, and z momentum eqs.
C     The variables in common are already initialized for the current face.
C
      REAL DFXDAX,DFXDAY,DFXDAZ
      REAL DFYDAX,DFYDAY,DFYDAZ
      REAL DFZDAX,DFZDAY,DFZDAZ

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LANCDEC.FOR'
C     Declare and set the moving grid variables to 0.
      REAL DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC
      DATA DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC/0.,0.,0.,0.,0.,0./

      REAL DERI
      REAL COEF
```

```
",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx1a,dx1b,dx1c,dx1d,dslead,dslebc}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx1a[Const] : 1 ; _dx1b[Const] : 1 ; _dx1c[Const] : 1 ; _dx1d[Const] : 1
_dslead[Const] : 1 ; _dslebc[Const] : 1
_x1ac[Const] : 1 ; _x1bc[Const] : 1 ; _x1cc[Const] : 1 ; _x1dc[Const] : 1
_x2ac[Const] : 1 ; _x2bc[Const] : 1 ; _x2cc[Const] : 1 ; _x2dc[Const] : 1
_x3ac[Const] : 1 ; _x3bc[Const] : 1 ; _x3cc[Const] : 1 ; _x3dc[Const] : 1
_d1dlea[Const] : 1 ; _d1dleb[Const] : 1
_d1dlec[Const] : 1 ; _d1dled[Const] : 1
_d2dlea[Const] : 1 ; _d2dleb[Const] : 1
_d2dlec[Const] : 1 ; _d2dled[Const] : 1
_d3dlea[Const] : 1 ; _d3dleb[Const] : 1
_d3dlec[Const] : 1 ; _d3dled[Const] : 1

/* Algorithm */

x1av : x1ac + dx1a + d1dlea dslead
deriv["X1 at a.",'x1av,"anc.sgf"]

x1bv : x1bc + dx1b + d1dleb dslebc
deriv["X1 at b.",'x1bv,"anc.sgf"]

x1cv : x1cc + dx1c + d1dlec dslebc
deriv["X1 at c.",'x1cv,"anc.sgf"]

x1dv : x1dc + dx1d + d1dled dslead
deriv["X1 at d.",'x1dv,"anc.sgf"]

x2av : x2ac + d2dlea dslead
deriv["X2 at a.",'x2av,"anc.sgf"]

x2bv : x2bc + d2dleb dslebc
deriv["X2 at b.",'x2bv,"anc.sgf"]

x2cv : x2cc + d2dlec dslebc
deriv["X2 at c.",'x2cv,"anc.sgf"]

x2dv : x2dc + d2dled dslead
deriv["X2 at d.",'x2dv,"anc.sgf"]

za : x3ac + d3dlea dslead
deriv["Z or x3 at a.",'za,"anc.sgf"]

zb : x3bc + d3dleb dslebc
deriv["Z or x3 at b.",'zb,"anc.sgf"]

zc : x3cc + d3dlec dslebc
deriv["Z or x3 at c.",'zc,"anc.sgf"]

zd : x3dc + d3dled dslead
deriv["Z or x3 at d.",'zd,"anc.sgf"]

xa : x2av Sin[x1av]
deriv["X at a calculated.",'xa,"anc.sgf"]

xb : x2bv Sin[x1bv]
deriv["X at b calculated.",'xb,"anc.sgf"]
```

```
xc : x2cv Sin[x1cv]
deriv["X at c calculated.",'xc,"anc.sgf"]

xd : x2dv Sin[x1dv]
deriv["X at d calculated.",'xd,"anc.sgf"]

ya : x2av Cos[x1av]
deriv["Y at a calculated.",'ya,"anc.sgf"]

yb : x2bv Cos[x1bv]
deriv["Y at b calculated.",'yb,"anc.sgf"]

yc : x2cv Cos[x1cv]
deriv["Y at c calculated.",'yc,"anc.sgf"]

yd : x2dv Cos[x1dv]
deriv["Y at d calculated.",'yd,"anc.sgf"]

dx1 : ((xc + xd) - (xa + xb))/2
deriv["Delta x for vector 1",'dx1,"anc.sgf"]

dy1 : ((yc + yd) - (ya + yb))/2
deriv["Delta y for vector 1",'dy1,"anc.sgf"]

dz1 : ((zc +  zd) - (za + zb))/2
deriv["Delta z for vector 1",'dz1,"anc.sgf"]

dx2 : ((xb + xc) - (xa + xd))/2
deriv["Delta x for vector 2",'dx2,"anc.sgf"]

dy2 : ((yb + yc) - (ya + yd))/2
deriv["Delta y for vector 2",'dy2,"anc.sgf"]

dz2 : ((zb + zc) - (za + zd))/2
deriv["Delta z for vector 2",'dz2,"anc.sgf"]

ax : dy1 dz2 - dy2 dz1
deriv["X component of normal.",'ax,"anc.sgf"]

ay : dx2 dz1 - dx1 dz2
deriv["Y component of normal.",'ay,"anc.sgf"]

az : dx1 dy2 - dx2 dy1
deriv["Z component of normal.",'az,"anc.sgf"]

/* Include in exprelist the dependence of ax, ay and az on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ax,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ay,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[az,dii] \
]

Lpr["
C     The dependence on delta x1 at volume nodes.
      CALL LDX1VN(LA,DFXDAX*AXD1+DFXDAY*AYD1+DFXDAZ*AZD1,
     1               DFYDAX*AXD1+DFYDAY*AYD1+DFYDAZ*AZD1,
     1               DFZDAX*AXD1+DFZDAY*AYD1+DFZDAZ*AZD1)

      CALL LDX1VN(LB,DFXDAX*AXD2+DFXDAY*AYD2+DFXDAZ*AZD2,
     1               DFYDAX*AXD2+DFYDAY*AYD2+DFYDAZ*AZD2,
     1              DFZDAX*AXD2+DFZDAY*AYD2+DFZDAZ*AZD2)

      CALL LDX1VN(LC,DFXDAX*AXD3+DFXDAY*AYD3+DFXDAZ*AZD3,
     1              DFYDAX*AXD3+DFYDAY*AYD3+DFYDAZ*AZD3,
     1              DFZDAX*AXD3+DFZDAY*AYD3+DFZDAZ*AZD3)

      CALL LDX1VN(LD,DFXDAX*AXD4+DFXDAY*AYD4+DFXDAZ*AZD4,
     1              DFYDAX*AXD4+DFYDAY*AYD4+DFYDAZ*AZD4,
     1              DFZDAX*AXD4+DFZDAY*AYD4+DFZDAZ*AZD4)

C     The dependence on delta sble for a and d.
      CALL LSBLEN(JSLEAD,NJJ,DFXDAX*AXD5+DFXDAY*AYD5+DFXDAZ*AZD5,
     1                       DFYDAX*AXD5+DFYDAY*AYD5+DFYDAZ*AZD5,
     1                       DFZDAX*AXD5+DFZDAY*AYD5+DFZDAZ*AZD5)

C     The dependence on delta sble for b and c.
      CALL LSBLEN(JSLEBC,NJJ,DFXDAX*AXD6+DFXDAY*AYD6+DFXDAZ*AZD6,
     1                       DFYDAX*AXD6+DFYDAY*AYD6+DFYDAZ*AZD6,
     1                       DFZDAX*AXD6+DFZDAY*AYD6+DFZDAZ*AZD6)

",filename]

Lpr["

      RETURN
      END
",filename]

/* Write the declaration file. */

<"decfile.in"

decfile["Lancdec.sgf"]
```

## File PDSI.IN

```
/* This session creates the subroutine which calculate the physical derivatives
of psi1 and psi2 given the Jacobian matrix and the grid derivatives of psi1
and psi2.  The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy pdsi.sgf;* pdsi.osf
!dele pdsi.sgf;*
!@cgf pdsi.sgf

filename : "pdsi.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RPDSI(IND,SI1V,SI2V,SI3V)
C     Right Physical Derivatives of pSI1 and psi2.
C
C     This routine calculates the physical derivatives of psi1 and psi2.
C
C     IND is the face and psi indicator.
C          IND         face      psi
C           1           A         1
C           2           B         1
C           3           S         1
C           4           A         2
```

```
C          5            B          2
C          6            S          2
C
      INTEGER IND
      REAL SI1V,SI2V,SI3V

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RPDSIDEC.FOR'

C     Declare the elements of the Jacobian matrix which are fixed.
      REAL D11F,D21F,D31F,D12F,D22F,D32F,D13F,D23F,D33F
C     Declare the elements of the Jacobian matrix which are moving.
      REAL D12M,D13M
C     Declare the grid derivatives of psi.
      REAL SIG1,SIG2,SIG3
C     Declare the leading edge derivative wrt xi2.  Also set to 0.
      REAL SLE2
      DATA SLE2/0./
C     Declare the leading edge value.  Also set to 0.
      REAL SBLE
      DATA SBLE/0./
C     Declare the leading edge grid derivatives.
      REAL D11LE,D21LE,D31LE,D12LE,D22LE,D32LE,D13LE,D23LE,D33LE

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the elements of the Jacobian matrix and the grid derivatives of psi.
      IF(IND.EQ.1)THEN
        CALL RCD1A(SIG1,SIG2,SIG3)
        CALL FGDA1(X1A,D11F,D12F,D13F)
        CALL FGDA1(X2A,D21F,D22F,D23F)
        CALL FGDA1(X3A,D31F,D32F,D33F)
        CALL FGDA1(X1LEA,D11LE,D12LE,D13LE)
        CALL FGDA1(X2LEA,D21LE,D22LE,D23LE)
        CALL FGDA1(X3LEA,D31LE,D32LE,D33LE)
        CALL RMGDA1(D12M,D13M)
        IF(IFTYP.EQ.3)THEN
          D12F = J42D
          D22F = J52D
          D32F = J62D
        END IF
      ELSE IF(IND.EQ.2)THEN
        CALL RCD1B(SIG1,SIG2,SIG3)
        CALL FGDB1(X1A,D11F,D12F,D13F)
        CALL FGDB1(X2A,D21F,D22F,D23F)
        CALL FGDB1(X3A,D31F,D32F,D33F)
        CALL FGDB1(X1LEA,D11LE,D12LE,D13LE)
        CALL FGDB1(X2LEA,D21LE,D22LE,D23LE)
        CALL FGDB1(X3LEA,D31LE,D32LE,D33LE)
        CALL RMGDB1(D12M,D13M)
      ELSE IF(IND.EQ.3)THEN
        CALL RCD1S(SIG1,SIG2,SIG3)
        CALL FGDS1(X1A,D11F,D12F,D13F)
        CALL FGDS1(X2A,D21F,D22F,D23F)
        CALL FGDS1(X3A,D31F,D32F,D33F)
        CALL FGDS1(X1LEA,D11LE,D12LE,D13LE)
        CALL FGDS1(X2LEA,D21LE,D22LE,D23LE)
        CALL FGDS1(X3LEA,D31LE,D32LE,D33LE)
        CALL RMGDS1(D12M,D13M)
        IF(IFTYP.EQ.9)THEN
          D12F = J42D
          D22F = J52D
          D32F = J62D
        END IF
      ELSE IF(IND.EQ.4)THEN
        CALL RCD2A(SIG1,SIG2,SIG3)
        CALL FGDA2(X1B,D11F,D12F,D13F)
        CALL FGDA2(X2B,D21F,D22F,D23F)
        CALL FGDA2(X3B,D31F,D32F,D33F)
        CALL FGDA2(X1LEB,D11LE,D12LE,D13LE)
        CALL FGDA2(X2LEB,D21LE,D22LE,D23LE)
        CALL FGDA2(X3LEB,D31LE,D32LE,D33LE)
        CALL RMGDA2(D12M,D13M)
      ELSE IF(IND.EQ.5)THEN
        CALL RCD2B(SIG1,SIG2,SIG3)
        CALL FGDB2(X1B,D11F,D12F,D13F)
        CALL FGDB2(X2B,D21F,D22F,D23F)
        CALL FGDB2(X3B,D31F,D32F,D33F)
        CALL FGDB2(X1LEB,D11LE,D12LE,D13LE)
        CALL FGDB2(X2LEB,D21LE,D22LE,D23LE)
        CALL FGDB2(X3LEB,D31LE,D32LE,D33LE)
        CALL RMGDB2(D12M,D13M)
        IF(IFTYP.EQ.3)THEN
          D11F = J12D
          D21F = J22D
          D31F = J32D
        END IF
      ELSE IF(IND.EQ.6)THEN
        CALL RCD2S(SIG1,SIG2,SIG3)
        CALL FGDS2(X1B,D11F,D12F,D13F)
        CALL FGDS2(X2B,D21F,D22F,D23F)
        CALL FGDS2(X3B,D31F,D32F,D33F)
        CALL FGDS2(X1LEB,D11LE,D12LE,D13LE)
        CALL FGDS2(X2LEB,D21LE,D22LE,D23LE)
        CALL FGDS2(X3LEB,D31LE,D32LE,D33LE)
        CALL RMGDS2(D12M,D13M)
        IF(IFTYP.EQ.9)THEN
          D11F = J12D
          D21F = J22D
          D31F = J32D
        END IF
      END IF

",filename]

Lpr["C     Add the fixed and moving terms of J4 and J7.",filename]

d1le : (d1dlea + d1dleb + d1dlec + d1dled)/4
fort["d x1/dsble",'d1le,"pdsi.sgf"]

d2le : (d2dlea + d2dleb + d2dlec + d2dled)/4
fort["d x2/dsble",'d2le,"pdsi.sgf"]

d3le : (d3dlea + d3dleb + d3dlec + d3dled)/4
fort["d x3/dsble",'d3le,"pdsi.sgf"]

j1 : d11f + d11le sble
fort["J1",'j1,"pdsi.sgf"]

j2 : d21f + d21le sble
fort["J2",'j2,"pdsi.sgf"]

j3 : d31f + d31le sble
fort["J3",'j3,"pdsi.sgf"]
```

```
j4 : d12f + d12m + d12le sble + d1le sle2
fort["J4",'j4,"pdsi.sgf"]

j5 : d22f + d22le sble + d2le sle2
fort["J5",'j5,"pdsi.sgf"]

j6 : d32f + d32le sble + d3le sle2
fort["J6",'j6,"pdsi.sgf"]

j7 : d13f + d13m + d13le sble
fort["J7",'j7,"pdsi.sgf"]

j8 : d23f + d23le sble
fort["J8",'j8,"pdsi.sgf"]

j9 : d33f + d33le sble
fort["J9",'j9,"pdsi.sgf"]

jac : j1 j5 j9 + j2 j6 j7 + j4 j8 j3 - j3 j5 j7 - j1 j6 j8 - j2 j4 j9
fort["Jacobian",'jac,"pdsi.sgf"]

Lpr["C     Calculate the Jacobian matrix inverse.",filename]

ji1 : (j5 j9 - j6 j8)/jac
fort["Dxi1/dx1",'ji1,"pdsi.sgf"]

ji2 : -(j4 j9 - j6 j7)/jac
fort["Dxi1/dx2",'ji2,"pdsi.sgf"]

ji3 : (j4 j8 - j5 j7)/jac
fort["Dxi1/dx3",'ji3,"pdsi.sgf"]

ji4 : -(j2 j9 - j3 j8)/jac
fort["Dxi2/dx1",'ji4,"pdsi.sgf"]

ji5 : (j1 j9 - j3 j7)/jac
fort["Dxi2/dx2",'ji5,"pdsi.sgf"]

ji6 : -(j1 j8 - j2 j7)/jac
fort["Dxi2/dx3",'ji6,"pdsi.sgf"]

ji7 : (j2 j6 - j3 j5)/jac
fort["Dxi3/dx1",'ji7,"pdsi.sgf"]

ji8 : -(j1 j6 - j3 j4)/jac
fort["Dxi3/dx2",'ji8,"pdsi.sgf"]

ji9 : (j1 j5 - j2 j4)/jac
fort["Dxi3/dx3",'ji9,"pdsi.sgf"]

Lpr["C     The physical derivatives.",filename]

si1 : sig1 ji1 + sig2 ji4 + sig3 ji7
fort["X1 derivative of psi.",'si1,"pdsi.sgf"]

si2 : sig1 ji2 + sig2 ji5 + sig3 ji8
fort["X2 derivative of psi.",'si2,"pdsi.sgf"]

si3 : sig1 ji3 + sig2 ji6 + sig3 ji9
fort["X3 derivative of psi.",'si3,"pdsi.sgf"]

Lpr["
      SI1V = SI1
      SI2V = SI2
      SI3V = SI3
      RETURN
      END

",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rpdsidec.sgf;* Rpdsidec.osf
!dele Rpdsidec.sgf;*
!@cgf Rpdsidec.sgf

<"decfile.in"

decfile["Rpdsidec.sgf"]

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lpdsidec.sgf;* Lpdsidec.osf
!dele Lpdsidec.sgf;*
!@cgf Lpdsidec.sgf

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "pdsi.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LPDSI(IND,DFXD1,DFXD2,DFXD3,
     1                     DFYD1,DFYD2,DFYD3,
     1                     DFZD1,DFZD2,DFZD3)
C     Left Physical Derivatives of pSI1 and psi2.
C
C     This routine calculates the contribution of the physical derivatives
C     of psi1 and psi2 to the left hand sides of the x, y and z momentum eqs.
C
C     IND is the face and psi indicator.
C          IND         face      psi
C           1           A         1
C           2           B         1
C           3           S         1
C           4           A         2
C           5           B         2
C           6           S         2
C
      INTEGER IND
      REAL DFXD1,DFXD2,DFXD3
      REAL DFYD1,DFYD2,DFYD3
      REAL DFZD1,DFZD2,DFZD3

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LPDSIDEC.FOR'
      REAL DERI

C     Declare the elements of the Jacobian matrix which are fixed.
      REAL D11F,D21F,D31F,D12F,D22F,D32F,D13F,D23F,D33F
C     Declare the elements of the Jacobian matrix which are moving.
```

```
      REAL D12M,D13M
C     Declare the grid derivatives of psi.
      REAL SIG1,SIG2,SIG3

C     Declare the leading edge derivative wrt xi2.  Also set to 0.
      REAL SLE2
      DATA SLE2/0./
C     Declare the leading edge value.  Also set to 0.
      REAL SBLE
      DATA SBLE/0./
C     Declare the leading edge grid derivatives.
      REAL D11LE,D21LE,D31LE,D12LE,D22LE,D32LE,D13LE,D23LE,D33LE

C     Moving grid derivative changes.
      REAL DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13

C     Grid derivatives of psi changes.
      REAL DFXDG1,DFXDG2,DFXDG3
      REAL DFYDG1,DFYDG2,DFYDG3
      REAL DFZDG1,DFZDG2,DFZDG3

C     Leading edge derivative changes.
      REAL DFXDL2,DFYDL2,DFZDL2

C     Leading edge changes.
      REAL DFXDLE,DFYDLE,DFZDLE

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {d12m,d13m,sig1,sig2,sig3,sle2,sble}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_d11f[Const] : 1 ; _d21f[Const] : 1 ; _d31f[Const] : 1
_d12f[Const] : 1 ; _d22f[Const] : 1 ; _d32f[Const] : 1
_d13f[Const] : 1 ; _d23f[Const] : 1 ; _d33f[Const] : 1
_d11le[Const] : 1 ; _d21le[Const] : 1 ; _d31le[Const] : 1
_d12le[Const] : 1 ; _d22le[Const] : 1 ; _d32le[Const] : 1
_d13le[Const] : 1 ; _d23le[Const] : 1 ; _d33le[Const] : 1
_d12m[Const] : 1 ; _d13m[Const] : 1
_sig1[Const] : 1 ; _sig2[Const] : 1 ; _sig3[Const] : 1
_sle2[Const] : 1 ; _sble[Const] : 1
_d1dlea[Const] : 1 ; _d1dleb[Const] : 1
_d1dlec[Const] : 1 ; _d1dled[Const] : 1
_d2dlea[Const] : 1 ; _d2dleb[Const] : 1
_d2dlec[Const] : 1 ; _d2dled[Const] : 1
_d3dlea[Const] : 1 ; _d3dleb[Const] : 1
_d3dlec[Const] : 1 ; _d3dled[Const] : 1

/* Algorithm */

Lpr["

C      Get the elements of the Jacobian matrix and the grid derivatives of psi.
       IF(IND.EQ.1)THEN
         CALL RCD1A(SIG1,SIG2,SIG3)
         CALL FGDA1(X1A,D11F,D12F,D13F)
         CALL FGDA1(X2A,D21F,D22F,D23F)
         CALL FGDA1(X3A,D31F,D32F,D33F)
         CALL FGDA1(X1LEA,D11LE,D12LE,D13LE)
         CALL FGDA1(X2LEA,D21LE,D22LE,D23LE)
```

```
  CALL FGDA1(X3LEA,D31LE,D32LE,D33LE)
  CALL RMGDA1(D12M,D13M)
  IF(IFTYP.EQ.3)THEN
    D12F = J42D
    D22F = J52D
    D32F = J62D
  END IF
ELSE IF(IND.EQ.2)THEN
  CALL RCD1B(SIG1,SIG2,SIG3)
  CALL FGDB1(X1A,D11F,D12F,D13F)
  CALL FGDB1(X2A,D21F,D22F,D23F)
  CALL FGDB1(X3A,D31F,D32F,D33F)
  CALL FGDB1(X1LEA,D11LE,D12LE,D13LE)
  CALL FGDB1(X2LEA,D21LE,D22LE,D23LE)
  CALL FGDB1(X3LEA,D31LE,D32LE,D33LE)
  CALL RMGDB1(D12M,D13M)
ELSE IF(IND.EQ.3)THEN
  CALL RCD1S(SIG1,SIG2,SIG3)
  CALL FGDS1(X1A,D11F,D12F,D13F)
  CALL FGDS1(X2A,D21F,D22F,D23F)
  CALL FGDS1(X3A,D31F,D32F,D33F)
  CALL FGDS1(X1LEA,D11LE,D12LE,D13LE)
  CALL FGDS1(X2LEA,D21LE,D22LE,D23LE)
  CALL FGDS1(X3LEA,D31LE,D32LE,D33LE)
  CALL RMGDS1(D12M,D13M)
  IF(IFTYP.EQ.9)THEN
    D12F = J42D
    D22F = J52D
    D32F = J62D
  END IF
ELSE IF(IND.EQ.4)THEN
  CALL RCD2A(SIG1,SIG2,SIG3)
  CALL FGDA2(X1B,D11F,D12F,D13F)
  CALL FGDA2(X2B,D21F,D22F,D23F)
  CALL FGDA2(X3B,D31F,D32F,D33F)
  CALL FGDA2(X1LEB,D11LE,D12LE,D13LE)
  CALL FGDA2(X2LEB,D21LE,D22LE,D23LE)
  CALL FGDA2(X3LEB,D31LE,D32LE,D33LE)
  CALL RMGDA2(D12M,D13M)
ELSE IF(IND.EQ.5)THEN
  CALL RCD2B(SIG1,SIG2,SIG3)
  CALL FGDB2(X1B,D11F,D12F,D13F)
  CALL FGDB2(X2B,D21F,D22F,D23F)
  CALL FGDB2(X3B,D31F,D32F,D33F)
  CALL FGDB2(X1LEB,D11LE,D12LE,D13LE)
  CALL FGDB2(X2LEB,D21LE,D22LE,D23LE)
  CALL FGDB2(X3LEB,D31LE,D32LE,D33LE)
  CALL RMGDB2(D12M,D13M)
  IF(IFTYP.EQ.3)THEN
    D11F = J12D
    D21F = J22D
    D31F = J32D
  END IF
ELSE IF(IND.EQ.6)THEN
  CALL RCD2S(SIG1,SIG2,SIG3)
  CALL FGDS2(X1B,D11F,D12F,D13F)
  CALL FGDS2(X2B,D21F,D22F,D23F)
  CALL FGDS2(X3B,D31F,D32F,D33F)
  CALL FGDS2(X1LEB,D11LE,D12LE,D13LE)
  CALL FGDS2(X2LEB,D21LE,D22LE,D23LE)
  CALL FGDS2(X3LEB,D31LE,D32LE,D33LE)
  CALL RMGDS2(D12M,D13M)
  IF(IFTYP.EQ.9)THEN
    D11F = J12D
    D21F = J22D
    D31F = J32D
```

```
        END IF
      END IF

",filename]

Lpr["C     Add the fixed and moving terms of J4 and J7.",filename]

d1le : (d1d1lea + d1d1leb + d1d1lec + d1d1led)/4
deriv["d x1/dsble",'d1le,"pdsi.sgf"]

d2le : (d2d1lea + d2d1leb + d2d1lec + d2d1led)/4
deriv["d x2/dsble",'d2le,"pdsi.sgf"]

d3le : (d3d1lea + d3d1leb + d3d1lec + d3d1led)/4
deriv["d x3/dsble",'d3le,"pdsi.sgf"]

j1 : d11f + d11le sble
deriv["J1",'j1,"pdsi.sgf"]

j2 : d21f + d21le sble
deriv["J2",'j2,"pdsi.sgf"]

j3 : d31f + d31le sble
deriv["J3",'j3,"pdsi.sgf"]

j4 : d12f + d12m + d12le sble + d1le sle2
deriv["J4",'j4,"pdsi.sgf"]

j5 : d22f + d22le sble + d2le sle2
deriv["J5",'j5,"pdsi.sgf"]

j6 : d32f + d32le sble + d3le sle2
deriv["J6",'j6,"pdsi.sgf"]

j7 : d13f + d13m + d13le sble
deriv["J7",'j7,"pdsi.sgf"]

j8 : d23f + d23le sble
deriv["J8",'j8,"pdsi.sgf"]

j9 : d33f + d33le sble
deriv["J9",'j9,"pdsi.sgf"]

jac : j1 j5 j9 + j2 j6 j7 + j4 j8 j3 - j3 j5 j7 - j1 j6 j8 - j2 j4 j9
deriv["Jacobian",'jac,"pdsi.sgf"]

Lpr["C     Calculate the Jacobian matrix inverse.",filename]

ji1 : (j5 j9 - j6 j8)/jac
deriv["Dxi1/dx1",'ji1,"pdsi.sgf"]

ji2 : -(j4 j9 - j6 j7)/jac
deriv["Dxi1/dx2",'ji2,"pdsi.sgf"]

ji3 : (j4 j8 - j5 j7)/jac
deriv["Dxi1/dx3",'ji3,"pdsi.sgf"]

ji4 : -(j2 j9 - j3 j8)/jac
deriv["Dxi2/dx1",'ji4,"pdsi.sgf"]

ji5 : (j1 j9 - j3 j7)/jac
deriv["Dxi2/dx2",'ji5,"pdsi.sgf"]

ji6 : -(j1 j8 - j2 j7)/jac
deriv["Dxi2/dx3",'ji6,"pdsi.sgf"]
```

```
ji7 : (j2 j6 - j3 j5)/jac
deriv["Dxi3/dx1",'ji7,"pdsi.sgf"]

ji8 : -(j1 j6 - j3 j4)/jac
deriv["Dxi3/dx2",'ji8,"pdsi.sgf"]

ji9 : (j1 j5 - j2 j4)/jac
deriv["Dxi3/dx3",'ji9,"pdsi.sgf"]

Lpr["C     The physical derivatives.",filename]

si1 : sig1 ji1 + sig2 ji4 + sig3 ji7
deriv["X1 derivative of psi.",'si1,"pdsi.sgf"]

si2 : sig1 ji2 + sig2 ji5 + sig3 ji8
deriv["X2 derivative of psi.",'si2,"pdsi.sgf"]

si3 : sig1 ji3 + sig2 ji6 + sig3 ji9
deriv["X3 derivative of psi.",'si3,"pdsi.sgf"]

/* Include in exprelist the dependence of si1, si2, and si3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[si1,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[si2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[si3,dii] \
]

/* Memory managment */
Lpr[Mem[]]
exprlist : Union[exprlist]
nel : Len[exprlist]
Gc[]
Lpr[Mem[]]

Lpr["
C     The dependence on the moving elements of the Jacobian matrix and the
C     grid derivatives of psi.
      DFXD12 = DFXD1*SI1D1 + DFXD2*SI2D1 + DFXD3*SI3D1
      DFXD13 = DFXD1*SI1D2 + DFXD2*SI2D2 + DFXD3*SI3D2
      DFYD12 = DFYD1*SI1D1 + DFYD2*SI2D1 + DFYD3*SI3D1
      DFYD13 = DFYD1*SI1D2 + DFYD2*SI2D2 + DFYD3*SI3D2
      DFZD12 = DFZD1*SI1D1 + DFZD2*SI2D1 + DFZD3*SI3D1
      DFZD13 = DFZD1*SI1D2 + DFZD2*SI2D2 + DFZD3*SI3D2

      DFXDG1 = DFXD1*SI1D3 + DFXD2*SI2D3 + DFXD3*SI3D3
      DFXDG2 = DFXD1*SI1D4 + DFXD2*SI2D4 + DFXD3*SI3D4
      DFXDG3 = DFXD1*SI1D5 + DFXD2*SI2D5 + DFXD3*SI3D5

      DFYDG1 = DFYD1*SI1D3 + DFYD2*SI2D3 + DFYD3*SI3D3
      DFYDG2 = DFYD1*SI1D4 + DFYD2*SI2D4 + DFYD3*SI3D4
      DFYDG3 = DFYD1*SI1D5 + DFYD2*SI2D5 + DFYD3*SI3D5

      DFZDG1 = DFZD1*SI1D3 + DFZD2*SI2D3 + DFZD3*SI3D3
      DFZDG2 = DFZD1*SI1D4 + DFZD2*SI2D4 + DFZD3*SI3D4
      DFZDG3 = DFZD1*SI1D5 + DFZD2*SI2D5 + DFZD3*SI3D5

      DFXDL2 = DFXD1*SI1D6 + DFXD2*SI2D6 + DFXD3*SI3D6
      DFYDL2 = DFYD1*SI1D6 + DFYD2*SI2D6 + DFYD3*SI3D6
      DFZDL2 = DFZD1*SI1D6 + DFZD2*SI2D6 + DFZD3*SI3D6

      DFXDLE = DFXD1*SI1D7 + DFXD2*SI2D7 + DFXD3*SI3D7
      DFYDLE = DFYD1*SI1D7 + DFYD2*SI2D7 + DFYD3*SI3D7
      DFZDLE = DFZD1*SI1D7 + DFZD2*SI2D7 + DFZD3*SI3D7
      IF(IND.EQ.1)THEN
        CALL LMGDA1(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
        CALL LCD1A(DFXDG1,DFXDG2,DFXDG3,DFYDG1,DFYDG2,DFYDG3,
     1             DFZDG1,DFZDG2,DFZDG3)
        CALL LLEDA1(DFXDL2,DFYDL2,DFZDL2)
      ELSE IF(IND.EQ.2)THEN
        CALL LMGDB1(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
        CALL LCD1B(DFXDG1,DFXDG2,DFXDG3,DFYDG1,DFYDG2,DFYDG3,
     1             DFZDG1,DFZDG2,DFZDG3)
        CALL LLEDB1(DFXDL2,DFYDL2,DFZDL2)
      ELSE IF(IND.EQ.3)THEN
        CALL LMGDS1(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
        CALL LCD1S(DFXDG1,DFXDG2,DFXDG3,DFYDG1,DFYDG2,DFYDG3,
     1             DFZDG1,DFZDG2,DFZDG3)
        CALL LLEDA1(DFXDL2,DFYDL2,DFZDL2)
      ELSE IF(IND.EQ.4)THEN
        CALL LMGDA2(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
        CALL LCD2A(DFXDG1,DFXDG2,DFXDG3,DFYDG1,DFYDG2,DFYDG3,
     1             DFZDG1,DFZDG2,DFZDG3)
        CALL LLEDA2(DFXDL2,DFYDL2,DFZDL2)
      ELSE IF(IND.EQ.5)THEN
        CALL LMGDB2(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
        CALL LCD2B(DFXDG1,DFXDG2,DFXDG3,DFYDG1,DFYDG2,DFYDG3,
     1             DFZDG1,DFZDG2,DFZDG3)
        CALL LLEDB2(DFXDL2,DFYDL2,DFZDL2)
      ELSE IF(IND.EQ.6)THEN
        CALL LMGDS2(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
        CALL LCD2S(DFXDG1,DFXDG2,DFXDG3,DFYDG1,DFYDG2,DFYDG3,
     1             DFZDG1,DFZDG2,DFZDG3)
        CALL LLEDA2(DFXDL2,DFYDL2,DFZDL2)
      END IF

C     Put half the leading edge change on a-d and half on b-c.
      CALL LSBLEN(JSLEAD,NJJ,0.5*DFXDLE,0.5*DFYDLE,0.5*DFZDLE)
      CALL LSBLEN(JSLEBC,NJJ,0.5*DFXDLE,0.5*DFYDLE,0.5*DFZDLE)

",filename]

Lpr["

      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["Lpdsidec.sgf"]
```

## File MGDA1.IN

```
/* This session creates the subroutine which calculates the moving grid
derivatives in xi2, and xi3 on an A face.  The left hand side subroutine
is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mgda1.sgf;* mgda1.osf
!dele mgda1.sgf;*
!@cgf mgda1.sgf
```

```
filename : "mgda1.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMGDA1(D12MV,D13MV)
C     Right Moving Grid Derivatives for an A face for psi1 calculations.
C
C     This routine calculates the moving grid derivatives
C     with respect to xi2 and xi3 on an A face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3.
C     This routine is called indirectly by RMA or LMA so the variables
C     in common are already initialized.
C
      REAL D12MV,D13MV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMGDA1DEC.FOR'

      IF(MGD.EQ.-1)THEN
        D12MV = 0.
        D13MV = 0.
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

ad2case : {(dx16 - dx14)/2,\
           (-3dx15 + 4dx16 - dx14)/2,\
           (3dx15 - 4dx14 + dx16)/2,\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2",\
          "MGD.EQ.3"}
casefort[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgda1.sgf"]

ad3 : (dx18 - dx12)/2
fort["D(delta theta)/Dxi3",'ad3,"mgda1.sgf"]

Lpr["
      D12MV = AD2
      D13MV = AD3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmgda1dec.sgf;* Rmgda1dec.osf
!dele Rmgda1dec.sgf;*
```

```
!@cgf Rmgda1dec.sgf

<"decfile.in"

decfile["Rmgda1dec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mgda1.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMGDA1(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
C     Left Moving Grid Derivatives for an A face for psi1 calculations.
C
C     This routine calculates the contribution of the moving grid derivatives
C     with respect to xi2 and xi3 on an A face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3 to the x, y, and z momentum eqs.
C     This routine is called indirectly by LMA so the variables
C     in common are already initialized.
C
      REAL DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMGDA1DEC.FOR'
      REAL DERI
      REAL COEF

      IF(MGD.EQ.-1)THEN
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx11,dx12,dx13,dx14,dx15,dx16,dx17,dx18,dx19}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx11[Const] : 1 ; _dx12[Const] : 1 ; _dx13[Const] : 1
_dx14[Const] : 1 ; _dx15[Const] : 1 ; _dx16[Const] : 1
_dx17[Const] : 1 ; _dx18[Const] : 1 ; _dx19[Const] : 1

/* Algorithm */

ad2case : {(dx16 - dx14)/2,\
           (-3dx15 + 4dx16 - dx14)/2,\
           (3dx15 - 4dx14 + dx16)/2,\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2",\
          "MGD.EQ.3"}
casederiv[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgda1.sgf"]

ad3 : (dx18 - dx12)/2
deriv["D(delta theta)/Dxi3",'ad3,"mgda1.sgf"]
```

```
/* Include in exprelist the dependence of ad2 and ad3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad3,dii]; \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C


C
C     delta theta at 1
      COEF = DFXD12*AD2D1 + DFXD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 2
      COEF = DFXD12*AD2D2 + DFXD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 3
      COEF = DFXD12*AD2D3 + DFXD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 4
      COEF = DFXD12*AD2D4 + DFXD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 5
      COEF = DFXD12*AD2D5 + DFXD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 6
      COEF = DFXD12*AD2D6 + DFXD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 7
      COEF = DFXD12*AD2D7 + DFXD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 8
      COEF = DFXD12*AD2D8 + DFXD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 9
      COEF = DFXD12*AD2D9 + DFXD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
```

```
C

C
C     delta theta at 1
      COEF = DFYD12*AD2D1 + DFYD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 2
      COEF = DFYD12*AD2D2 + DFYD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 3
      COEF = DFYD12*AD2D3 + DFYD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 4
      COEF = DFYD12*AD2D4 + DFYD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 5
      COEF = DFYD12*AD2D5 + DFYD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 6
      COEF = DFYD12*AD2D6 + DFYD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 7
      COEF = DFYD12*AD2D7 + DFYD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 8
      COEF = DFYD12*AD2D8 + DFYD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 9
      COEF = DFYD12*AD2D9 + DFYD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     delta theta at 1
      COEF = DFZD12*AD2D1 + DFZD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 2
      COEF = DFZD12*AD2D2 + DFZD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 3
      COEF = DFZD12*AD2D3 + DFZD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 4
      COEF = DFZD12*AD2D4 + DFZD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFZ,ICOLZ,NCZ)
```

```
C
C     delta theta at 5
      COEF = DFZD12*AD2D5 + DFZD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 6
      COEF = DFZD12*AD2D6 + DFZD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 7
      COEF = DFZD12*AD2D7 + DFZD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 8
      COEF = DFZD12*AD2D8 + DFZD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 9
      COEF = DFZD12*AD2D9 + DFZD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmgda1dec.sgf;* Lmgda1dec.osf
!dele Lmgda1dec.sgf;*
!@cgf Lmgda1dec.sgf

<"decfile.in"

decfile["Lmgda1dec.sgf"]
```

## File MGDB1.IN

```
/* This session creates the subroutine which calculates the moving grid
derivatives in xi2, and xi3 on a B face.  The left hand side subroutine
is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mgdb1.sgf;* mgdb1.osf
!dele mgdb1.sgf;*
!@cgf mgdb1.sgf

filename : "mgdb1.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMGDB1(D12MV,D13MV)
C     Right Moving Grid Derivatives for a B face for psi1 calculations.
C
C     This routine calculates the moving grid derivatives
C     with respect to xi2 and xi3 on a B face d(delta theta)/dxi2 and
```

```
C     d(delta theta)/dxi3.
C     This routine is called indirectly by RMB or LMB so the variables
C     in common are already initialized.
C
      REAL D12MV,D13MV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMGDB1DEC.FOR'

      IF(MGD.EQ.-1)THEN
        D12MV = 0.
        D13MV = 0.
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

ad2 : dx14 - dx13
fort["D(delta theta)/Dxi2",'ad2,"mgdb1.sgf"]

ad3 : ((dx15 + dx16) - (dx11 + dx12))/4
fort["D(delta theta)/Dxi3",'ad3,"mgdb1.sgf"]

Lpr["
      D12MV = AD2
      D13MV = AD3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmgdb1dec.sgf;* Rmgdb1dec.osf
!dele Rmgdb1dec.sgf;*
!@cgf Rmgdb1dec.sgf

<"decfile.in"

decfile["Rmgdb1dec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mgdb1.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMGDB1(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
C     Left Moving Grid Derivatives for a B face for psi1 calculations.
```

```
C
C     This routine calculates the contribution of the moving grid derivatives
C     with respect to xi2 and xi3 on a B face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3 to the x, y, and z momentum eqs.
C     This routine is called indirectly by LMB so the variables
C     in common are already initialized.
C
      REAL DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMGDB1DEC.FOR'
      REAL DERI
      REAL COEF

      IF(MGD.EQ.-1)THEN
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx11,dx12,dx13,dx14,dx15,dx16,dx17,dx18,dx19}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx11[Const] : 1 ; _dx12[Const] : 1 ; _dx13[Const] : 1
_dx14[Const] : 1 ; _dx15[Const] : 1 ; _dx16[Const] : 1
_dx17[Const] : 1 ; _dx18[Const] : 1 ; _dx19[Const] : 1

/* Algorithm */

ad2 : dx14 - dx13
deriv["D(delta theta)/Dxi2",'ad2,"mgdb1.sgf"]

ad3 : ((dx15 + dx16) - (dx11 + dx12))/4
deriv["D(delta theta)/Dxi3",'ad3,"mgdb1.sgf"]

/* Include in exprelist the dependence of ad2 and ad3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad3,dii]; \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     delta theta at 1
      COEF = DFXD12*AD2D1 + DFXD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 2
```

```
      COEF = DFXD12*AD2D2 + DFXD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 3
      COEF = DFXD12*AD2D3 + DFXD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 4
      COEF = DFXD12*AD2D4 + DFXD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 5
      COEF = DFXD12*AD2D5 + DFXD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 6
      COEF = DFXD12*AD2D6 + DFXD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 7
      COEF = DFXD12*AD2D7 + DFXD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 8
      COEF = DFXD12*AD2D8 + DFXD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 9
      COEF = DFXD12*AD2D9 + DFXD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     delta theta at 1
      COEF = DFYD12*AD2D1 + DFYD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 2
      COEF = DFYD12*AD2D2 + DFYD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 3
      COEF = DFYD12*AD2D3 + DFYD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 4
      COEF = DFYD12*AD2D4 + DFYD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 5
      COEF = DFYD12*AD2D5 + DFYD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 6
      COEF = DFYD12*AD2D6 + DFYD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFY,ICOLY,NCY)
```

```
C
C     delta theta at 7
      COEF = DFYD12*AD2D7 + DFYD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 8
      COEF = DFYD12*AD2D8 + DFYD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 9
      COEF = DFYD12*AD2D9 + DFYD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     delta theta at 1
      COEF = DFZD12*AD2D1 + DFZD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 2
      COEF = DFZD12*AD2D2 + DFZD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 3
      COEF = DFZD12*AD2D3 + DFZD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 4
      COEF = DFZD12*AD2D4 + DFZD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 5
      COEF = DFZD12*AD2D5 + DFZD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 6
      COEF = DFZD12*AD2D6 + DFZD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 7
      COEF = DFZD12*AD2D7 + DFZD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 8
      COEF = DFZD12*AD2D8 + DFZD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 9
      COEF = DFZD12*AD2D9 + DFZD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
```

```
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmgdb1dec.sgf;* Lmgdb1dec.osf
!dele Lmgdb1dec.sgf;*
!@cgf Lmgdb1dec.sgf

<"decfile.in"

decfile["Lmgdb1dec.sgf"]
```

## File MGDS1.IN

```
/* This session creates the subroutine which calculates the moving grid
derivatives in xi2, and xi3 on an S face.  The left hand side subroutine
is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mgds1.sgf;* mgds1.osf
!dele mgds1.sgf;*
!@cgf mgds1.sgf

filename : "mgds1.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMGDS1(D12MV,D13MV)
C     Right Moving Grid Derivatives for an S face for psi1 calculations.
C
C     This routine calculates the moving grid derivatives
C     with respect to xi2 and xi3 on an S face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3.
C     This routine is called indirectly by RMS or LMS so the variables
C     in common are already initialized.
C
      REAL D12MV,D13MV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMGDS1DEC.FOR'

      IF(MGD.EQ.-1)THEN
        D12MV = 0.
        D13MV = 0.
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

ad2case : {((dx13 + dx16) - (dx11 + dx14))/4,\
          (-3(dx12 + dx15) + 4(dx13 + dx16) - (dx11 + dx14))/4,\
          (3(dx12 + dx15) - 4(dx11 + dx14) + (dx13 + dx16))/4,\
          0}
```

```
ad2cond : {"MGD.EQ.0",\
           "MGD.EQ.1",\
           "MGD.EQ.2",\
           "MGD.EQ.3"}
casefort[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgds1.sgf"]

ad3 : dx15 - dx12
fort["D(delta theta)/Dxi3",'ad3,"mgds1.sgf"]

Lpr["
      D12MV = AD2
      D13MV = AD3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmgds1dec.sgf;* Rmgds1dec.osf
!dele Rmgds1dec.sgf;*
!@cgf Rmgds1dec.sgf

<"decfile.in"

decfile["Rmgds1dec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mgds1.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMGDS1(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
C     Left Moving Grid Derivatives for an S face for psi1 calculations.
C
C     This routine calculates the contribution of the moving grid derivatives
C     with respect to xi2 and xi3 on an S face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3 to the x, y, and z momentum eqs.
C     This routine is called indirectly by LMS so the variables
C     in common are already initialized.
C
      REAL DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMGDS1DEC.FOR'
      REAL DERI
      REAL COEF

      IF(MGD.EQ.-1)THEN
        RETURN
      END IF
",filename]
```

```
/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx11,dx12,dx13,dx14,dx15,dx16,dx17,dx18,dx19}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx11[Const] : 1 ; _dx12[Const] : 1 ; _dx13[Const] : 1
_dx14[Const] : 1 ; _dx15[Const] : 1 ; _dx16[Const] : 1
_dx17[Const] : 1 ; _dx18[Const] : 1 ; _dx19[Const] : 1

/* Algorithm */

ad2case : {((dx13 + dx16) - (dx11 + dx14))/4,\
           (-3(dx12 + dx15) + 4(dx13 + dx16) - (dx11 + dx14))/4,\
           (3(dx12 + dx15) - 4(dx11 + dx14) + (dx13 + dx16))/4,\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2",\
          "MGD.EQ.3"}
casederiv[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgds1.sgf"]

ad3 : dx15 - dx12
deriv["D(delta theta)/Dxi3",'ad3,"mgds1.sgf"]

/* Include in exprelist the dependence of ad2 and ad3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad3,dii]; \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     delta theta at 1
      COEF = DFXD12*AD2D1 + DFXD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 2
      COEF = DFXD12*AD2D2 + DFXD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 3
      COEF = DFXD12*AD2D3 + DFXD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 4
      COEF = DFXD12*AD2D4 + DFXD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 5
```

```
      COEF = DFXD12*AD2D5 + DFXD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 6
      COEF = DFXD12*AD2D6 + DFXD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 7
      COEF = DFXD12*AD2D7 + DFXD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 8
      COEF = DFXD12*AD2D8 + DFXD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 9
      COEF = DFXD12*AD2D9 + DFXD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     delta theta at 1
      COEF = DFYD12*AD2D1 + DFYD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 2
      COEF = DFYD12*AD2D2 + DFYD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 3
      COEF = DFYD12*AD2D3 + DFYD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 4
      COEF = DFYD12*AD2D4 + DFYD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 5
      COEF = DFYD12*AD2D5 + DFYD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 6
      COEF = DFYD12*AD2D6 + DFYD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 7
      COEF = DFYD12*AD2D7 + DFYD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 8
      COEF = DFYD12*AD2D8 + DFYD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 9
      COEF = DFYD12*AD2D9 + DFYD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFY,ICOLY,NCY)
```

```
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     delta theta at 1
      COEF = DFZD12*AD2D1 + DFZD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 2
      COEF = DFZD12*AD2D2 + DFZD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 3
      COEF = DFZD12*AD2D3 + DFZD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 4
      COEF = DFZD12*AD2D4 + DFZD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 5
      COEF = DFZD12*AD2D5 + DFZD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 6
      COEF = DFZD12*AD2D6 + DFZD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 7
      COEF = DFZD12*AD2D7 + DFZD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 8
      COEF = DFZD12*AD2D8 + DFZD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 9
      COEF = DFZD12*AD2D9 + DFZD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmgds1dec.sgf;* Lmgds1dec.osf
!dele Lmgds1dec.sgf;*
!@cgf Lmgds1dec.sgf

<"decfile.in"
```

```
decfile["Lmgds1dec.sgf"]
```

## File MGDA2.IN

```
/* This session creates the subroutine which calculates the moving grid
derivatives in xi2, and xi3 on an A face.  The left hand side subroutine
is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mgda2.sgf;* mgda2.osf
!dele mgda2.sgf;*
!@cgf mgda2.sgf

filename : "mgda2.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMGDA2(D12MV,D13MV)
C     Right Moving Grid Derivatives for an A face for psi2 calculations.
C
C     This routine calculates the moving grid derivatives
C     with respect to xi2 and xi3 on an A face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3.
C     This routine is called indirectly by RMA or LMA so the variables
C     in common are already initialized.
C
      REAL D12MV,D13MV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMGDA2DEC.FOR'

      IF(MGD.EQ.-1)THEN
        D12MV = 0.
        D13MV = 0.
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

ad2case : {(dx16 - dx14)/2,\
           (dx16 - dx15),\
           (dx15 - dx14),\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2",\
          "MGD.EQ.3"}
casefort[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgda2.sgf"]

ad3case : {((dx17 + 2dx18 + dx19) - (dx11 + 2dx12 + dx13))/8,\
           (dx18 - dx12)/2}
ad3cond : {"MGD.EQ.0",\
          ".TRUE."}
casefort[2,"D(delta theta)/Dxi3",ad3,ad3case,ad3cond,"mgda2.sgf"]
```

```
Lpr["
      D12MV = AD2
      D13MV = AD3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmgda2dec.sgf;* Rmgda2dec.osf
!dele Rmgda2dec.sgf;*
!@cgf Rmgda2dec.sgf

<"decfile.in"

decfile["Rmgda2dec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mgda2.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMGDA2(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
C     Left Moving Grid Derivatives for an A face for psi2 calculations.
C
C     This routine calculates the contribution of the moving grid derivatives
C     with respect to xi2 and xi3 on an A face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3 to the x, y, and z momentum eqs.
C     This routine is called indirectly by LMA so the variables
C     in common are already initialized.
C
      REAL DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMGDA2DEC.FOR'
      REAL DERI
      REAL COEF

      IF(MGD.EQ.-1)THEN
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx11,dx12,dx13,dx14,dx15,dx16,dx17,dx18,dx19}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
```

```
so they are considered independent by the derivative operator. */
_dx11[Const] : 1 ; _dx12[Const] : 1 ; _dx13[Const] : 1
_dx14[Const] : 1 ; _dx15[Const] : 1 ; _dx16[Const] : 1
_dx17[Const] : 1 ; _dx18[Const] : 1 ; _dx19[Const] : 1

/* Algorithm */

ad2case : {(dx16 - dx14)/2,\
           (dx16 - dx15),\
           (dx15 - dx14),\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2",\
          "MGD.EQ.3"}
casederiv[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgda2.sgf"]

ad3case : {((dx17 + 2dx18 + dx19) - (dx11 + 2dx12 + dx13))/8,\
           (dx18 - dx12)/2}
ad3cond : {"MGD.EQ.0",\
          ".TRUE."}
casederiv[2,"D(delta theta)/Dxi3",ad3,ad3case,ad3cond,"mgda2.sgf"]

/* Include in exprelist the dependence of ad2 and ad3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad3,dii]; \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     delta theta at 1
      COEF = DFXD12*AD2D1 + DFXD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 2
      COEF = DFXD12*AD2D2 + DFXD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 3
      COEF = DFXD12*AD2D3 + DFXD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 4
      COEF = DFXD12*AD2D4 + DFXD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 5
      COEF = DFXD12*AD2D5 + DFXD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 6
      COEF = DFXD12*AD2D6 + DFXD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFX,ICOLX,NCX)
```

```
C
C     delta theta at 7
      COEF = DFXD12*AD2D7 + DFXD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 8
      COEF = DFXD12*AD2D8 + DFXD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 9
      COEF = DFXD12*AD2D9 + DFXD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     delta theta at 1
      COEF = DFYD12*AD2D1 + DFYD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 2
      COEF = DFYD12*AD2D2 + DFYD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 3
      COEF = DFYD12*AD2D3 + DFYD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 4
      COEF = DFYD12*AD2D4 + DFYD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 5
      COEF = DFYD12*AD2D5 + DFYD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 6
      COEF = DFYD12*AD2D6 + DFYD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 7
      COEF = DFYD12*AD2D7 + DFYD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 8
      COEF = DFYD12*AD2D8 + DFYD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 9
      COEF = DFYD12*AD2D9 + DFYD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
```

```
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     delta theta at 1
      COEF = DFZD12*AD2D1 + DFZD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 2
      COEF = DFZD12*AD2D2 + DFZD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 3
      COEF = DFZD12*AD2D3 + DFZD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 4
      COEF = DFZD12*AD2D4 + DFZD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 5
      COEF = DFZD12*AD2D5 + DFZD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 6
      COEF = DFZD12*AD2D6 + DFZD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 7
      COEF = DFZD12*AD2D7 + DFZD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 8
      COEF = DFZD12*AD2D8 + DFZD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 9
      COEF = DFZD12*AD2D9 + DFZD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmgda2dec.sgf;* Lmgda2dec.osf
!dele Lmgda2dec.sgf;*
!@cgf Lmgda2dec.sgf

<"decfile.in"

decfile["Lmgda2dec.sgf"]
```

## File MGDB2.IN

```
/* This session creates the subroutine which calculates the moving grid
derivatives in xi2, and xi3 on a B face.  The left hand side subroutine
is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mgdb2.sgf;* mgdb2.osf
!dele mgdb2.sgf;*
!@cgf mgdb2.sgf

filename : "mgdb2.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMGDB2(D12MV,D13MV)
C     Right Moving Grid Derivatives for a B face for psi2 calculations.
C
C     This routine calculates the moving grid derivatives
C     with respect to xi2 and xi3 on a B face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3.
C     This routine is called indirectly by RMB or LMB so the variables
C     in common are already initialized.
C
      REAL D12MV,D13MV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMGDB2DEC.FOR'

      IF(MGD.EQ.-1)THEN
        D12MV = 0.
        D13MV = 0.
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

ad2case : {(-3dx13 + 2dx14 + dx17)/4,\
           (3dx14 - 2dx13 - dx17)/4,\
           ((dx14 + dx18) - (dx13 + dx17))/4,\
           0}
ad2cond : {"MGD.EQ.0",\
           "MGD.EQ.1",\
           "MGD.EQ.2"}
casefort[3,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgdb2.sgf"]

ad3 : ((dx15 + dx16) - (dx11 + dx12))/4
fort["D(delta theta)/Dxi3",'ad3,"mgdb2.sgf"]

Lpr["
      D12MV = AD2
      D13MV = AD3
",filename]

Lpr["
C
      RETURN
      END

",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmgdb2dec.sgf;* Rmgdb2dec.osf
!dele Rmgdb2dec.sgf;*
!@cgf Rmgdb2dec.sgf

<"decfile.in"

decfile["Rmgdb2dec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mgdb2.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMGDB2(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
C     Left Moving Grid Derivatives for a B face for psi2 calculations.
C
C     This routine calculates the contribution of the moving grid derivatives
C     with respect to xi2 and xi3 on a B face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3 to the x, y, and z momentum eqs.
C     This routine is called indirectly by LMB so the variables
C     in common are already initialized.
C
      REAL DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMGDB2DEC.FOR'
      REAL DERI
      REAL COEF

      IF(MGD.EQ.-1)THEN
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx11,dx12,dx13,dx14,dx15,dx16,dx17,dx18,dx19}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx11[Const] : 1 ; _dx12[Const] : 1 ; _dx13[Const] : 1
_dx14[Const] : 1 ; _dx15[Const] : 1 ; _dx16[Const] : 1
_dx17[Const] : 1 ; _dx18[Const] : 1 ; _dx19[Const] : 1

/* Algorithm */

ad2case : {(-3dx13 + 2dx14 + dx17)/4,\
           (3dx14 - 2dx13 - dx17)/4,\
           ((dx14 + dx18) - (dx13 + dx17))/4,\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2"}
casederiv[3,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgdb2.sgf"]

ad3 : ((dx15 + dx16) - (dx11 + dx12))/4
deriv["D(delta theta)/Dxi3",'ad3,"mgdb2.sgf"]

/* Include in exprelist the dependence of ad2 and ad3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad3,dii]; \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     delta theta at 1
      COEF = DFXD12*AD2D1 + DFXD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 2
      COEF = DFXD12*AD2D2 + DFXD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 3
      COEF = DFXD12*AD2D3 + DFXD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 4
      COEF = DFXD12*AD2D4 + DFXD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 5
      COEF = DFXD12*AD2D5 + DFXD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 6
      COEF = DFXD12*AD2D6 + DFXD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 7
      COEF = DFXD12*AD2D7 + DFXD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 8
      COEF = DFXD12*AD2D8 + DFXD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 9
      COEF = DFXD12*AD2D9 + DFXD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFX,ICOLX,NCX)
```

```
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     delta theta at 1
      COEF = DFYD12*AD2D1 + DFYD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 2
      COEF = DFYD12*AD2D2 + DFYD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 3
      COEF = DFYD12*AD2D3 + DFYD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 4
      COEF = DFYD12*AD2D4 + DFYD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 5
      COEF = DFYD12*AD2D5 + DFYD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 6
      COEF = DFYD12*AD2D6 + DFYD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 7
      COEF = DFYD12*AD2D7 + DFYD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 8
      COEF = DFYD12*AD2D8 + DFYD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 9
      COEF = DFYD12*AD2D9 + DFYD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     delta theta at 1
      COEF = DFZD12*AD2D1 + DFZD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 2
```

```
       COEF = DFZD12*AD2D2 + DFZD13*AD3D2
       CALL ADCOFS(COEF,IDX12+IDTHO,COFZ,ICOLZ,NCZ)
C
C      delta theta at 3
       COEF = DFZD12*AD2D3 + DFZD13*AD3D3
       CALL ADCOFS(COEF,IDX13+IDTHO,COFZ,ICOLZ,NCZ)
C
C      delta theta at 4
       COEF = DFZD12*AD2D4 + DFZD13*AD3D4
       CALL ADCOFS(COEF,IDX14+IDTHO,COFZ,ICOLZ,NCZ)
C
C      delta theta at 5
       COEF = DFZD12*AD2D5 + DFZD13*AD3D5
       CALL ADCOFS(COEF,IDX15+IDTHO,COFZ,ICOLZ,NCZ)
C
C      delta theta at 6
       COEF = DFZD12*AD2D6 + DFZD13*AD3D6
       CALL ADCOFS(COEF,IDX16+IDTHO,COFZ,ICOLZ,NCZ)
C
C      delta theta at 7
       COEF = DFZD12*AD2D7 + DFZD13*AD3D7
       CALL ADCOFS(COEF,IDX17+IDTHO,COFZ,ICOLZ,NCZ)
C
C      delta theta at 8
       COEF = DFZD12*AD2D8 + DFZD13*AD3D8
       CALL ADCOFS(COEF,IDX18+IDTHO,COFZ,ICOLZ,NCZ)
C
C      delta theta at 9
       COEF = DFZD12*AD2D9 + DFZD13*AD3D9
       CALL ADCOFS(COEF,IDX19+IDTHO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
       RETURN
       END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmgdb2dec.sgf;* Lmgdb2dec.osf
!dele Lmgdb2dec.sgf;*
!@cgf Lmgdb2dec.sgf

<"decfile.in"

decfile["Lmgdb2dec.sgf"]
```

## File MGDS2.IN

```
/* This session creates the subroutine which calculates the moving grid
derivatives in xi2, and xi3 on an S face.  The left hand side subroutine
is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mgds2.sgf;* mgds2.osf
!dele mgds2.sgf;*
!@cgf mgds2.sgf

filename : "mgds2.sgf"
```

```
/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMGDS2(D12MV,D13MV)
C     Right Moving Grid Derivatives for an S face for psi2 calculations.
C
C     This routine calculates the moving grid derivatives
C     with respect to xi2 and xi3 on an S face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3.
C     This routine is called indirectly by RMS or LMS so the variables
C     in common are already initialized.
C
      REAL D12MV,D13MV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMGDS2DEC.FOR'

      IF(MGD.EQ.-1)THEN
        D12MV = 0.
        D13MV = 0.
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

ad2case : {((dx13 + dx16) - (dx11 + dx14))/4,\
           ((dx13 + dx16) - (dx12 + dx15))/2,\
           ((dx12 + dx15) - (dx11 + dx14))/2,\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2",\
          "MGD.EQ.3"}
casefort[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgds2.sgf"]

ad3case : {((dx14 + 2dx15 + dx16) - (dx11 + 2dx12 + dx13))/4,\
           dx15 - dx12}
ad3cond : {"MGD.EQ.0",\
          ".TRUE."}
casefort[2,"D(delta theta)/Dxi3",ad3,ad3case,ad3cond,"mgds2.sgf"]

Lpr["
      D12MV = AD2
      D13MV = AD3
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmgds2dec.sgf;* Rmgds2dec.osf
```

```
!dele Rmgds2dec.sgf;*
!@cgf Rmgds2dec.sgf

<"decfile.in"

decfile["Rmgds2dec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mgds2.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMGDS2(DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13)
C     Left Moving Grid Derivatives for an S face for psi1 calculations.
C
C     This routine calculates the contribution of the moving grid derivatives
C     with respect to xi2 and xi3 on an S face d(delta theta)/dxi2 and
C     d(delta theta)/dxi3 to the x, y, and z momentum eqs.
C     This routine is called indirectly by LMS so the variables
C     in common are already initialized.
C
      REAL DFXD12,DFXD13,DFYD12,DFYD13,DFZD12,DFZD13

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMGDS2DEC.FOR'
      REAL DERI
      REAL COEF

      IF(MGD.EQ.-1)THEN
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx11,dx12,dx13,dx14,dx15,dx16,dx17,dx18,dx19}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx11[Const] : 1 ; _dx12[Const] : 1 ; _dx13[Const] : 1
_dx14[Const] : 1 ; _dx15[Const] : 1 ; _dx16[Const] : 1
_dx17[Const] : 1 ; _dx18[Const] : 1 ; _dx19[Const] : 1

/* Algorithm */

ad2case : {((dx13 + dx16) - (dx11 + dx14))/4,\
           ((dx13 + dx16) - (dx12 + dx15))/2,\
           ((dx12 + dx15) - (dx11 + dx14))/2,\
           0}
ad2cond : {"MGD.EQ.0",\
          "MGD.EQ.1",\
          "MGD.EQ.2",\
          "MGD.EQ.3"}
casederiv[4,"D(delta theta)/Dxi2",ad2,ad2case,ad2cond,"mgds2.sgf"]

ad3case : {((dx14 + 2dx15 + dx16) - (dx11 + 2dx12 + dx13))/4,\
           dx15 - dx12}
ad3cond : {"MGD.EQ.0",\
           ".TRUE."}
casederiv[2,"D(delta theta)/Dxi3",ad3,ad3case,ad3cond,"mgds2.sgf"]

/* Include in exprelist the dependence of ad2 and ad3 on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad2,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[ad3,dii]; \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C


C
C     delta theta at 1
      COEF = DFXD12*AD2D1 + DFXD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 2
      COEF = DFXD12*AD2D2 + DFXD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 3
      COEF = DFXD12*AD2D3 + DFXD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 4
      COEF = DFXD12*AD2D4 + DFXD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 5
      COEF = DFXD12*AD2D5 + DFXD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 6
      COEF = DFXD12*AD2D6 + DFXD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 7
      COEF = DFXD12*AD2D7 + DFXD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 8
      COEF = DFXD12*AD2D8 + DFXD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 9
      COEF = DFXD12*AD2D9 + DFXD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
```

```
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     delta theta at 1
      COEF = DFYD12*AD2D1 + DFYD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 2
      COEF = DFYD12*AD2D2 + DFYD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 3
      COEF = DFYD12*AD2D3 + DFYD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 4
      COEF = DFYD12*AD2D4 + DFYD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 5
      COEF = DFYD12*AD2D5 + DFYD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 6
      COEF = DFYD12*AD2D6 + DFYD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 7
      COEF = DFYD12*AD2D7 + DFYD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 8
      COEF = DFYD12*AD2D8 + DFYD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 9
      COEF = DFYD12*AD2D9 + DFYD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C

C
C     delta theta at 1
      COEF = DFZD12*AD2D1 + DFZD13*AD3D1
      CALL ADCOFS(COEF,IDX11+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 2
      COEF = DFZD12*AD2D2 + DFZD13*AD3D2
      CALL ADCOFS(COEF,IDX12+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 3
      COEF = DFZD12*AD2D3 + DFZD13*AD3D3
      CALL ADCOFS(COEF,IDX13+IDTHO,COFZ,ICOLZ,NCZ)
```

```
C
C     delta theta at 4
      COEF = DFZD12*AD2D4 + DFZD13*AD3D4
      CALL ADCOFS(COEF,IDX14+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 5
      COEF = DFZD12*AD2D5 + DFZD13*AD3D5
      CALL ADCOFS(COEF,IDX15+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 6
      COEF = DFZD12*AD2D6 + DFZD13*AD3D6
      CALL ADCOFS(COEF,IDX16+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 7
      COEF = DFZD12*AD2D7 + DFZD13*AD3D7
      CALL ADCOFS(COEF,IDX17+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 8
      COEF = DFZD12*AD2D8 + DFZD13*AD3D8
      CALL ADCOFS(COEF,IDX18+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 9
      COEF = DFZD12*AD2D9 + DFZD13*AD3D9
      CALL ADCOFS(COEF,IDX19+IDTHO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmgds2dec.sgf;* Lmgds2dec.osf
!dele Lmgds2dec.sgf;*
!@cgf Lmgds2dec.sgf

<"decfile.in"

decfile["Lmgds2dec.sgf"]
```

## File VELP.IN

```
/* This session creates the subroutine which calculates the velocity from
the physical derivatives of psi1 and psi2 for planer coordinates.
The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy velp.sgf;* velp.osf
!dele velp.sgf;*
!@cgf velp.sgf

filename : "velp.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RVELP(IND,RWXV,RWYV,RWZV)
C     Right VELocity calculation for planer coordinates.
C
C     This routine calculates the 3 components of velocity from the
C     physical derivatives of psi1 and psi2.
C
C     IND is the face indicator.
C          IND         face
C           1           A
C           2           B
C           3           S
C
      INTEGER IND
      REAL RWXV,RWYV,RWZV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RVELPDEC.FOR'

C     Declare the derivatives of psi1 and psi2.
      REAL SI11,SI12,SI13,SI21,SI22,SI23

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the derivatives of psi1 and psi2.
      IF(IND.EQ.1)THEN
        CALL RPDSI(1,SI11,SI12,SI13)
        CALL RPDSI(4,SI21,SI22,SI23)
      ELSE IF(IND.EQ.2)THEN
        CALL RPDSI(2,SI11,SI12,SI13)
        CALL RPDSI(5,SI21,SI22,SI23)
      ELSE IF(IND.EQ.3)THEN
        CALL RPDSI(3,SI11,SI12,SI13)
        CALL RPDSI(6,SI21,SI22,SI23)
      END IF

",filename]

rwx : si12 si23 - si13 si22
fort["Rho Wx.",'rwx,"velp.sgf"]

rwy : si13 si21 - si11 si23
fort["Rho Wy.",'rwy,"velp.sgf"]

rwz : si11 si22 - si12 si21
fort["Rho Wz.",'rwz,"velp.sgf"]

Lpr["
      RWXV = RWX
      RWYV = RWY
      RWZV = RWZ

      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */
/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rvelpdec.sgf;* Rvelpdec.osf
!dele Rvelpdec.sgf;*
!@cgf Rvelpdec.sgf

<"decfile.in"

decfile["Rvelpdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "velp.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LVELP(IND,DFXDWX,DFXDWY,DFXDWZ,
     1                       DFYDWX,DFYDWY,DFYDWZ,
     1                       DFZDWX,DFZDWY,DFZDWZ)
C     Left VELocity calculation for planer coordinates.
C
C     This routine calculates the contribution of the 3 components of
C     velocity from the physical derivatives of psi1 and psi2 to the left hand
C     sides of the x, y and z momentum eqs.
C
C     IND is the face indicator.
C          IND          face
C           1            A
C           2            B
C           3            S
C
      INTEGER IND
      REAL DFXDWX,DFXDWY,DFXDWZ
      REAL DFYDWX,DFYDWY,DFYDWZ
      REAL DFZDWX,DFZDWY,DFZDWZ

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LVELPDEC.FOR'
      REAL DERI

C     Declare the derivatives of psi1 and psi2.
      REAL SI11,SI12,SI13,SI21,SI22,SI23

C     The changes in the physical derivatives.
      REAL DFXD11,DFXD12,DFXD13
      REAL DFYD11,DFYD12,DFYD13
      REAL DFZD11,DFZD12,DFZD13

      REAL DFXD21,DFXD22,DFXD23
      REAL DFYD21,DFYD22,DFYD23
      REAL DFZD21,DFZD22,DFZD23

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {si11,si12,si13,si21,si22,si23}
niv : Len[invar]
```

```
/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_si11[Const] : 1 ; _si12[Const] : 1 ; _si13[Const] : 1
_si21[Const] : 1 ; _si22[Const] : 1 ; _si23[Const] : 1

/* Algorithm */

Lpr["

C     Get the derivatives of psi1 and psi2.
      IF(IND.EQ.1)THEN
        CALL RPDSI(1,SI11,SI12,SI13)
        CALL RPDSI(4,SI21,SI22,SI23)
      ELSE IF(IND.EQ.2)THEN
        CALL RPDSI(2,SI11,SI12,SI13)
        CALL RPDSI(5,SI21,SI22,SI23)
      ELSE IF(IND.EQ.3)THEN
        CALL RPDSI(3,SI11,SI12,SI13)
        CALL RPDSI(6,SI21,SI22,SI23)
      END IF

",filename]

rwx : si12 si23 - si13 si22
deriv["Rho Wx.",'rwx,"velp.sgf"]

rwy : si13 si21 - si11 si23
deriv["Rho Wy.",'rwy,"velp.sgf"]

rwz : si11 si22 - si12 si21
deriv["Rho Wz.",'rwz,"velp.sgf"]

/* Include in exprelist the dependence of rwx, rwy, and rwz on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[rwx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[rwy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[rwz,dii] \
]

Lpr["
C     The dependence on the psi derivatives.
      DFXD11 = DFXDWX*RWXD1 + DFXDWY*RWYD1 + DFXDWZ*RWZD1
      DFXD12 = DFXDWX*RWXD2 + DFXDWY*RWYD2 + DFXDWZ*RWZD2
      DFXD13 = DFXDWX*RWXD3 + DFXDWY*RWYD3 + DFXDWZ*RWZD3

      DFXD21 = DFXDWX*RWXD4 + DFXDWY*RWYD4 + DFXDWZ*RWZD4
      DFXD22 = DFXDWX*RWXD5 + DFXDWY*RWYD5 + DFXDWZ*RWZD5
      DFXD23 = DFXDWX*RWXD6 + DFXDWY*RWYD6 + DFXDWZ*RWZD6

      DFYD11 = DFYDWX*RWXD1 + DFYDWY*RWYD1 + DFYDWZ*RWZD1
      DFYD12 = DFYDWX*RWXD2 + DFYDWY*RWYD2 + DFYDWZ*RWZD2
      DFYD13 = DFYDWX*RWXD3 + DFYDWY*RWYD3 + DFYDWZ*RWZD3

      DFYD21 = DFYDWX*RWXD4 + DFYDWY*RWYD4 + DFYDWZ*RWZD4
      DFYD22 = DFYDWX*RWXD5 + DFYDWY*RWYD5 + DFYDWZ*RWZD5
      DFYD23 = DFYDWX*RWXD6 + DFYDWY*RWYD6 + DFYDWZ*RWZD6

      DFZD11 = DFZDWX*RWXD1 + DFZDWY*RWYD1 + DFZDWZ*RWZD1
      DFZD12 = DFZDWX*RWXD2 + DFZDWY*RWYD2 + DFZDWZ*RWZD2
      DFZD13 = DFZDWX*RWXD3 + DFZDWY*RWYD3 + DFZDWZ*RWZD3
      DFZD21 = DFZDWX*RWXD4 + DFZDWY*RWYD4 + DFZDWZ*RWZD4
      DFZD22 = DFZDWX*RWXD5 + DFZDWY*RWYD5 + DFZDWZ*RWZD5
      DFZD23 = DFZDWX*RWXD6 + DFZDWY*RWYD6 + DFZDWZ*RWZD6

      IF(IND.EQ.1)THEN
        CALL LPDSI(1,DFXD11,DFXD12,DFXD13,DFYD11,DFYD12,DFYD13,
     1             DFZD11,DFZD12,DFZD13)
        CALL LPDSI(4,DFXD21,DFXD22,DFXD23,DFYD21,DFYD22,DFYD23,
     1             DFZD21,DFZD22,DFZD23)
      ELSE IF(IND.EQ.2)THEN
        CALL LPDSI(2,DFXD11,DFXD12,DFXD13,DFYD11,DFYD12,DFYD13,
     1             DFZD11,DFZD12,DFZD13)
        CALL LPDSI(5,DFXD21,DFXD22,DFXD23,DFYD21,DFYD22,DFYD23,
     1             DFZD21,DFZD22,DFZD23)
      ELSE IF(IND.EQ.3)THEN
        CALL LPDSI(3,DFXD11,DFXD12,DFXD13,DFYD11,DFYD12,DFYD13,
     1             DFZD11,DFZD12,DFZD13)
        CALL LPDSI(6,DFXD21,DFXD22,DFXD23,DFYD21,DFYD22,DFYD23,
     1             DFZD21,DFZD22,DFZD23)
      END IF

",filename]

Lpr["

      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lvelpdec.sgf;* Lvelpdec.osf
!dele Lvelpdec.sgf;*
!@cgf Lvelpdec.sgf

<"decfile.in"

decfile["Lvelpdec.sgf"]
```

## File VELC.IN

```
/* This session creates the subroutine which calculates the velocity from
the physical derivatives of psi1 and psi2 for cylindrical coordinates.
The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy velc.sgf;* velc.osf
!dele velc.sgf;*
!@cgf velc.sgf

filename : "velc.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RVELC(IND,RWXV,RWYV,RWZV)
C     Right VELocity calculation for Cylindrical coordinates.
C
```

```
C     This routine calculates the 3 components of velocity from the
C     physical derivatives of psi1 and psi2.
C
C     IND is the face indicator.
C         IND        face
C          1          A
C          2          B
C          3          S
C
      INTEGER IND
      REAL RWXV,RWYV,RWZV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RVELCDEC.FOR'

C     Declare the derivatives of psi1 and psi2.
      REAL SI11,SI12,SI13,SI21,SI22,SI23

C     Declare the r and theta at the face center.
      REAL R,X1

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the derivatives of psi1 and psi2.
      IF(IND.EQ.1)THEN
        CALL RPDSI(1,SI11,SI12,SI13)
        CALL RPDSI(4,SI21,SI22,SI23)
      ELSE IF(IND.EQ.2)THEN
        CALL RPDSI(2,SI11,SI12,SI13)
        CALL RPDSI(5,SI21,SI22,SI23)
      ELSE IF(IND.EQ.3)THEN
        CALL RPDSI(3,SI11,SI12,SI13)
        CALL RPDSI(6,SI21,SI22,SI23)
      END IF

C     Get the r and theta at the face center.
      CALL RRTHFC(R,X1)

",filename]

rw1 : si12 si23 - si13 si22
fort["Rho W(theta).",'rw1,"velc.sgf"]

rwr : (si13 si21 - si11 si23)/r
fort["Rho Wr.",'rwr,"velc.sgf"]

rwz : (si11 si22 - si12 si21)/r
fort["Rho Wz.",'rwz,"velc.sgf"]

rwx : rwr Sin[x1] + rw1 Cos[x1]
fort["Rho Wx.",'rwx,"velc.sgf"]

rwy : rwr Cos[x1] - rw1 Sin[x1]
fort["Rho Wy.",'rwy,"velc.sgf"]

Lpr["
      RWXV = RWX
      RWYV = RWY
      RWZV = RWZ

      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rvelcdec.sgf;* Rvelcdec.osf
!dele Rvelcdec.sgf;*
!@cgf Rvelcdec.sgf

<"decfile.in"

decfile["Rvelcdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "velc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LVELC(IND,DFXDWX,DFXDWY,DFXDWZ,
     1                     DFYDWX,DFYDWY,DFYDWZ,
     1                     DFZDWX,DFZDWY,DFZDWZ)
C     Left VELocity calculation for Cylindrical coordinates.
C
C     This routine calculates the contribution of the 3 components of
C     velocity from the physical derivatives of psi1 and psi2 to the left hand
C     sides of the x, y and z momentum eqs.
C
C     IND is the face indicator.
C          IND        face
C           1          A
C           2          B
C           3          S
C
      INTEGER IND
      REAL DFXDWX,DFXDWY,DFXDWZ
      REAL DFYDWX,DFYDWY,DFYDWZ
      REAL DFZDWX,DFZDWY,DFZDWZ

      INCLUDE 'FACE.FOR'
 C    These declarations are generated by SMP:
      INCLUDE 'LVELCDEC.FOR'
      REAL DERI

 C    Declare the derivatives of psi1 and psi2.
      REAL SI11,SI12,SI13,SI21,SI22,SI23

 C    Declare the r and theta at the face center.
      REAL R,X1

 C    The changes in the physical derivatives.
      REAL DFXD11,DFXD12,DFXD13
      REAL DFYD11,DFYD12,DFYD13
      REAL DFZD11,DFZD12,DFZD13
      REAL DFXD21,DFXD22,DFXD23
      REAL DFYD21,DFYD22,DFYD23
      REAL DFZD21,DFZD22,DFZD23

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {si11,si12,si13,si21,si22,si23,r,x1}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_si11[Const] : 1 ; _si12[Const] : 1 ; _si13[Const] : 1
_si21[Const] : 1 ; _si22[Const] : 1 ; _si23[Const] : 1
_r[Const] : 1 ; _x1[Const] : 1

/* Algorithm */

Lpr["

C     Get the derivatives of psi1 and psi2.
      IF(IND.EQ.1)THEN
        CALL RPDSI(1,SI11,SI12,SI13)
        CALL RPDSI(4,SI21,SI22,SI23)
      ELSE IF(IND.EQ.2)THEN
        CALL RPDSI(2,SI11,SI12,SI13)
        CALL RPDSI(5,SI21,SI22,SI23)
      ELSE IF(IND.EQ.3)THEN
        CALL RPDSI(3,SI11,SI12,SI13)
        CALL RPDSI(6,SI21,SI22,SI23)
      END IF

C     Get the r and theta at the face center.
      CALL RRTHFC(R,X1)

",filename]

rw1 : si12 si23 - si13 si22
deriv["Rho W(theta).",'rw1,"velc.sgf"]

rwr : (si13 si21 - si11 si23)/r
deriv["Rho Wr.",'rwr,"velc.sgf"]

rwz : (si11 si22 - si12 si21)/r
deriv["Rho Wz.",'rwz,"velc.sgf"]

rwx : rwr Sin[x1] + rw1 Cos[x1]
deriv["Rho Wx.",'rwx,"velc.sgf"]

rwy : rwr Cos[x1] - rw1 Sin[x1]
deriv["Rho Wy.",'rwy,"velc.sgf"]

/* Include in exprelist the dependence of rwx, rwy, and rwz on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[rwx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[rwy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[rwz,dii] \
]

Lpr["
C     The dependence on the psi derivatives.
      DFXD11 = DFXDWX*RWXD1 + DFXDWY*RWYD1 + DFXDWZ*RWZD1
      DFXD12 = DFXDWX*RWXD2 + DFXDWY*RWYD2 + DFXDWZ*RWZD2
      DFXD13 = DFXDWX*RWXD3 + DFXDWY*RWYD3 + DFXDWZ*RWZD3

      DFXD21 = DFXDWX*RWXD4 + DFXDWY*RWYD4 + DFXDWZ*RWZD4
      DFXD22 = DFXDWX*RWXD5 + DFXDWY*RWYD5 + DFXDWZ*RWZD5
      DFXD23 = DFXDWX*RWXD6 + DFXDWY*RWYD6 + DFXDWZ*RWZD6

      DFYD11 = DFYDWX*RWXD1 + DFYDWY*RWYD1 + DFYDWZ*RWZD1
      DFYD12 = DFYDWX*RWXD2 + DFYDWY*RWYD2 + DFYDWZ*RWZD2
      DFYD13 = DFYDWX*RWXD3 + DFYDWY*RWYD3 + DFYDWZ*RWZD3

      DFYD21 = DFYDWX*RWXD4 + DFYDWY*RWYD4 + DFYDWZ*RWZD4
      DFYD22 = DFYDWX*RWXD5 + DFYDWY*RWYD5 + DFYDWZ*RWZD5
      DFYD23 = DFYDWX*RWXD6 + DFYDWY*RWYD6 + DFYDWZ*RWZD6

      DFZD11 = DFZDWX*RWXD1 + DFZDWY*RWYD1 + DFZDWZ*RWZD1
      DFZD12 = DFZDWX*RWXD2 + DFZDWY*RWYD2 + DFZDWZ*RWZD2
      DFZD13 = DFZDWX*RWXD3 + DFZDWY*RWYD3 + DFZDWZ*RWZD3

      DFZD21 = DFZDWX*RWXD4 + DFZDWY*RWYD4 + DFZDWZ*RWZD4
      DFZD22 = DFZDWX*RWXD5 + DFZDWY*RWYD5 + DFZDWZ*RWZD5
      DFZD23 = DFZDWX*RWXD6 + DFZDWY*RWYD6 + DFZDWZ*RWZD6

      IF(IND.EQ.1)THEN
        CALL LPDSI(1,DFXD11,DFXD12,DFXD13,DFYD11,DFYD12,DFYD13,
     1             DFZD11,DFZD12,DFZD13)
        CALL LPDSI(4,DFXD21,DFXD22,DFXD23,DFYD21,DFYD22,DFYD23,
     1             DFZD21,DFZD22,DFZD23)
      ELSE IF(IND.EQ.2)THEN
        CALL LPDSI(2,DFXD11,DFXD12,DFXD13,DFYD11,DFYD12,DFYD13,
     1             DFZD11,DFZD12,DFZD13)
        CALL LPDSI(5,DFXD21,DFXD22,DFXD23,DFYD21,DFYD22,DFYD23,
     1             DFZD21,DFZD22,DFZD23)
      ELSE IF(IND.EQ.3)THEN
        CALL LPDSI(3,DFXD11,DFXD12,DFXD13,DFYD11,DFYD12,DFYD13,
     1             DFZD11,DFZD12,DFZD13)
        CALL LPDSI(6,DFXD21,DFXD22,DFXD23,DFYD21,DFYD22,DFYD23,
     1             DFZD21,DFZD22,DFZD23)
      END IF

",filename]

Lpr["
C     The dependence on r and theta at the face center.
      CALL LRTHFC(DFXDWX*RWXD7+DFXDWY*RWYD7+DFXDWZ*RWZD7,
     1            DFXDWX*RWXD8+DFXDWY*RWYD8+DFXDWZ*RWZD8,
     1            DFYDWX*RWXD7+DFYDWY*RWYD7+DFYDWZ*RWZD7,
     1            DFYDWX*RWXD8+DFYDWY*RWYD8+DFYDWZ*RWZD8,
     1            DFZDWX*RWXD7+DFZDWY*RWYD7+DFZDWZ*RWZD7,
     1            DFZDWX*RWXD8+DFZDWY*RWYD8+DFZDWZ*RWZD8)

",filename]

Lpr["

      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */
/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lvelcdec.sgf;* Lvelcdec.osf
!dele Lvelcdec.sgf;*
!@cgf Lvelcdec.sgf

<"decfile.in"

decfile["Lvelcdec.sgf"]
```

## File VOLP.IN

```
/* This session creates the subroutine which calculates the Volume
for planer grids.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy volp.sgf;* volp.osf
!dele volp.sgf;*
!@cgf volp.sgf

filename : "volp.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RVOLP(VOLV)
C     Right Volume calculation Planer.
C
C     This routine calculates the volume for a planer grid.
C     The variables in common are already initialized for the volume.
C
      REAL VOLV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RVOLPDEC.FOR'

C     Declare and set the moving grid variables to 0.
      REAL DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L
      REAL DSLEF,DSLEE
      DATA DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L/8*0./
      DATA DSLEF,DSLEE/2*0./

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

x1e : x1ec + dx1e + d1dlee dslee
fort["X1 at e.",'x1e,"volp.sgf"]

x1f : x1fc + dx1f + d1dlef dslef
fort["X1 at f.",'x1f,"volp.sgf"]

x1g : x1gc + dx1g + d1dleg dslef
fort["X1 at g.",'x1g,"volp.sgf"]

x1h : x1hc + dx1h + d1dleh dslee
```

```
fort["X1 at h.",'x1h,"volp.sgf"]

x1i : x1ic + dx1i + d1dlei dslee
fort["X1 at i.",'x1i,"volp.sgf"]

x1j : x1jc + dx1j + d1dlej dslef
fort["X1 at j.",'x1j,"volp.sgf"]

x1k : x1kc + dx1k + d1dlek dslef
fort["X1 at k.",'x1k,"volp.sgf"]

x1l : x1lc + dx1l + d1dlel dslee
fort["X1 at l.",'x1l,"volp.sgf"]

x2e : x2ec + d2dlee dslee
fort["X2 at e.",'x2e,"volp.sgf"]

x2f : x2fc + d2dlef dslef
fort["X2 at f.",'x2f,"volp.sgf"]

x2g : x2gc + d2dleg dslef
fort["X2 at g.",'x2g,"volp.sgf"]

x2h : x2hc + d2dleh dslee
fort["X2 at h.",'x2h,"volp.sgf"]

x2i : x2ic + d2dlei dslee
fort["X2 at i.",'x2i,"volp.sgf"]

x2j : x2jc + d2dlej dslef
fort["X2 at j.",'x2j,"volp.sgf"]

x2k : x2kc + d2dlek dslef
fort["X2 at k.",'x2k,"volp.sgf"]

x2l : x2lc + d2dlel dslee
fort["X2 at l.",'x2l,"volp.sgf"]

ze : x3ec + d3dlee dslee
fort["Z or x3 at e.",'ze,"volp.sgf"]

zf : x3fc + d3dlef dslef
fort["Z or x3 at f.",'zf,"volp.sgf"]

zg : x3gc + d3dleg dslef
fort["Z or x3 at g.",'zg,"volp.sgf"]

zh : x3hc + d3dleh dslee
fort["Z or x3 at h.",'zh,"volp.sgf"]

zi : x3ic + d3dlei dslee
fort["Z or x3 at i.",'zi,"volp.sgf"]

zj : x3jc + d3dlej dslef
fort["Z or x3 at j.",'zj,"volp.sgf"]

zk : x3kc + d3dlek dslef
fort["Z or x3 at k.",'zk,"volp.sgf"]

zl : x3lc + d3dlel dslee
fort["Z or x3 at l.",'zl,"volp.sgf"]

xe: x1e
fort["X at e calculated.",'xe,"volp.sgf"]

xf: x1f
```

```
fort["X at f calculated.",'xf,"volp.sgf"]

xg: x1g
fort["X at g calculated.",'xg,"volp.sgf"]

xh: x1h
fort["X at h calculated.",'xh,"volp.sgf"]

xi: x1i
fort["X at i calculated.",'xi,"volp.sgf"]

xj: x1j
fort["X at j calculated.",'xj,"volp.sgf"]

xk: x1k
fort["X at k calculated.",'xk,"volp.sgf"]

xl: x1l
fort["X at l calculated.",'xl,"volp.sgf"]

ye: x2e
fort["Y at e calculated.",'ye,"volp.sgf"]

yf: x2f
fort["Y at f calculated.",'yf,"volp.sgf"]

yg: x2g
fort["Y at g calculated.",'yg,"volp.sgf"]

yh: x2h
fort["Y at h calculated.",'yh,"volp.sgf"]

yi: x2i
fort["Y at i calculated.",'yi,"volp.sgf"]

yj: x2j
fort["Y at j calculated.",'yj,"volp.sgf"]

yk: x2k
fort["Y at k calculated.",'yk,"volp.sgf"]

yl: x2l
fort["Y at l calculated.",'yl,"volp.sgf"]

dx1 : ((xg + xh + xk + xl) - (xe + xf + xi + xj))/4
fort["Delta x for vector 1",'dx1,"volp.sgf"]

dy1 : ((yg + yh + yk + yl) - (ye + yf + yi + yj))/4
fort["Delta y for vector 1",'dy1,"volp.sgf"]

dz1 : ((zg + zh + zk + zl) - (ze + zf + zi + zj))/4
fort["Delta z for vector 1",'dz1,"volp.sgf"]

dx2 : ((xf + xg + xj + xk) - (xe + xh + xi + xl))/4
fort["Delta x for vector 2",'dx2,"volp.sgf"]

dy2 : ((yf + yg + yj + yk) - (ye + yh + yi + yl))/4
fort["Delta y for vector 2",'dy2,"volp.sgf"]

dz2 : ((zf + zg + zj + zk) - (ze + zh + zi + zl))/4
fort["Delta z for vector 2",'dz2,"volp.sgf"]

dx3 : ((xi + xj + xk + xl) - (xe + xf + xg + xh))/4
fort["Delta x for vector 3",'dx3,"volp.sgf"]

dy3 : ((yi + yj + yk + yl) - (ye + yf + yg + yh))/4
```

```
fort["Delta y for vector 3",'dy3,"volp.sgf"]

dz3 : ((zi + zj + zk + zl) - (ze + zf + zg + zh))/4
fort["Delta z for vector 3",'dz3,"volp.sgf"]

vol : (dy1 dz2 - dy2 dz1)dx3 + (dx2 dz1 - dx1 dz2)dy3 +\
      (dx1 dy2 - dx2 dy1)dz3
fort["Volume",'vol,"volp.sgf"]

Lpr["
      VOLV = VOL
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rvolpdec.sgf;* Rvolpdec.osf
!dele Rvolpdec.sgf;*
!@cgf Rvolpdec.sgf

<"decfile.in"

decfile["Rvolpdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lvolpdec.sgf;* Lvolpdec.osf
!dele Lvolpdec.sgf;*
!@cgf Lvolpdec.sgf

filename : "volp.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LVOLP(DFXDV,DFYDV,DFZDV)
C     Left Volume calculation Planer.
C
C     This routine calculates the contribution of the volume for a planer grid
C     to the left hand sides of the x, y, and z momentum eqs.
C     The variables in common are already initialized for the volume.
C
      REAL DFXDV,DFYDV,DFZDV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LVOLPDEC.FOR'

C     Declare and set the moving grid variables to 0.
```

```
      REAL DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L
      REAL DSLEF,DSLEE
      DATA DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L/8*0./
      DATA DSLEF,DSLEE/2*0./

      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx1e,dx1f,dx1g,dx1h,dx1i,dx1j,dx1k,dx1l,dslef,dslee}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx1e[Const] : 1 ; _dx1f[Const] : 1 ; _dx1g[Const] : 1 ; _dx1h[Const] : 1
_dx1i[Const] : 1 ; _dx1j[Const] : 1 ; _dx1k[Const] : 1 ; _dx1l[Const] : 1
_dslef[Const] : 1 ; _dslee[Const] : 1
_x1ec[Const] : 1 ; _x1fc[Const] : 1 ; _x1gc[Const] : 1 ; _x1hc[Const] : 1
_x2ec[Const] : 1 ; _x2fc[Const] : 1 ; _x2gc[Const] : 1 ; _x2hc[Const] : 1
_x3ec[Const] : 1 ; _x3fc[Const] : 1 ; _x3gc[Const] : 1 ; _x3hc[Const] : 1
_x1ic[Const] : 1 ; _x1jc[Const] : 1 ; _x1kc[Const] : 1 ; _x1lc[Const] : 1
_x2ic[Const] : 1 ; _x2jc[Const] : 1 ; _x2kc[Const] : 1 ; _x2lc[Const] : 1
_x3ic[Const] : 1 ; _x3jc[Const] : 1 ; _x3kc[Const] : 1 ; _x3lc[Const] : 1
_d1dlee[Const] : 1 ; _d1dlef[Const] : 1
_d1dleg[Const] : 1 ; _d1dleh[Const] : 1
_d1dlei[Const] : 1 ; _d1dlej[Const] : 1
_d1dlek[Const] : 1 ; _d1dlel[Const] : 1
_d2dlee[Const] : 1 ; _d2dlef[Const] : 1
_d2dleg[Const] : 1 ; _d2dleh[Const] : 1
_d2dlei[Const] : 1 ; _d2dlej[Const] : 1
_d2dlek[Const] : 1 ; _d2dlel[Const] : 1
_d3dlee[Const] : 1 ; _d3dlef[Const] : 1
_d3dleg[Const] : 1 ; _d3dleh[Const] : 1
_d3dlei[Const] : 1 ; _d3dlej[Const] : 1
_d3dlek[Const] : 1 ; _d3dlel[Const] : 1

/* Algorithm */

x1e : x1ec + dx1e + d1dlee dslee
deriv["X1 at e.",'x1e,"volp.sgf"]

x1f : x1fc + dx1f + d1dlef dslef
deriv["X1 at f.",'x1f,"volp.sgf"]

x1g : x1gc + dx1g + d1dleg dslef
deriv["X1 at g.",'x1g,"volp.sgf"]

x1h : x1hc + dx1h + d1dleh dslee
deriv["X1 at h.",'x1h,"volp.sgf"]

x1i : x1ic + dx1i + d1dlei dslee
deriv["X1 at i.",'x1i,"volp.sgf"]

x1j : x1jc + dx1j + d1dlej dslef
deriv["X1 at j.",'x1j,"volp.sgf"]

x1k : x1kc + dx1k + d1dlek dslef
deriv["X1 at k.",'x1k,"volp.sgf"]

x1l : x1lc + dx1l + d1dlel dslee
deriv["X1 at l.",'x1l,"volp.sgf"]
```

```
x2e : x2ec + d2dlee dslee
deriv["X2 at e.",'x2e,"volp.sgf"]

x2f : x2fc + d2dlef dslef
deriv["X2 at f.",'x2f,"volp.sgf"]

x2g : x2gc + d2dleg dslef
deriv["X2 at g.",'x2g,"volp.sgf"]

x2h : x2hc + d2dleh dslee
deriv["X2 at h.",'x2h,"volp.sgf"]

x2i : x2ic + d2dlei dslee
deriv["X2 at i.",'x2i,"volp.sgf"]

x2j : x2jc + d2dlej dslef
deriv["X2 at j.",'x2j,"volp.sgf"]

x2k : x2kc + d2dlek dslef
deriv["X2 at k.",'x2k,"volp.sgf"]

x2l : x2lc + d2dlel dslee
deriv["X2 at l.",'x2l,"volp.sgf"]

ze : x3ec + d3dlee dslee
deriv["Z or x3 at e.",'ze,"volp.sgf"]

zf : x3fc + d3dlef dslef
deriv["Z or x3 at f.",'zf,"volp.sgf"]

zg : x3gc + d3dleg dslef
deriv["Z or x3 at g.",'zg,"volp.sgf"]

zh : x3hc + d3dleh dslee
deriv["Z or x3 at h.",'zh,"volp.sgf"]

zi : x3ic + d3dlei dslee
deriv["Z or x3 at i.",'zi,"volp.sgf"]

zj : x3jc + d3dlej dslef
deriv["Z or x3 at j.",'zj,"volp.sgf"]

zk : x3kc + d3dlek dslef
deriv["Z or x3 at k.",'zk,"volp.sgf"]

zl : x3lc + d3dlel dslee
deriv["Z or x3 at l.",'zl,"volp.sgf"]

xe: x1e
deriv["X at e calculated.",'xe,"volp.sgf"]

xf: x1f
deriv["X at f calculated.",'xf,"volp.sgf"]

xg: x1g
deriv["X at g calculated.",'xg,"volp.sgf"]

xh: x1h
deriv["X at h calculated.",'xh,"volp.sgf"]

xi: x1i
deriv["X at i calculated.",'xi,"volp.sgf"]

xj: x1j
deriv["X at j calculated.",'xj,"volp.sgf"]
```

```
xk: x1k
deriv["X at k calculated.",'xk,"volp.sgf"]

xl: x1l
deriv["X at l calculated.",'xl,"volp.sgf"]

ye: x2e
deriv["Y at e calculated.",'ye,"volp.sgf"]

yf: x2f
deriv["Y at f calculated.",'yf,"volp.sgf"]

yg: x2g
deriv["Y at g calculated.",'yg,"volp.sgf"]

yh: x2h
deriv["Y at h calculated.",'yh,"volp.sgf"]

yi: x2i
deriv["Y at i calculated.",'yi,"volp.sgf"]

yj: x2j
deriv["Y at j calculated.",'yj,"volp.sgf"]

yk: x2k
deriv["Y at k calculated.",'yk,"volp.sgf"]

yl: x2l
deriv["Y at l calculated.",'yl,"volp.sgf"]

dx1 : ((xg + xh + xk + xl) - (xe + xf + xi + xj))/4
deriv["Delta x for vector 1",'dx1,"volp.sgf"]

dy1 : ((yg + yh + yk + yl) - (ye + yf + yi + yj))/4
deriv["Delta y for vector 1",'dy1,"volp.sgf"]

dz1 : ((zg + zh + zk + zl) - (ze + zf + zi + zj))/4
deriv["Delta z for vector 1",'dz1,"volp.sgf"]

dx2 : ((xf + xg + xj + xk) - (xe + xh + xi + xl))/4
deriv["Delta x for vector 2",'dx2,"volp.sgf"]

dy2 : ((yf + yg + yj + yk) - (ye + yh + yi + yl))/4
deriv["Delta y for vector 2",'dy2,"volp.sgf"]

dz2 : ((zf + zg + zj + zk) - (ze + zh + zi + zl))/4
deriv["Delta z for vector 2",'dz2,"volp.sgf"]

dx3 : ((xi + xj + xk + xl) - (xe + xf + xg + xh))/4
deriv["Delta x for vector 3",'dx3,"volp.sgf"]

dy3 : ((yi + yj + yk + yl) - (ye + yf + yg + yh))/4
deriv["Delta y for vector 3",'dy3,"volp.sgf"]

dz3 : ((zi + zj + zk + zl) - (ze + zf + zg + zh))/4
deriv["Delta z for vector 3",'dz3,"volp.sgf"]

vol : (dy1 dz2 - dy2 dz1)dx3 + (dx2 dz1 - dx1 dz2)dy3 +\
      (dx1 dy2 - dx2 dy1)dz3
deriv["Volume",'vol,"volp.sgf"]

/* Include in exprelist the dependence of vol on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[vol,dii]; \
]

Lpr["
C     The dependence on delta x1 at volume nodes.
      CALL LDX1VN(LE,DFXDV*VOLD1,DFYDV*VOLD1,DFZDV*VOLD1)

      CALL LDX1VN(LF,DFXDV*VOLD2,DFYDV*VOLD2,DFZDV*VOLD2)

      CALL LDX1VN(LG,DFXDV*VOLD3,DFYDV*VOLD3,DFZDV*VOLD3)

      CALL LDX1VN(LH,DFXDV*VOLD4,DFYDV*VOLD4,DFZDV*VOLD4)

      CALL LDX1VN(LI,DFXDV*VOLD5,DFYDV*VOLD5,DFZDV*VOLD5)

      CALL LDX1VN(LJ,DFXDV*VOLD6,DFYDV*VOLD6,DFZDV*VOLD6)

      CALL LDX1VN(LK,DFXDV*VOLD7,DFYDV*VOLD7,DFZDV*VOLD7)

      CALL LDX1VN(LL,DFXDV*VOLD8,DFYDV*VOLD8,DFZDV*VOLD8)

C     The dependence on delta sble upper.
      CALL LSBLEN(JSLEU,NJJ,DFXDV*VOLD9,DFYDV*VOLD9,DFZDV*VOLD9)

C     The dependence on delta sble lower.
      CALL LSBLEN(JSLEL,NJJ,DFXDV*VOLD10,DFYDV*VOLD10,DFZDV*VOLD10)

",filename]

Lpr["

      RETURN
      END
",filename]

<"decfile.in"

decfile["Lvolpdec.sgf"]
```

## File VOLC.IN

```
/* This session creates the subroutine which calculates the Volume
for cylindrical grids.  The left hand side subroutine is also
created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy volc.sgf;* volc.osf
!dele volc.sgf;*
!@cgf volc.sgf

filename : "volc.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RVOLC(VOLV)
C     Right Volume calculation Cylindrical.
C
C     This routine calculates the volume for a cylindrical grid.
C     The variables in common are already initialized for the volume.
C
      REAL VOLV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RVOLCDEC.FOR'

C     Declare and set the moving grid variables to 0.
      REAL DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L
      REAL DSLEF,DSLEE
      DATA DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L/8*0./
      DATA DSLEF,DSLEE/2*0./

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

x1e : x1ec + dx1e + d1dlee dslee
fort["X1 at e.",'x1e,"volc.sgf"]

x1f : x1fc + dx1f + d1dlef dslef
fort["X1 at f.",'x1f,"volc.sgf"]

x1g : x1gc + dx1g + d1dleg dslef
fort["X1 at g.",'x1g,"volc.sgf"]

x1h : x1hc + dx1h + d1dleh dslee
fort["X1 at h.",'x1h,"volc.sgf"]

x1i : x1ic + dx1i + d1dlei dslee
fort["X1 at i.",'x1i,"volc.sgf"]

x1j : x1jc + dx1j + d1dlej dslef
fort["X1 at j.",'x1j,"volc.sgf"]

x1k : x1kc + dx1k + d1dlek dslef
fort["X1 at k.",'x1k,"volc.sgf"]

x1l : x1lc + dx1l + d1dlel dslee
fort["X1 at l.",'x1l,"volc.sgf"]

x2e : x2ec + d2dlee dslee
fort["X2 at e.",'x2e,"volc.sgf"]

x2f : x2fc + d2dlef dslef
fort["X2 at f.",'x2f,"volc.sgf"]

x2g : x2gc + d2dleg dslef
fort["X2 at g.",'x2g,"volc.sgf"]

x2h : x2hc + d2dleh dslee
fort["X2 at h.",'x2h,"volc.sgf"]

x2i : x2ic + d2dlei dslee
fort["X2 at i.",'x2i,"volc.sgf"]

x2j : x2jc + d2dlej dslef
fort["X2 at j.",'x2j,"volc.sgf"]

x2k : x2kc + d2dlek dslef
fort["X2 at k.",'x2k,"volc.sgf"]

x2l : x2lc + d2dlel dslee
```

```
fort["X2 at l.",'x2l,"volc.sgf"]

ze : x3ec + d3dlee dslee
fort["Z or x3 at e.",'ze,"volc.sgf"]

zf : x3fc + d3dlef dslef
fort["Z or x3 at f.",'zf,"volc.sgf"]

zg : x3gc + d3dleg dslef
fort["Z or x3 at g.",'zg,"volc.sgf"]

zh : x3hc + d3dleh dslee
fort["Z or x3 at h.",'zh,"volc.sgf"]

zi : x3ic + d3dlei dslee
fort["Z or x3 at i.",'zi,"volc.sgf"]

zj : x3jc + d3dlej dslef
fort["Z or x3 at j.",'zj,"volc.sgf"]

zk : x3kc + d3dlek dslef
fort["Z or x3 at k.",'zk,"volc.sgf"]

zl : x3lc + d3dlel dslee
fort["Z or x3 at l.",'zl,"volc.sgf"]

xe: x2e Sin[x1e]
fort["X at e calculated.",'xe,"volc.sgf"]

xf: x2f Sin[x1f]
fort["X at f calculated.",'xf,"volc.sgf"]

xg: x2g Sin[x1g]
fort["X at g calculated.",'xg,"volc.sgf"]

xh: x2h Sin[x1h]
fort["X at h calculated.",'xh,"volc.sgf"]

xi: x2i Sin[x1i]
fort["X at i calculated.",'xi,"volc.sgf"]

xj: x2j Sin[x1j]
fort["X at j calculated.",'xj,"volc.sgf"]

xk: x2k Sin[x1k]
fort["X at k calculated.",'xk,"volc.sgf"]

xl: x2l Sin[x1l]
fort["X at l calculated.",'xl,"volc.sgf"]

ye: x2e Cos[x1e]
fort["Y at e calculated.",'ye,"volc.sgf"]

yf: x2f Cos[x1f]
fort["Y at f calculated.",'yf,"volc.sgf"]

yg: x2g Cos[x1g]
fort["Y at g calculated.",'yg,"volc.sgf"]

yh: x2h Cos[x1h]
fort["Y at h calculated.",'yh,"volc.sgf"]

yi: x2i Cos[x1i]
fort["Y at i calculated.",'yi,"volc.sgf"]

yj: x2j Cos[x1j]
```

```
fort["Y at j calculated.",'yj,"volc.sgf"]

yk: x2k Cos[x1k]
fort["Y at k calculated.",'yk,"volc.sgf"]

yl: x2l Cos[x1l]
fort["Y at l calculated.",'yl,"volc.sgf"]

dx1 : ((xg + xh + xk + xl) - (xe + xf + xi + xj))/4
fort["Delta x for vector 1",'dx1,"volc.sgf"]

dy1 : ((yg + yh + yk + yl) - (ye + yf + yi + yj))/4
fort["Delta y for vector 1",'dy1,"volc.sgf"]

dz1 : ((zg + zh + zk + zl) - (ze + zf + zi + zj))/4
fort["Delta z for vector 1",'dz1,"volc.sgf"]

dx2 : ((xf + xg + xj + xk) - (xe + xh + xi + xl))/4
fort["Delta x for vector 2",'dx2,"volc.sgf"]

dy2 : ((yf + yg + yj + yk) - (ye + yh + yi + yl))/4
fort["Delta y for vector 2",'dy2,"volc.sgf"]

dz2 : ((zf + zg + zj + zk) - (ze + zh + zi + zl))/4
fort["Delta z for vector 2",'dz2,"volc.sgf"]

dx3 : ((xi + xj + xk + xl) - (xe + xf + xg + xh))/4
fort["Delta x for vector 3",'dx3,"volc.sgf"]

dy3 : ((yi + yj + yk + yl) - (ye + yf + yg + yh))/4
fort["Delta y for vector 3",'dy3,"volc.sgf"]

dz3 : ((zi + zj + zk + zl) - (ze + zf + zg + zh))/4
fort["Delta z for vector 3",'dz3,"volc.sgf"]

vol : (dy1 dz2 - dy2 dz1)dx3 + (dx2 dz1 - dx1 dz2)dy3 +\
      (dx1 dy2 - dx2 dy1)dz3
fort["Volume",'vol,"volc.sgf"]

Lpr["
      VOLV = VOL
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rvolcdec.sgf;* Rvolcdec.osf
!dele Rvolcdec.sgf;*
!@cgf Rvolcdec.sgf

<"decfile.in"

decfile["Rvolcdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]
```

```
/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lvolcdec.sgf;* Lvolcdec.osf
!dele Lvolcdec.sgf;*
!@cgf Lvolcdec.sgf

filename : "volc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LVOLC(DFXDV,DFYDV,DFZDV)
C     Left Volume calculation Cylindrical.
C
C     This routine calculates the contribution of the volume for a cylindrical
C     grid to the left hand sides of the x, y, and z momentum eqs.
C     The variables in common are already initialized for the volume.
C
      REAL DFXDV,DFYDV,DFZDV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LVOLCDEC.FOR'

C     Declare and set the moving grid variables to 0.
      REAL DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L
      REAL DSLEF,DSLEE
      DATA DX1E,DX1F,DX1G,DX1H,DX1I,DX1J,DX1K,DX1L/8*0./
      DATA DSLEF,DSLEE/2*0./

      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx1e,dx1f,dx1g,dx1h,dx1i,dx1j,dx1k,dx1l,dslef,dslee}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx1e[Const] : 1 ; _dx1f[Const] : 1 ; _dx1g[Const] : 1 ; _dx1h[Const] : 1
_dx1i[Const] : 1 ; _dx1j[Const] : 1 ; _dx1k[Const] : 1 ; _dx1l[Const] : 1
_dslef[Const] : 1 ; _dslee[Const] : 1
_x1ec[Const] : 1 ; _x1fc[Const] : 1 ; _x1gc[Const] : 1 ; _x1hc[Const] : 1
_x2ec[Const] : 1 ; _x2fc[Const] : 1 ; _x2gc[Const] : 1 ; _x2hc[Const] : 1
_x3ec[Const] : 1 ; _x3fc[Const] : 1 ; _x3gc[Const] : 1 ; _x3hc[Const] : 1
_x1ic[Const] : 1 ; _x1jc[Const] : 1 ; _x1kc[Const] : 1 ; _x1lc[Const] : 1
_x2ic[Const] : 1 ; _x2jc[Const] : 1 ; _x2kc[Const] : 1 ; _x2lc[Const] : 1
_x3ic[Const] : 1 ; _x3jc[Const] : 1 ; _x3kc[Const] : 1 ; _x3lc[Const] : 1
_d1dlee[Const] : 1 ; _d1dlef[Const] : 1
_d1dleg[Const] : 1 ; _d1dleh[Const] : 1
_d1dlei[Const] : 1 ; _d1dlej[Const] : 1
_d1dlek[Const] : 1 ; _d1dlel[Const] : 1
_d2dlee[Const] : 1 ; _d2dlef[Const] : 1
_d2dleg[Const] : 1 ; _d2dleh[Const] : 1
_d2dlei[Const] : 1 ; _d2dlej[Const] : 1
_d2dlek[Const] : 1 ; _d2dlel[Const] : 1
```

```
_d3dlee[Const] : 1 ; _d3dlef[Const] : 1
_d3dleg[Const] : 1 ; _d3dleh[Const] : 1
_d3dlei[Const] : 1 ; _d3dlej[Const] : 1
_d3dlek[Const] : 1 ; _d3dlel[Const] : 1

/* Algorithm */

x1e : x1ec + dx1e + d1dlee dslee
deriv["X1 at e.",'x1e,"volc.sgf"]

x1f : x1fc + dx1f + d1dlef dslef
deriv["X1 at f.",'x1f,"volc.sgf"]

x1g : x1gc + dx1g + d1dleg dslef
deriv["X1 at g.",'x1g,"volc.sgf"]

x1h : x1hc + dx1h + d1dleh dslee
deriv["X1 at h.",'x1h,"volc.sgf"]

x1i : x1ic + dx1i + d1dlei dslee
deriv["X1 at i.",'x1i,"volc.sgf"]

x1j : x1jc + dx1j + d1dlej dslef
deriv["X1 at j.",'x1j,"volc.sgf"]

x1k : x1kc + dx1k + d1dlek dslef
deriv["X1 at k.",'x1k,"volc.sgf"]

x1l : x1lc + dx1l + d1dlel dslee
deriv["X1 at l.",'x1l,"volc.sgf"]

x2e : x2ec + d2dlee dslee
deriv["X2 at e.",'x2e,"volc.sgf"]

x2f : x2fc + d2dlef dslef
deriv["X2 at f.",'x2f,"volc.sgf"]

x2g : x2gc + d2dleg dslef
deriv["X2 at g.",'x2g,"volc.sgf"]

x2h : x2hc + d2dleh dslee
deriv["X2 at h.",'x2h,"volc.sgf"]

x2i : x2ic + d2dlei dslee
deriv["X2 at i.",'x2i,"volc.sgf"]

x2j : x2jc + d2dlej dslef
deriv["X2 at j.",'x2j,"volc.sgf"]

x2k : x2kc + d2dlek dslef
deriv["X2 at k.",'x2k,"volc.sgf"]

x2l : x2lc + d2dlel dslee
deriv["X2 at l.",'x2l,"volc.sgf"]

ze : x3ec + d3dlee dslee
deriv["Z or x3 at e.",'ze,"volc.sgf"]

zf : x3fc + d3dlef dslef
deriv["Z or x3 at f.",'zf,"volc.sgf"]

zg : x3gc + d3dleg dslef
deriv["Z or x3 at g.",'zg,"volc.sgf"]

zh : x3hc + d3dleh dslee
deriv["Z or x3 at h.",'zh,"volc.sgf"]
```

```
zi : x3ic + d3dlei dslee
deriv["Z or x3 at i.",'zi,"volc.sgf"]

zj : x3jc + d3dlej dslef
deriv["Z or x3 at j.",'zj,"volc.sgf"]

zk : x3kc + d3dlek dslef
deriv["Z or x3 at k.",'zk,"volc.sgf"]

zl : x3lc + d3dlel dslee
deriv["Z or x3 at l.",'zl,"volc.sgf"]

xe: x2e Sin[x1e]
deriv["X at e calculated.",'xe,"volc.sgf"]

xf: x2f Sin[x1f]
deriv["X at f calculated.",'xf,"volc.sgf"]

xg: x2g Sin[x1g]
deriv["X at g calculated.",'xg,"volc.sgf"]

xh: x2h Sin[x1h]
deriv["X at h calculated.",'xh,"volc.sgf"]

xi: x2i Sin[x1i]
deriv["X at i calculated.",'xi,"volc.sgf"]

xj: x2j Sin[x1j]
deriv["X at j calculated.",'xj,"volc.sgf"]

xk: x2k Sin[x1k]
deriv["X at k calculated.",'xk,"volc.sgf"]

xl: x2l Sin[x1l]
deriv["X at l calculated.",'xl,"volc.sgf"]

ye: x2e Cos[x1e]
deriv["Y at e calculated.",'ye,"volc.sgf"]

yf: x2f Cos[x1f]
deriv["Y at f calculated.",'yf,"volc.sgf"]

yg: x2g Cos[x1g]
deriv["Y at g calculated.",'yg,"volc.sgf"]

yh: x2h Cos[x1h]
deriv["Y at h calculated.",'yh,"volc.sgf"]

yi: x2i Cos[x1i]
deriv["Y at i calculated.",'yi,"volc.sgf"]

yj: x2j Cos[x1j]
deriv["Y at j calculated.",'yj,"volc.sgf"]

yk: x2k Cos[x1k]
deriv["Y at k calculated.",'yk,"volc.sgf"]

yl: x2l Cos[x1l]
deriv["Y at l calculated.",'yl,"volc.sgf"]

dx1 : ((xg + xh + xk + xl) - (xe + xf + xi + xj))/4
deriv["Delta x for vector 1",'dx1,"volc.sgf"]

dy1 : ((yg + yh + yk + yl) - (ye + yf + yi + yj))/4
deriv["Delta y for vector 1",'dy1,"volc.sgf"]
```

```
dz1 : ((zg + zh + zk + zl) - (ze + zf + zi + zj))/4
deriv["Delta z for vector 1",'dz1,"volc.sgf"]

dx2 : ((xf + xg + xj + xk) - (xe + xh + xi + xl))/4
deriv["Delta x for vector 2",'dx2,"volc.sgf"]

dy2 : ((yf + yg + yj + yk) - (ye + yh + yi + yl))/4
deriv["Delta y for vector 2",'dy2,"volc.sgf"]

dz2 : ((zf + zg + zj + zk) - (ze + zh + zi + zl))/4
deriv["Delta z for vector 2",'dz2,"volc.sgf"]

dx3 : ((xi + xj + xk + xl) - (xe + xf + xg + xh))/4
deriv["Delta x for vector 3",'dx3,"volc.sgf"]

dy3 : ((yi + yj + yk + yl) - (ye + yf + yg + yh))/4
deriv["Delta y for vector 3",'dy3,"volc.sgf"]

dz3 : ((zi + zj + zk + zl) - (ze + zf + zg + zh))/4
deriv["Delta z for vector 3",'dz3,"volc.sgf"]

vol : (dy1 dz2 - dy2 dz1)dx3 + (dx2 dz1 - dx1 dz2)dy3 +\
      (dx1 dy2 - dx2 dy1)dz3
deriv["Volume",'vol,"volc.sgf"]

/* Include in exprelist the dependence of vol on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[vol,dii]; \
]

Lpr["
C     The dependence on delta x1 at volume nodes.
      CALL LDX1VN(LE,DFXDV*VOLD1,DFYDV*VOLD1,DFZDV*VOLD1)

      CALL LDX1VN(LF,DFXDV*VOLD2,DFYDV*VOLD2,DFZDV*VOLD2)

      CALL LDX1VN(LG,DFXDV*VOLD3,DFYDV*VOLD3,DFZDV*VOLD3)

      CALL LDX1VN(LH,DFXDV*VOLD4,DFYDV*VOLD4,DFZDV*VOLD4)

      CALL LDX1VN(LI,DFXDV*VOLD5,DFYDV*VOLD5,DFZDV*VOLD5)

      CALL LDX1VN(LJ,DFXDV*VOLD6,DFYDV*VOLD6,DFZDV*VOLD6)

      CALL LDX1VN(LK,DFXDV*VOLD7,DFYDV*VOLD7,DFZDV*VOLD7)

      CALL LDX1VN(LL,DFXDV*VOLD8,DFYDV*VOLD8,DFZDV*VOLD8)

C     The dependence on delta sble upper.
      CALL LSBLEN(JSLEU,NJJ,DFXDV*VOLD9,DFYDV*VOLD9,DFZDV*VOLD9)

C     The dependence on delta sble lower.
      CALL LSBLEN(JSLEL,NJJ,DFXDV*VOLD10,DFYDV*VOLD10,DFZDV*VOLD10)

",filename]

Lpr["

      RETURN
      END
",filename]
```

```
<"decfile.in"

decfile["Lvolcdec.sgf"]
```

## File DX1VN.IN

```
/* This session creates the subroutine which calculates the moving grid
delta theta at a volume node.  The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy dx1vn.sgf;* dx1vn.osf
!dele dx1vn.sgf;*
!@cgf dx1vn.sgf

filename : "dx1vn.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RDX1VN(JN,DX1V)
C     Right moving grid Delta X1 or delta theta at a Volume Node.
C
C     This routine calculates the moving grid delta x1 or delta theta
C     at a volume node.
C
      INTEGER JN
      REAL DX1V

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RDX1VNDEC.FOR'

      CALL IDX1VN(JN)

      IF(JVN.EQ.-1)THEN
        DX1V = 0.
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

dx1case : {(dx11n + dx12n)/2,\
           (3dx11n + 3dx13n - dx12n - dx14n)/4,\
           (dx11n + dx12n + dx13n + dx14n)/4}
dx1cond : {"JVN.EQ.0",\
          "JVN.EQ.1",\
          ".TRUE."}
casefort[3,"Delta x1 at a volume node",dx1,dx1case,dx1cond,"dx1vn.sgf"]

Lpr["
      DX1V = DX1
",filename]

Lpr["
C
      RETURN
      END
```

```
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rdx1vndec.sgf;* Rdx1vndec.osf
!dele Rdx1vndec.sgf;*
!@cgf Rdx1vndec.sgf

<"decfile.in"

decfile["Rdx1vndec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "dx1vn.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LDX1VN(JN,DFXDX1,DFYDX1,DFZDX1)
C     Left moving grid Delta X1 or delta theta at a Volume Node.
C
C     This routine calculates the contribution of the moving grid delta x1
C     or delta theta at a volume node to the x, y, or z momentum eqs.
C
      INTEGER JN
      REAL DFXDX1,DFYDX1,DFZDX1

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LDX1VNDEC.FOR'
      REAL DERI
      REAL COEF

      CALL IDX1VN(JN)

      IF(JVN.EQ.-1)THEN
        RETURN
      END IF
",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx11n,dx12n,dx13n,dx14n}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx11n[Const] : 1 ; _dx12n[Const] : 1 ; _dx13n[Const] : 1 ; _dx14n[Const] : 1

/* Algorithm */

dx1case : {(dx11n + dx12n)/2,\
           (3dx11n + 3dx13n - dx12n - dx14n)/4,\
           (dx11n + dx12n + dx13n + dx14n)/4}
dx1cond : {"JVN.EQ.0",\
          "JVN.EQ.1",\
          ".TRUE."}
casederiv[3,"Delta x1 at a volume node",dx1,dx1case,dx1cond,"dx1vn.sgf"]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C
C     delta theta at 1
      COEF = DFXDX1*DX1D1
      CALL ADCOFS(COEF,IDTH1+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 2
      COEF = DFXDX1*DX1D2
      CALL ADCOFS(COEF,IDTH2+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 3
      COEF = DFXDX1*DX1D3
      CALL ADCOFS(COEF,IDTH3+IDTHO,COFX,ICOLX,NCX)
C
C     delta theta at 4
      COEF = DFXDX1*DX1D4
      CALL ADCOFS(COEF,IDTH4+IDTHO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C
C     delta theta at 1
      COEF = DFYDX1*DX1D1
      CALL ADCOFS(COEF,IDTH1+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 2
      COEF = DFYDX1*DX1D2
      CALL ADCOFS(COEF,IDTH2+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 3
      COEF = DFYDX1*DX1D3
      CALL ADCOFS(COEF,IDTH3+IDTHO,COFY,ICOLY,NCY)
C
C     delta theta at 4
      COEF = DFYDX1*DX1D4
      CALL ADCOFS(COEF,IDTH4+IDTHO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
```

```
C
C     delta theta at 1
      COEF = DFZDX1*DX1D1
      CALL ADCOFS(COEF,IDTH1+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 2
      COEF = DFZDX1*DX1D2
      CALL ADCOFS(COEF,IDTH2+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 3
      COEF = DFZDX1*DX1D3
      CALL ADCOFS(COEF,IDTH3+IDTHO,COFZ,ICOLZ,NCZ)
C
C     delta theta at 4
      COEF = DFZDX1*DX1D4
      CALL ADCOFS(COEF,IDTH4+IDTHO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Ldx1vndec.sgf;* Ldx1vndec.osf
!dele Ldx1vndec.sgf;*
!@cgf Ldx1vndec.sgf

<"decfile.in"

decfile["Ldx1vndec.sgf"]
```

## File ENTEQ.IN

```
/* This session creates the subroutine which calculates the left side of
the entropy convection equation.  */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy enteq.sgf;* enteq.osf
!dele enteq.sgf;*
!@cgf enteq.sgf

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy enteqdec.sgf;* enteqdec.osf
!dele enteqdec.sgf;*
!@cgf enteqdec.sgf

filename : "enteq.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE ENTEQ(I,J,K,NI,NJ)
C     ENTropy convection EQuation for an S-face.
C
C     This routine calculates the left side of the entropy convection
C     equation.
C
      INTEGER I,J,K,NI,NJ

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'ENTEQDEC.FOR'
      REAL DERI
      REAL COEF

      REAL RESD11,RESD12,RESD15,RESD16

C     Initialize the variables in the FACE common.
      CALL IENTEQ(I,J,K)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {sd,s1,s2,s3,s4,s5,asd,as1,as2,as3,as4,as5,\
 bsd,bs1,bs2,bs3,bs4,bs5}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_sd[Const] : 1 ; _s1[Const] : 1 ; _s2[Const] : 1 ; _s3[Const] : 1
_s4[Const] : 1 ; _s5[Const] : 1
_asd[Const] : 1 ; _as1[Const] : 1 ; _as2[Const] : 1 ; _as3[Const] : 1
_as4[Const] : 1 ; _as5[Const] : 1
_bsd[Const] : 1 ; _bs1[Const] : 1 ; _bs2[Const] : 1 ; _bs3[Const] : 1
_bs4[Const] : 1 ; _bs5[Const] : 1

/* Algorithm */

ds1case : {0,\
           s3 - s1,\
           s1 - s2,\
           s3 - s2}
ds1cond : {"NI.EQ.2",\
          "I.EQ.1",\
          "I.EQ.NI-1",\
          ".TRUE."}
casederiv[4,"Delta entropy in xi1.",ds1,ds1case,ds1cond,"enteq.sgf"]

dascase : {1,\
           as3 - as1,\
           as1 - as2,\
           as3 - as2}
dascond : {"NI.EQ.2",\
          "I.EQ.1",\
          "I.EQ.NI-1",\
          ".TRUE."}
casederiv[4,"Delta psi1 in xi1.",das,dascase,dascond,"enteq.sgf"]

ds2case : {0,\
           s5 - s1,\
           s1 - s4,\
           s5 - s4}
ds2cond : {"NJ.EQ.2",\
          "J.EQ.1",\
          "J.EQ.NJ-1",\
          ".TRUE."}
casederiv[4,"Delta entropy in xi2.",ds2,ds2case,ds2cond,"enteq.sgf"]
```

```
dbscase : {1,\
           bs5 - bs1,\
           bs1 - bs4,\
           bs5 - bs4}
dbscond : {"NJ.EQ.2",\
          "J.EQ.1",\
          "J.EQ.NJ-1",\
          ".TRUE."}
casederiv[4,"Delta psi2 in xi2.",dbs,dbscase,dbscond,"enteq.sgf"]

res : s1 - sd + ds1/das (asd-as1) + ds2/dbs(bsd-bs1)
deriv["Entropy interp error",'res,"enteq.sgf"]

/* Memory managment */
Lpr[Mem[]]
exprlist : Union[exprlist]
nel : Len[exprlist]
Gc[]
Lpr[Mem[]]

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Set NCX.
      NCX = 0
C
C     Entropy and Stream function at S faces.
      IF(CMPRS)THEN
        CALL LENTC(ISFD,RESD1)
        CALL LSAB(ISFD,RESD7,RESD13)

        CALL LENTC(ISF1,RESD2)
        CALL LSAB(ISF1,RESD8,RESD14)

        IF(I.NE.1)THEN
          CALL LENTC(ISF2,RESD3)
          CALL LSAB(ISF2,RESD9,RESD15)
        END IF

        IF(I.NE.NI-1)THEN
          CALL LENTC(ISF3,RESD4)
          CALL LSAB(ISF3,RESD10,RESD16)
        END IF

        IF(J.NE.1)THEN
          CALL LENTC(ISF4,RESD5)
          CALL LSAB(ISF4,RESD11,RESD17)
        END IF

        IF(J.NE.NJ-1)THEN
          CALL LENTC(ISF5,RESD6)
          CALL LSAB(ISF5,RESD12,RESD18)
        END IF
      ELSE
        CALL LENT(ISFD,RESD1)
        CALL LSAB(ISFD,RESD7,RESD13)

        CALL LENT(ISF1,RESD2)
        CALL LSAB(ISF1,RESD8,RESD14)

        IF(I.NE.1)THEN
          CALL LENT(ISF2,RESD3)
          CALL LSAB(ISF2,RESD9,RESD15)
        END IF
        IF(I.NE.NI-1)THEN
          CALL LENT(ISF3,RESD4)
          CALL LSAB(ISF3,RESD10,RESD16)
        END IF

        IF(J.NE.1)THEN
          CALL LENT(ISF4,RESD5)
          CALL LSAB(ISF4,RESD11,RESD17)
        END IF

        IF(J.NE.NJ-1)THEN
          CALL LENT(ISF5,RESD6)
          CALL LSAB(ISF5,RESD12,RESD18)
        END IF
      END IF

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["enteqdec.sgf"]
```

## File ENT.IN

```
/* This session creates the subroutine which calculates the entropy
at an S face.  The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy ent.sgf;* ent.osf
!dele ent.sgf;*
!@cgf ent.sgf

filename : "ent.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RENT(ISF,ENT)
C     Right calculation of entropy for an s-face.
C
C     This routine calculates the entropy at the S face ISF.
C
      INTEGER ISF
      REAL ENT

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RENTDEC.FOR'

C     Declare the contravarient velocities.
      REAL RWX,RWY,RWZ

C     Declare the radius and theta.
      REAL R,X1V
C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

C     Get the radius and theta.
      CALL RRTHFC(R,X1V)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the velocities.
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

rws : rwx^2 + rwy^2 + rwz^2
fort["(rho W)^2",'rws,"ent.sgf"]

/* Incompressible flow */
rho : roc
fort["density",'rho,"ent.sgf"]

s : p + rws/(2 rho) - rho omg^2 r^2/2
fort["entropy",'s,"ent.sgf"]

Lpr["
      ENT = S

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rentdec.sgf;* Rentdec.osf
!dele Rentdec.sgf;*
!@cgf Rentdec.sgf

<"decfile.in"

decfile["Rentdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "ent.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"
```

```
Lpr["
      SUBROUTINE LENT(ISF,DEQDS)
C     Left Entropy for an s-face.
C
C     This routine multiplies DEQDS by the entropy coefficients and adds
C     them to the COLX arrays.
C
      INTEGER ISF
      REAL DEQDS

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LENTDEC.FOR'
      REAL DERI
C     Declare the contravarient velocities.
      REAL RWX,RWY,RWZ

C     Declare the radius and theta.
      REAL R,X1V

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

C     Get the radius and theta.
      CALL RRTHFC(R,X1V)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,rwx,rwy,rwz,r}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_roc[Const] : 1
_r[Const] : 1
_omg[Const] : 1

/* Algorithm */

Lpr["

C     Get the velocities.
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

rws : rwx^2 + rwy^2 + rwz^2
deriv["(rho W)^2",'rws,"ent.sgf"]

/* Incompressible flow */
rho : roc
deriv["density",'rho,"ent.sgf"]

s : p + rws/(2 rho) - rho omg^2 r^2/2
```

```
deriv["entropy",'s,"ent.sgf"]

/* Coefficients stored in X - Momentum Equation arrays. */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation array
C     NCX must be set.
C
C     ps
      CALL ADCOFS(DEQDS*SD1,IPSO+IP,COFX,ICOLX,NCX)
",filename]

/* Y and Z - Momentum Equation arrays. */

Lpr["
C
C     NCY and NCZ must be set.
      NCY = 0
      NCZ = 0

",filename]

Lpr["
C     The dependence on the velocities.  Put in x-momentum eq arrays.
      IF(PLANER)THEN
        CALL LVELP(3,DEQDS*SD2,DEQDS*SD3,DEQDS*SD4, 0.,0.,0., 0.,0.,0.)
      ELSE
        CALL LVELC(3,DEQDS*SD2,DEQDS*SD3,DEQDS*SD4, 0.,0.,0., 0.,0.,0.)
      END IF

C     The dependence on the radius.
      CALL LRTHFC(DEQDS*SD5,0., 0.,0., 0.,0.)

",filename]

Lpr["


      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lentdec.sgf;* Lentdec.osf
!dele Lentdec.sgf;*
!@cgf Lentdec.sgf

<"decfile.in"

decfile["Lentdec.sgf"]
```

## File AUXP.IN

```
/* This session creates the subroutine which calculates the residuals to
the auxilliary pressure equations.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
```

```
!copy auxp.sgf;* auxp.osf
!dele auxp.sgf;*
!@cgf auxp.sgf

filename : "auxp.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RAUXP(M)
C     Right AUXilliary Pressure equations for volume M.
C
C     This routine calculates the residual for the A and B auxilliary pressure
C     equations for volume M.  The A equation is in RESX and the B equation
C     is in RESY.
C
      INTEGER M

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RAUXPDEC.FOR'

C     Declare the velocities.
      REAL RWX1,RWY1,RWZ1,RWX2,RWY2,RWZ2

C     Declare the psi1 and psi2 curvature terms.
      REAL SI1C,SI2C

C     Initialize the variables for this equation.
      CALL IAUXP(M)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the velocities for S face 1.
      CALL IMS(ISF1)
      IF(PLANER)THEN
        CALL RVELP(3,RWX1,RWY1,RWZ1)
      ELSE
        CALL RVELC(3,RWX1,RWY1,RWZ1)
      END IF

C     Get the velocities for S face 2.
      CALL IMS(ISF2)
      IF(PLANER)THEN
        CALL RVELP(3,RWX2,RWY2,RWZ2)
      ELSE
        CALL RVELC(3,RWX2,RWY2,RWZ2)
      END IF

C     Get the psi1 curvature term.
      CALL RSI1CT(SI1C)

C     Get the psi2 curvature term.
      CALL RSI2CT(SI2C)

",filename]
```

```
ws1 : rwx1^2 + rwy1^2 + rwz1^2
fort["(rho W)^2 at s face 1.",'ws1,"auxp.sgf"]

ws2 : rwx2^2 + rwy2^2 + rwz2^2
fort["(rho W)^2 at s face 2.",'ws2,"auxp.sgf"]

wsb : (ws1 + ws2)/2
fort["(rho W)^2 average.",'wsb,"auxp.sgf"]

/* Incompressible flow */
rho : roc
fort["density",'rho,"auxp.sgf"]

pca : -4 ka wsb/rho si1c
fort["2 P_c for the A equation.",'pca,"auxp.sgf"]

pcb : -4 kb wsb/rho si2c
fort["2 P_c for the A equation.",'pcb,"auxp.sgf"]

aer : ps1 + ps2 - pa1 - pa2 - pca
fort["residual for A equation. (A error)",'aer,"auxp.sgf"]

ber : ps1 + ps2 - pb1 - pb2 - pcb
fort["residual for B equation. (B error)",'ber,"auxp.sgf"]

Lpr["
      RESX = -AER
      RESY = -BER

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rauxpdec.sgf;* Rauxpdec.osf
!dele Rauxpdec.sgf;*
!@cgf Rauxpdec.sgf

<"decfile.in"

decfile["Rauxpdec.sgf"]

/* Create the declaration file for the variables in exprlist which
will be output at the end of this routine.  This is done here rather
than at the end of the routine because of an error that was occuring.*/

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lauxpdec.sgf;* Lauxpdec.osf
!dele Lauxpdec.sgf;*
!@cgf Lauxpdec.sgf

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "auxp.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
```

```
<"casederiv.in"

Lpr["
      SUBROUTINE LAUXP(M)
C     Left AUXilliary Pressure equations for volume M.
C
C     This routine calculates the left hand side coefficients of the Jaqcobian
C     matrix for the A and B auxilliary pressure
C     equations for volume M.  The A equation is in the COFX arrays and
C     the B equation is in the COFY arrays.
C
      INTEGER M

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LAUXPDEC.FOR'
      REAL DERI

C     Declare the velocities.
      REAL RWX1,RWY1,RWZ1,RWX2,RWY2,RWZ2

C     Declare the psi1 and psi2 curvature terms.
      REAL SI1C,SI2C

C     Initialize the variables for this equation.
      CALL IAUXP(M)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {ps1,ps2,pa1,pa2,pb1,pb2,rwx1,rwy1,rwz1,rwx2,rwy2,rwz2,si1c,si2c}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_ps1[Const] : 1 ; _ps2[Const] : 1 ; _pa1[Const] : 1 ; _pa2[Const] : 1
_pb1[Const] : 1 ; _pb2[Const] : 1
_rwx1[Const] : 1 ; _rwy1[Const] : 1 ; _rwz1[Const] : 1
_rwx2[Const] : 1 ; _rwy2[Const] : 1 ; _rwz2[Const] : 1
_si1c[Const] : 1 ; _si2c[Const] : 1
_roc[Const] : 1 ; _ka[Const] : 1 ; _kb[Const] : 1
_h1[Const] : 1 ; _h2[Const] : 1 ; _or21[Const] : 1 ; _or22[Const] : 1
_gam[Const] : 1

/* Algorithm */

Lpr["

C     Get the velocities for S face 1.
      CALL IMS(ISF1)
      IF(PLANER)THEN
        CALL RVELP(3,RWX1,RWY1,RWZ1)
      ELSE
        CALL RVELC(3,RWX1,RWY1,RWZ1)
      END IF

C     Get the velocities for S face 2.
      CALL IMS(ISF2)
      IF(PLANER)THEN
        CALL RVELP(3,RWX2,RWY2,RWZ2)
      ELSE
        CALL RVELC(3,RWX2,RWY2,RWZ2)
      END IF
```

```
C     Get the psi1 curvature term.
      CALL RSI1CT(SI1C)

C     Get the psi2 curvature term.
      CALL RSI2CT(SI2C)

",filename]

ws1 : rwx1^2 + rwy1^2 + rwz1^2
deriv["(rho W)^2 at s face 1.",'ws1,"auxp.sgf"]

ws2 : rwx2^2 + rwy2^2 + rwz2^2
deriv["(rho W)^2 at s face 2.",'ws2,"auxp.sgf"]

wsb : (ws1 + ws2)/2
deriv["(rho W)^2 average.",'wsb,"auxp.sgf"]

/* Incompressible flow */
rho : roc
deriv["density",'rho,"auxp.sgf"]

pca : -4 ka wsb/rho si1c
deriv["2 P_c for the A equation.",'pca,"auxp.sgf"]

pcb : -4 kb wsb/rho si2c
deriv["2 P_c for the B equation.",'pcb,"auxp.sgf"]

aer : ps1 + ps2 - pa1 - pa2 - pca
deriv["residual for A equation. (A error)",'aer,"auxp.sgf"]

ber : ps1 + ps2 - pb1 - pb2 - pcb
deriv["residual for B equation. (B error)",'ber,"auxp.sgf"]

/* A axilliary equation stored in X - Momentum Equation arrays. */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation array
C     NCX must be set.
      NCX = 0
C
C     ps1
      CALL ADCOFS(AERD1,IPSO+ISF1,COFX,ICOLX,NCX)
C
C     ps2
      CALL ADCOFS(AERD2,IPSO+ISF2,COFX,ICOLX,NCX)
C
C     pa1
      CALL ADCOFS(AERD3,IPAO+IPA1,COFX,ICOLX,NCX)
C
C     pa2
      CALL ADCOFS(AERD4,IPAO+IPA2,COFX,ICOLX,NCX)
",filename]

/* B axilliary equation stored in Y - Momentum Equation arrays. */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation array
C     NCY must be set.
      NCY = 0
C
C     ps1
```

```
      CALL ADCOFS(BERD1,IPSO+ISF1,COFY,ICOLY,NCY)
C
C     ps2
      CALL ADCOFS(BERD2,IPSO+ISF2,COFY,ICOLY,NCY)
C
C     pb1
      CALL ADCOFS(BERD5,IPBO+IPB1,COFY,ICOLY,NCY)
C
C     pb2
      CALL ADCOFS(BERD6,IPBO+IPB2,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation arrays. */

Lpr["
C
C     NCZ must be set.
      NCZ = 0

",filename]

Lpr["
C     The dependence on the velocities for S face 2.
C     Put A equation in x-momentum eq arrays and B eq in y-momentum eq arrays.
C     S face 2 is still initialized.
      IF(PLANER)THEN
        CALL LVELP(3,AERD10,AERD11,AERD12, BERD10,BERD11,BERD12,
     1               0.,0.,0.)
      ELSE
        CALL LVELC(3,AERD10,AERD11,AERD12, BERD10,BERD11,BERD12,
     1               0.,0.,0.)
      END IF

C     The dependence on the velocities for S face 1.
C     Put A equation in x-momentum eq arrays and B eq in y-momentum eq arrays.
      CALL IMS(ISF1)
      IF(PLANER)THEN
        CALL LVELP(3,AERD7,AERD8,AERD9, BERD7,BERD8,BERD9, 0.,0.,0.)
      ELSE
        CALL LVELC(3,AERD7,AERD8,AERD9, BERD7,BERD8,BERD9, 0.,0.,0.)
      END IF

C     The dependence of the A eq on the psi1 curvature term.
      CALL LSI1CT(AERD13,0.)

C     The dependence of the B eq on the psi2 curvature term.
      CALL LSI2CT(0.,BERD14)

",filename]

Lpr["


      RETURN
      END
",filename]

<"decfile.in"

decfile["Lauxpdec.sgf"]
```

## File SI1CT.IN

```
/* This session creates the subroutine which calculates the Psi1 curvature
```

```
term used in the auxilliary pressure equations.  The left hand side
subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy si1ct.sgf;* si1ct.osf
!dele si1ct.sgf;*
!@cgf si1ct.sgf

filename : "si1ct.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RSI1CT(SI1CV)
C     Right pSI1 Curvature Term calculation.
C
C     This routine calculates the psi1 curvature term used in the A
C     auxilliary pressure equation.
C     This routine is called by RAUXP or LAUXP so the variables
C     in common are already initialized.
C
      REAL SI1CV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RSI1CTDEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

si1c : ((ap4 - ap1) - 2(ap5 - ap2) + (ap6 - ap3))/\
        ((ap4 + 2ap5 + ap6) - (ap1 + 2ap2 + ap3))
fort["Psi1 curvature term.",'si1c,"si1ct.sgf"]

Lpr["
      SI1CV = SI1C
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rsi1ctdec.sgf;* Rsi1ctdec.osf
!dele Rsi1ctdec.sgf;*
!@cgf Rsi1ctdec.sgf

<"decfile.in"

decfile["Rsi1ctdec.sgf"]

/* Reinitialize everything to ensure enough room. */
```

```
Set[]

filename : "si1ct.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LSI1CT(DSI1CX,DSI1CY)
C     Left pSI1 Curvature Term calculation.
C
C     This routine calculates the contribution of the psi1 curvature term
C     used in the auxilliary pressure equations
C     stored in the COFX and COFY arrays.
C     This routine is called by LAUXP so the variables
C     in common are already initialized.
C
      REAL DSI1CX,DSI1CY

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LSI1CTDEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {ap1,ap2,ap3,ap4,ap5,ap6}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_ap1[Const] : 1 ; _ap2[Const] : 1 ; _ap3[Const] : 1 ; _ap4[Const] : 1
_ap5[Const] : 1 ; _ap6[Const] : 1

/* Algorithm */

si1c : ((ap4 - ap1) - 2(ap5 - ap2) + (ap6 - ap3))/\
       ((ap4 + 2ap5 + ap6) - (ap1 + 2ap2 + ap3))
deriv["Psi1 curvature term.",'si1c,"si1ct.sgf"]

/* The A auxilliary pressure equation is stored as the X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi1 at 1
      COEF = DSI1CX*SI1CD1
      CALL ADCOFS(COEF,ISI11+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 2
      COEF = DSI1CX*SI1CD2
      CALL ADCOFS(COEF,ISI12+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 3
```

```
      COEF = DSI1CX*SI1CD3
      CALL ADCOFS(COEF,ISI13+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 4
      COEF = DSI1CX*SI1CD4
      CALL ADCOFS(COEF,ISI14+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 5
      COEF = DSI1CX*SI1CD5
      CALL ADCOFS(COEF,ISI15+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 6
      COEF = DSI1CX*SI1CD6
      CALL ADCOFS(COEF,ISI16+IPSI10,COFX,ICOLX,NCX)

",filename]

/* The B auxilliary pressure equation is stored as the Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     Psi1 at 1
      COEF = DSI1CY*SI1CD1
      CALL ADCOFS(COEF,ISI11+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 2
      COEF = DSI1CY*SI1CD2
      CALL ADCOFS(COEF,ISI12+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 3
      COEF = DSI1CY*SI1CD3
      CALL ADCOFS(COEF,ISI13+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 4
      COEF = DSI1CY*SI1CD4
      CALL ADCOFS(COEF,ISI14+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 5
      COEF = DSI1CY*SI1CD5
      CALL ADCOFS(COEF,ISI15+IPSI10,COFY,ICOLY,NCY)
C
C     Psi1 at 6
      COEF = DSI1CY*SI1CD6
      CALL ADCOFS(COEF,ISI16+IPSI10,COFY,ICOLY,NCY)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lsi1ctdec.sgf;* Lsi1ctdec.osf
!dele Lsi1ctdec.sgf;*
!@cgf Lsi1ctdec.sgf
```

```
<"decfile.in"

decfile["Lsi1ctdec.sgf"]
```

## File SI2CT.IN

```
/* This session creates the subroutine which calculates the Psi2 curvature
term used in the auxilliary pressure equations.  The left hand side
subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy si2ct.sgf;* si2ct.osf
!dele si2ct.sgf;*
!@cgf si2ct.sgf

filename : "si2ct.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RSI2CT(SI2CV)
C     Right pSI2 Curvature Term calculation.
C
C     This routine calculates the psi2 curvature term used in the B
C     auxilliary pressure equation.
C     This routine is called by RAUXP or LAUXP so the variables
C     in common are already initialized.
C
      REAL SI2CV

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RSI2CTDEC.FOR'

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

si2c : ((bp4 - bp1) - 2(bp5 - bp2) + (bp6 - bp3))/\
       ((bp4 + 2bp5 + bp6) - (bp1 + 2bp2 + bp3))
fort["Psi2 curvature term.",'si2c,"si2ct.sgf"]

Lpr["
      SI2CV = SI2C
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
```

```
!copy Rsi2ctdec.sgf;* Rsi2ctdec.osf
!dele Rsi2ctdec.sgf;*
!@cgf Rsi2ctdec.sgf

<"decfile.in"

decfile["Rsi2ctdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "si2ct.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LSI2CT(DSI2CX,DSI2CY)
C     Left pSI2 Curvature Term calculation.
C
C     This routine calculates the contribution of the psi2 curvature term
C     used in the auxilliary pressure equations
C     stored in the COFX and COFY arrays.
C     This routine is called by LAUXP so the variables
C     in common are already initialized.
C
      REAL DSI2CX,DSI2CY

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LSI2CTDEC.FOR'
      REAL DERI
      REAL COEF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {bp1,bp2,bp3,bp4,bp5,bp6}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_bp1[Const] : 1 ; _bp2[Const] : 1 ; _bp3[Const] : 1 ; _bp4[Const] : 1
_bp5[Const] : 1 ; _bp6[Const] : 1

/* Algorithm */

si2c : ((bp4 - bp1) - 2(bp5 - bp2) + (bp6 - bp3))/\
       ((bp4 + 2bp5 + bp6) - (bp1 + 2bp2 + bp3))
deriv["Psi2 curvature term.",'si2c,"si2ct.sgf"]

/* The A auxilliary pressure equation is stored as the X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C

C
C     Psi2 at 1
      COEF = DSI2CX*SI2CD1
      CALL ADCOFS(COEF,ISI21+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 2
      COEF = DSI2CX*SI2CD2
      CALL ADCOFS(COEF,ISI22+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 3
      COEF = DSI2CX*SI2CD3
      CALL ADCOFS(COEF,ISI23+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 4
      COEF = DSI2CX*SI2CD4
      CALL ADCOFS(COEF,ISI24+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 5
      COEF = DSI2CX*SI2CD5
      CALL ADCOFS(COEF,ISI25+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 6
      COEF = DSI2CX*SI2CD6
      CALL ADCOFS(COEF,ISI26+IPSI20,COFX,ICOLX,NCX)

",filename]

/* The B auxilliary pressure equation is stored as the Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C

C
C     Psi2 at 1
      COEF = DSI2CY*SI2CD1
      CALL ADCOFS(COEF,ISI21+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 2
      COEF = DSI2CY*SI2CD2
      CALL ADCOFS(COEF,ISI22+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 3
      COEF = DSI2CY*SI2CD3
      CALL ADCOFS(COEF,ISI23+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 4
      COEF = DSI2CY*SI2CD4
      CALL ADCOFS(COEF,ISI24+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 5
      COEF = DSI2CY*SI2CD5
      CALL ADCOFS(COEF,ISI25+IPSI20,COFY,ICOLY,NCY)
C
C     Psi2 at 6
      COEF = DSI2CY*SI2CD6
      CALL ADCOFS(COEF,ISI26+IPSI20,COFY,ICOLY,NCY)

",filename]

Lpr["
      RETURN
      END
",filename]
```

```
/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lsi2ctdec.sgf;* Lsi2ctdec.osf
!dele Lsi2ctdec.sgf;*
!@cgf Lsi2ctdec.sgf

<"decfile.in"

decfile["Lsi2ctdec.sgf"]
```

## File SBLEN.IN

```
/* This session creates the left hand side subroutine for the calculation
of the blade leading edge movement at a volume node. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy sblen.sgf;* sblen.osf
!dele sblen.sgf;*
!@cgf sblen.sgf

filename : "sblen.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LSBLEN(J,NJ,DFXDLE,DFYDLE,DFZDLE)
C     Left S Blade Leading Edge movement at a volume Node.
C
C     This routine calculates the contribution of the leading edge movement
C     at a volume node to the x, y, or z momentum eqs.
C
      INTEGER J,NJ
      REAL DFXDLE,DFYDLE,DFZDLE

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LSBLENDEC.FOR'
      REAL DERI
      REAL COEF

      INTEGER JLEI1,JLEI2
      REAL SBLEI1,SBLEI2
      DATA SBLEI1,SBLEI2/2*0./

      IF(NJ.EQ.2)THEN
        JLEI1 = 1
C       Dummy value.
        JLEI2 = 1
      ELSE IF(J.EQ.1)THEN
        JLEI1 = 1
        JLEI2 = 2
      ELSE IF(J.EQ.NJ)THEN
        JLEI1 = NJ-1
        JLEI2 = NJ-2
      ELSE
        JLEI1 = J - 1
        JLEI2 = J
      END IF

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {sbleil,sblei2}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_sbleil[Const] : 1 ; _sblei2[Const] : 1

/* Algorithm */

sblecase : {sbleil,\
            3/2 sbleil - 1/2 sblei2,\
            (sbleil + sblei2)/2}
sblecond : {"NJ.EQ.2",\
           "J.EQ.1 .OR. J.EQ.NJ",\
           ".TRUE."}
casederiv[3,"SBLE at a volume node",sble,sblecase,sblecond,"sblen.sgf"]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C
C     delta sblei at 1
      COEF = DFXDLE*SBLED1
      CALL ADCOFS(COEF,JLEI1+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 2
      COEF = DFXDLE*SBLED2
      CALL ADCOFS(COEF,JLEI2+ISBLEO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C
C     delta sblei at 1
      COEF = DFYDLE*SBLED1
      CALL ADCOFS(COEF,JLEI1+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 2
      COEF = DFYDLE*SBLED2
      CALL ADCOFS(COEF,JLEI2+ISBLEO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
```

```
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C
C     delta sblei at 1
      COEF = DFZDLE*SBLED1
      CALL ADCOFS(COEF,JLEI1+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 2
      COEF = DFZDLE*SBLED2
      CALL ADCOFS(COEF,JLEI2+ISBLEO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lsblendec.sgf;* Lsblendec.osf
!dele Lsblendec.sgf;*
!@cgf Lsblendec.sgf

<"decfile.in"

decfile["Lsblendec.sgf"]
```

## File LED.IN

```
/* This session creates the left hand side subroutine for the calculation
of the blade leading edge derivatives with respect to xi2. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy led.sgf;* led.osf
!dele led.sgf;*
!@cgf led.sgf

filename : "led.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LLEDA1(DFXDL2,DFYDL2,DFZDL2)
C     Left Leading Edge Derivative wrt xi2 for an A or S face used for
C     calculating psi1.
C
C     This routine calculates the contribution of the leading edge derivative
C     wrt xi2 for an A or S face for psi1 to the x, y, or z momentum eqs.
C
      REAL DFXDL2,DFYDL2,DFZDL2

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LLEDA1DEC.FOR'
      REAL DERI
      REAL COEF

      REAL SBLEI1,SBLEI2,SBLEI3
      DATA SBLEI1,SBLEI2,SBLEI3/3*0./

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {sblei1,sblei2,sblei3}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_sblei1[Const] : 1 ; _sblei2[Const] : 1 ; _sblei3[Const] : 1

/* Algorithm */

sle2case : {(sblei3 - sblei1)/2,\
            (-3sblei2 + 4sblei3 - sblei1)/2,\
            (3sblei2 - 4sblei1 + sblei3)/2,\
            0}
sle2cond : {"MGD.EQ.0",\
           "MGD.EQ.1",\
           "MGD.EQ.2",\
           "MGD.EQ.3"}
casederiv[4,"SBLE derivative wrt xi2",sle2,sle2case,sle2cond,"led.sgf"]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C
C     delta sblei at 1
      COEF = DFXDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 2
      COEF = DFXDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 3
      COEF = DFXDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C
C     delta sblei at 1
      COEF = DFYDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFY,ICOLY,NCY)
```

```
C
C     delta sblei at 2
      COEF = DFYDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 3
      COEF = DFYDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C
C     delta sblei at 1
      COEF = DFZDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 2
      COEF = DFZDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 3
      COEF = DFZDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Llledaldec.sgf;* Lledaldec.osf
!dele Lledaldec.sgf;*
!@cgf Lledaldec.sgf

<"decfile.in"

decfile["Lledaldec.sgf"]

Lpr["
      SUBROUTINE LLEDB1(DFXDL2,DFYDL2,DFZDL2)
C     Left Leading Edge Derivative wrt xi2 for a B face used for
C     calculating psi1.
C
C     This routine calculates the contribution of the leading edge derivative
C     wrt xi2 for a B face for psi1 to the x, y, or z momentum eqs.
C
      REAL DFXDL2,DFYDL2,DFZDL2

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LLEDB1DEC.FOR'
      REAL DERI
      REAL COEF
      REAL SBLEI1,SBLEI2,SBLEI3
      DATA SBLEI1,SBLEI2,SBLEI3/3*0./

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {sblei1,sblei2,sblei3}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_sblei1[Const] : 1 ; _sblei2[Const] : 1 ; _sblei3[Const] : 1

/* Algorithm */

sle2 : sblei2 - sblei1
deriv["SBLE derivative wrt xi2",'sle2,"led.sgf"]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C
C     delta sblei at 1
      COEF = DFXDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 2
      COEF = DFXDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 3
      COEF = DFXDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C
C     delta sblei at 1
      COEF = DFYDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 2
      COEF = DFYDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 3
      COEF = DFYDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFY,ICOLY,NCY)
```

```
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C
C     delta sblei at 1
      COEF = DFZDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 2
      COEF = DFZDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 3
      COEF = DFZDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lledb1dec.sgf;* Lledb1dec.osf
!dele Lledb1dec.sgf;*
!@cgf Lledb1dec.sgf

<"decfile.in"

decfile["Lledb1dec.sgf"]


Lpr["
      SUBROUTINE LLEDA2(DFXDL2,DFYDL2,DFZDL2)
C     Left Leading Edge Derivative wrt xi2 for an A or S face used for
C     calculating psi2.
C
C     This routine calculates the contribution of the leading edge derivative
C     wrt xi2 for an A or S face for psi2 to the x, y, or z momentum eqs.
C
      REAL DFXDL2,DFYDL2,DFZDL2

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LLEDA2DEC.FOR'
      REAL DERI
      REAL COEF

      REAL SBLEI1,SBLEI2,SBLEI3
      DATA SBLEI1,SBLEI2,SBLEI3/3*0./

",filename]

/* Initialize expression list. */
nel : 0
```

```
/* Initialize the dependant variable list. */
invar : {sbleil,sblei2,sblei3}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_sbleil[Const] : 1 ; _sblei2[Const] : 1 ; _sblei3[Const] : 1

/* Algorithm */

sle2case : {(sblei3 - sbleil)/2,\
            sblei3 - sblei2,\
            sblei2 - sbleil,\
            0}
sle2cond : {"MGD.EQ.0",\
           "MGD.EQ.1",\
           "MGD.EQ.2",\
           "MGD.EQ.3"}
casederiv[4,"SBLE derivative wrt xi2",sle2,sle2case,sle2cond,"led.sgf"]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C
C     delta sblei at 1
      COEF = DFXDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 2
      COEF = DFXDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 3
      COEF = DFXDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C
C     delta sblei at 1
      COEF = DFYDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 2
      COEF = DFYDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 3
      COEF = DFYDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFY,ICOLY,NCY)

",filename]
```

```
/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C
C     delta sblei at 1
      COEF = DFZDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 2
      COEF = DFZDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 3
      COEF = DFZDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lleda2dec.sgf;* Lleda2dec.osf
!dele Lleda2dec.sgf;*
!@cgf Lleda2dec.sgf

<"decfile.in"

decfile["Lleda2dec.sgf"]

Lpr["
      SUBROUTINE LLEDB2(DFXDL2,DFYDL2,DFZDL2)
C     Left Leading Edge Derivative wrt xi2 for a B face used for
C     calculating psi2.
C
C     This routine calculates the contribution of the leading edge derivative
C     wrt xi2 for a B face for psi2 to the x, y, or z momentum eqs.
C
      REAL DFXDL2,DFYDL2,DFZDL2

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LLEDB2DEC.FOR'
      REAL DERI
      REAL COEF

      REAL SBLEI1,SBLEI2,SBLEI3,SBLEI4
      DATA SBLEI1,SBLEI2,SBLEI3,SBLEI4/4*0./

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
```

```
invar : {sblei1,sblei2,sblei3,sblei4}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_sblei1[Const] : 1 ; _sblei2[Const] : 1 ; _sblei3[Const] : 1
_sblei4[Const] : 1

/* Algorithm */

sle2case : {-3sblei1 + 2sblei2 + sblei3,\
            3sblei2 - 2sblei1 - sblei3,\
            ((sblei2 + sblei4) - (sblei1 + sblei3))/4}
sle2cond : {"MGD.EQ.0",\
           "MGD.EQ.1",\
           "MGD.EQ.2"}
casederiv[3,"SBLE derivative wrt xi2",sle2,sle2case,sle2cond,"led.sgf"]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C
C     delta sblei at 1
      COEF = DFXDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 2
      COEF = DFXDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 3
      COEF = DFXDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFX,ICOLX,NCX)
C
C     delta sblei at 4
      COEF = DFXDL2*SLE2D4
      CALL ADCOFS(COEF,JBLEI4+ISBLEO,COFX,ICOLX,NCX)

",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C
C     delta sblei at 1
      COEF = DFYDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 2
      COEF = DFYDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 3
      COEF = DFYDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFY,ICOLY,NCY)
C
C     delta sblei at 4
      COEF = DFYDL2*SLE2D4
      CALL ADCOFS(COEF,JBLEI4+ISBLEO,COFY,ICOLY,NCY)

",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C
C     delta sblei at 1
      COEF = DFZDL2*SLE2D1
      CALL ADCOFS(COEF,JBLEI1+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 2
      COEF = DFZDL2*SLE2D2
      CALL ADCOFS(COEF,JBLEI2+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 3
      COEF = DFZDL2*SLE2D3
      CALL ADCOFS(COEF,JBLEI3+ISBLEO,COFZ,ICOLZ,NCZ)
C
C     delta sblei at 4
      COEF = DFZDL2*SLE2D4
      CALL ADCOFS(COEF,JBLEI4+ISBLEO,COFZ,ICOLZ,NCZ)

",filename]

Lpr["
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lledb2dec.sgf;* Lledb2dec.osf
!dele Lledb2dec.sgf;*
!@cgf Lledb2dec.sgf

<"decfile.in"

decfile["Lledb2dec.sgf"]
```

## File RTHFC.IN

```
/* This session creates the subroutine which calculates r, and theta at
the face center.  The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy rthfc.sgf;* rthfc.osf
!dele rthfc.sgf;*
!@cgf rthfc.sgf

filename : "rthfc.sgf"

/* Load in the fort routine. */
```

```
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RRTHFC(RV,X1V)
C     Right R THeta calculation at the Face Center.
C
C     This routine calculates r and theta at the face center for a
C     cylindrical grid.
C     The variables in common are already initialized for the current face.
C
      REAL RV,X1V

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RRTHFCDEC.FOR'
C     Declare and set the moving grid variables to 0.
      REAL DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC
      DATA DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC/0.,0.,0.,0.,0.,0./

C     If not a cylindrical grid, then the r and theta are not used
C     except for consistancy and should be set to 0.
      IF(PLANER)THEN
        RV = 0.
        X1V = 0.
      END IF

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

x1av : x1ac + dx1a + d1dlea dslead
fort["X1 at a.",'x1av,"rthfc.sgf"]

x1bv : x1bc + dx1b + d1dleb dslebc
fort["X1 at b.",'x1bv,"rthfc.sgf"]

x1cv : x1cc + dx1c + d1dlec dslebc
fort["X1 at c.",'x1cv,"rthfc.sgf"]

x1dv : x1dc + dx1d + d1dled dslead
fort["X1 at d.",'x1dv,"rthfc.sgf"]

x2av : x2ac + d2dlea dslead
fort["X2 at a.",'x2av,"rthfc.sgf"]

x2bv : x2bc + d2dleb dslebc
fort["X2 at b.",'x2bv,"rthfc.sgf"]

x2cv : x2cc + d2dlec dslebc
fort["X2 at c.",'x2cv,"rthfc.sgf"]

x2dv : x2dc + d2dled dslead
fort["X2 at d.",'x2dv,"rthfc.sgf"]

x1 : (x1av + x1bv + x1cv + x1dv)/4
fort["Theta or X1 at face center.",'x1,"rthfc.sgf"]

r : (x2av + x2bv + x2cv + x2dv)/4
fort["R or X2 at face center.",'r,"rthfc.sgf"]

Lpr["
      RV = R
      X1V = X1
",filename]

Lpr["
C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rrthfcdec.sgf;* Rrthfcdec.osf
!dele Rrthfcdec.sgf;*
!@cgf Rrthfcdec.sgf

<"decfile.in"

decfile["Rrthfcdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lrthfcdec.sgf;* Lrthfcdec.osf
!dele Lrthfcdec.sgf;*
!@cgf Lrthfcdec.sgf

filename : "rthfc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LRTHFC(DFXDR,DFXDX1,
     1 DFYDR,DFYDX1,DFZDR,DFZDX1)
C     Left R THeta calculation at the Face Center.
C
C     This routine calculates the contribution of r and theta at the
C     face center for a cylindrical grid to the left hand sides of
C     the x, y, and z momentum eqs.
C     The variables in common are already initialized for the current face.
C
      REAL DFXDR,DFXDX1
      REAL DFYDR,DFYDX1
      REAL DFZDR,DFZDX1

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LRTHFCDEC.FOR'
C     Declare and set the moving grid variables to 0.
      REAL DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC
      DATA DX1A,DX1B,DX1C,DX1D,DSLEAD,DSLEBC/0.,0.,0.,0.,0.,0./

      REAL DERI
      REAL COEF

C     If not a cylindrical grid, then the r and theta are not used
```

```
C     except for consistancy.  Just return.
      IF(PLANER)RETURN

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {dx1a,dx1b,dx1c,dx1d,dslead,dslebc}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_dx1a[Const] : 1 ; _dx1b[Const] : 1 ; _dx1c[Const] : 1 ; _dx1d[Const] : 1
_dslead[Const] : 1 ; _dslebc[Const] : 1
_x1ac[Const] : 1 ; _x1bc[Const] : 1 ; _x1cc[Const] : 1 ; _x1dc[Const] : 1
_x2ac[Const] : 1 ; _x2bc[Const] : 1 ; _x2cc[Const] : 1 ; _x2dc[Const] : 1
_d1dlea[Const] : 1 ; _d1dleb[Const] : 1
_d1dlec[Const] : 1 ; _d1dled[Const] : 1
_d2dlea[Const] : 1 ; _d2dleb[Const] : 1
_d2dlec[Const] : 1 ; _d2dled[Const] : 1

/* Algorithm */

x1av : x1ac + dx1a + d1dlea dslead
deriv["X1 at a.",'x1av,"rthfc.sgf"]

x1bv : x1bc + dx1b + d1dleb dslebc
deriv["X1 at b.",'x1bv,"rthfc.sgf"]

x1cv : x1cc + dx1c + d1dlec dslebc
deriv["X1 at c.",'x1cv,"rthfc.sgf"]

x1dv : x1dc + dx1d + d1dled dslead
deriv["X1 at d.",'x1dv,"rthfc.sgf"]

x2av : x2ac + d2dlea dslead
deriv["X2 at a.",'x2av,"rthfc.sgf"]

x2bv : x2bc + d2dleb dslebc
deriv["X2 at b.",'x2bv,"rthfc.sgf"]

x2cv : x2cc + d2dlec dslebc
deriv["X2 at c.",'x2cv,"rthfc.sgf"]

x2dv : x2dc + d2dled dslead
deriv["X2 at d.",'x2dv,"rthfc.sgf"]

x1 : (x1av + x1bv + x1cv + x1dv)/4
deriv["Theta or X1 at face center.",'x1,"rthfc.sgf"]

r : (x2av + x2bv + x2cv + x2dv)/4
deriv["R or X2 at face center.",'r,"rthfc.sgf"]

/* Include in exprelist the dependence of r and theta on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[r,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[x1,dii]; \
]

Lpr["
```

```
C     The dependence on delta x1 at volume nodes.
      CALL LDX1VN(LA,DFXDR*RD1+DFXDX1*X1D1,
     1              DFYDR*RD1+DFYDX1*X1D1,
     1              DFZDR*RD1+DFZDX1*X1D1)

      CALL LDX1VN(LB,DFXDR*RD2+DFXDX1*X1D2,
     1              DFYDR*RD2+DFYDX1*X1D2,
     1              DFZDR*RD2+DFZDX1*X1D2)

      CALL LDX1VN(LC,DFXDR*RD3+DFXDX1*X1D3,
     1              DFYDR*RD3+DFYDX1*X1D3,
     1              DFZDR*RD3+DFZDX1*X1D3)

      CALL LDX1VN(LD,DFXDR*RD4+DFXDX1*X1D4,
     1              DFYDR*RD4+DFYDX1*X1D4,
     1              DFZDR*RD4+DFZDX1*X1D4)

C     The dependence on delta sble for a and d.
      CALL LSBLEN(JSLEAD,NJJ,DFXDR*RD5+DFXDX1*X1D5,
     1                       DFYDR*RD5+DFYDX1*X1D5,
     1                       DFZDR*RD5+DFZDX1*X1D5)

C     The dependence on delta sble for b and c.
      CALL LSBLEN(JSLEBC,NJJ,DFXDR*RD6+DFXDX1*X1D6,
     1                       DFYDR*RD6+DFYDX1*X1D6,
     1                       DFZDR*RD6+DFZDX1*X1D6)

",filename]

Lpr["

      RETURN
      END
",filename]

/* Write the declaration file. */

<"decfile.in"

decfile["Lrthfcdec.sgf"]
```

## File MAC.IN

```
/* This session creates the subroutine which calculate the contribution
of an A face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum for compressible
flow.  The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mac.sgf;* mac.osf
!dele mac.sgf;*
!@cgf mac.sgf

filename : "mac.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMAC(IAF,FM)
C     Right Momentum equation for an A-face for Compressible flow.
C
C     This routine calculates the contribution of the
C     A face IAF to the residuals
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IAF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMACDEC.FOR'

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ

C     Declare the radius and x1 dummy value.
      REAL R,DUM

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMA(IAF)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
fort["X component of pressure",'px,"mac.sgf"]

py : p ay
fort["Y component of pressure",'py,"mac.sgf"]

pz : p az
fort["Z component of pressure",'pz,"mac.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to residual.",filename]
Lpr["C       variables and return.",filename]

fx : px fm
fort["Face contribution to X-momentum eq.",'fx,"mac.sgf"]

fy : py fm
fort["Face contribution to Y-momentum eq.",'fy,"mac.sgf"]

fz : pz fm
```

```
fort["Face contribution to Z-momentum eq.",'fz,"mac.sgf"]

Lpr["
        RESX = RESX - FX
        RESY = RESY - FY
        RESZ = RESZ - FZ
        RETURN
      END IF

C     Non-wall A Faces.

C     Get the mass flux and the velocities.
      CALL RMASA(MAS)
      IF(PLANER)THEN
        CALL RVELP(1,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(1,RWX,RWY,RWZ)
      END IF

C     Get the radius and x1 dummy value.
      CALL RRTHFC(R,DUM)

",filename]

/* Compressible flow */
rws : rwx^2 + rwy^2 + rwz^2
fort["(rho W)^2",'rws,"mac.sgf"]

or2 : (omg r)^2 / 2
fort["(omega r)^2/2",'or2,"mac.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
fort["density",'rho,"mac.sgf"]

fx : ( (mas rwx)/rho + px ) fm
fort["Face contribution to X-momentum eq.",'fx,"mac.sgf"]

fy : ( (mas rwy)/rho + py ) fm
fort["Face contribution to Y-momentum eq.",'fy,"mac.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
fort["Face contribution to Z-momentum eq.",'fz,"mac.sgf"]

Lpr["
      RESX = RESX - FX
      RESY = RESY - FY
      RESZ = RESZ - FZ

C     Save some of the calculated quantities.
      IF(ISAVE.EQ.1)CALL SAVEA(IAF,RWX,RWY,RWZ,RHO)

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmacdec.sgf;* Rmacdec.osf
!dele Rmacdec.sgf;*
!@cgf Rmacdec.sgf
```

```
<"decfile.in"

decfile["Rmacdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mac.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMAC(IAF,FM)
C     Left Momentum equation for an A-face.
C
C     This routine calculates the contribution of the
C     A face IAF to the left hand sides
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IAF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMACDEC.FOR'
      REAL DERI
C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ

C     Declare the radius and x1 dummy value.
      REAL R,DUM

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMA(IAF)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,ax,ay,az,mas,rwx,rwy,rwz,r}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_ax[Const] : 1 ; _ay[Const] : 1 ; _az[Const] : 1
_mas[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_gam[Const] : 1 ; _h[Const] : 1 ; _fm[Const] : 1 ; _omg[Const] : 1
_roc[Const] : 1
_r[Const] : 1

/* Algorithm */

Lpr["
```

```
C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
deriv["X component of pressure",'px,"mac.sgf"]

py : p ay
deriv["Y component of pressure",'py,"mac.sgf"]

pz : p az
deriv["Z component of pressure",'pz,"mac.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to matrix.",filename]
Lpr["C       and return.",filename]

fx : px fm
deriv["Face contribution to X-momentum eq.",'fx,"mac.sgf"]

fy : py fm
deriv["Face contribution to Y-momentum eq.",'fy,"mac.sgf"]

fz : pz fm
deriv["Face contribution to Z-momentum eq.",'fz,"mac.sgf"]

Lpr["
C
C       Put Jacobians in correct location in matrix.
C       X - Momentum Equation
C       NCX must be set already.
C
C       pa
        CALL ADCOFS(FXD1,IPAO+IP,COFX,ICOLX,NCX)
C
C       Y - Momentum Equation
C       NCY must be set already.
C
C       pa
        CALL ADCOFS(FYD1,IPAO+IP,COFY,ICOLY,NCY)
C
C       Z - Momentum Equation
C       NCZ must be set already.
C
C       pa
        CALL ADCOFS(FZD1,IPAO+IP,COFZ,ICOLZ,NCZ)
C

C       The dependence on the area normals.
        IF(PLANER)THEN
          CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        ELSE
          CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        END IF
```

```
         RETURN
       END IF

C      Non-wall A Faces.

C      Get the mass flux and the velocities.
       CALL RMASA(MAS)
       IF(PLANER)THEN
         CALL RVELP(1,RWX,RWY,RWZ)
       ELSE
         CALL RVELC(1,RWX,RWY,RWZ)
       END IF

C      Get the radius and x1 dummy value.
       CALL RRTHFC(R,DUM)

",filename]

/* Compressible flow */
rws : rwx^2 + rwy^2 + rwz^2
deriv["(rho W)^2",'rws,"mac.sgf"]

or2 : (omg r)^2 / 2
deriv["(omega r)^2/2",'or2,"mac.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
deriv["density",'rho,"mac.sgf"]

fx : ( (mas rwx)/rho + px ) fm
deriv["Face contribution to X-momentum eq.",'fx,"mac.sgf"]

fy : ( (mas rwy)/rho + py ) fm
deriv["Face contribution to Y-momentum eq.",'fy,"mac.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
deriv["Face contribution to Z-momentum eq.",'fz,"mac.sgf"]

/* Include in exprelist the dependence of fx, fy, and fz on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[fx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fz,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      X - Momentum Equation
C      NCX must be set already.
C
C      pa
       CALL ADCOFS(FXD1,IPAO+IP,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
```

```
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C     pa
      CALL ADCOFS(FYD1,IPAO+IP,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C     pa
      CALL ADCOFS(FZD1,IPAO+IP,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
C     The dependence on the area normals.
      IF(PLANER)THEN
        CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1            FZD2,FZD3,FZD4)
      ELSE
        CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1            FZD2,FZD3,FZD4)
      END IF

",filename]

Lpr["
C     The dependence on the mass flux.
      CALL LMASA(FXD5,FYD5,FZD5)

",filename]

Lpr["
C     The dependence on the velocities.
      IF(PLANER)THEN
        CALL LVELP(1,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
      ELSE
        CALL LVELC(1,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
      END IF

C     The dependence on the radius.
      CALL LRTHFC(FXD9,0.,FYD9,0.,FZD9,0.)

",filename]

Lpr["

      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmacdec.sgf;* Lmacdec.osf
!dele Lmacdec.sgf;*
!@cgf Lmacdec.sgf

<"decfile.in"
```

```
decfile["Lmacdec.sgf"]
```

## File MBC.IN

```
/* This session creates the subroutine which calculate the contribution
of an B face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum for compressible
flow.  The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy mbc.sgf;* mbc.osf
!dele mbc.sgf;*
!@cgf mbc.sgf

filename : "mbc.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMBC(IBF,FM)
C     Right Momentum equation for a B-face for Compressible flow.
C
C     This routine calculates the contribution of the
C     B face IBF to the residuals
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IBF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMBCDEC.FOR'

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ

C     Declare the radius and x1 dummy value.
      REAL R,DUM

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMB(IBF)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
fort["X component of pressure",'px,"mbc.sgf"]

py : p ay
fort["Y component of pressure",'py,"mbc.sgf"]

pz : p az
fort["Z component of pressure",'pz,"mbc.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to residual.",filename]
Lpr["C       variables and return.",filename]

fx : px fm
fort["Face contribution to X-momentum eq.",'fx,"mbc.sgf"]

fy : py fm
fort["Face contribution to Y-momentum eq.",'fy,"mbc.sgf"]

fz : pz fm
fort["Face contribution to Z-momentum eq.",'fz,"mbc.sgf"]

Lpr["
        RESX = RESX - FX
        RESY = RESY - FY
        RESZ = RESZ - FZ
        RETURN
      END IF

C     Non-wall A Faces.

C     Get the mass flux and the velocities.
      CALL RMASB(MAS)
      IF(PLANER)THEN
        CALL RVELP(2,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(2,RWX,RWY,RWZ)
      END IF

C     Get the radius and x1 dummy value.
      CALL RRTHFC(R,DUM)

",filename]

/* Compressible flow */
rws : rwx^2 + rwy^2 + rwz^2
fort["(rho W)^2",'rws,"mbc.sgf"]

or2 : (omg r)^2 / 2
fort["(omega r)^2/2",'or2,"mbc.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
fort["density",'rho,"mbc.sgf"]

fx : ( (mas rwx)/rho + px ) fm
fort["Face contribution to X-momentum eq.",'fx,"mbc.sgf"]
```

```
fy : ( (mas rwy)/rho + py ) fm
fort["Face contribution to Y-momentum eq.",'fy,"mbc.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
fort["Face contribution to Z-momentum eq.",'fz,"mbc.sgf"]

Lpr["
      RESX = RESX - FX
      RESY = RESY - FY
      RESZ = RESZ - FZ

C     Save some of the calculated quantities.
      IF(ISAVE.EQ.1)CALL SAVEB(IBF,RWX,RWY,RWZ,RHO)

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmbcdec.sgf;* Rmbcdec.osf
!dele Rmbcdec.sgf;*
!@cgf Rmbcdec.sgf

<"decfile.in"

decfile["Rmbcdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "mbc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMBC(IBF,FM)
C     Left Momentum equation for a B-face.
C
C     This routine calculates the contribution of the
C     B face IBF to the left hand sides
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER IBF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMBCDEC.FOR'
      REAL DERI
C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ
```

```
C     Declare the radius and x1 dummy value.
      REAL R,DUM

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMB(IBF)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,ax,ay,az,mas,rwx,rwy,rwz,r}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_ax[Const] : 1 ; _ay[Const] : 1 ; _az[Const] : 1
_mas[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_gam[Const] : 1 ; _h[Const] : 1 ; _fm[Const] : 1 ; _omg[Const] : 1
_roc[Const] : 1
_r[Const] : 1

/* Algorithm */

Lpr["

C     Get the area normals.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
      ELSE
        CALL RANC(AX,AY,AZ)
      END IF

",filename]

Lpr["C     Pressure terms.",filename]

px : p ax
deriv["X component of pressure",'px,"mbc.sgf"]

py : p ay
deriv["Y component of pressure",'py,"mbc.sgf"]

pz : p az
deriv["Z component of pressure",'pz,"mbc.sgf"]

Lpr["C     Check if wall.",filename]
Lpr["      IF(IFTYP.LT.0)THEN",filename]
Lpr["C       Wall so calculate face contribution, add to matrix.",filename]
Lpr["C       and return.",filename]

fx : px fm
deriv["Face contribution to X-momentum eq.",'fx,"mbc.sgf"]

fy : py fm
deriv["Face contribution to Y-momentum eq.",'fy,"mbc.sgf"]

fz : pz fm
deriv["Face contribution to Z-momentum eq.",'fz,"mbc.sgf"]

Lpr["
C
```

```
C       Put Jacobians in correct location in matrix.
C       X - Momentum Equation
C       NCX must be set already.
C
C       pb
        CALL ADCOFS(FXD1,IPBO+IP,COFX,ICOLX,NCX)
C
C       Y - Momentum Equation
C       NCY must be set already.
C
C       pb
        CALL ADCOFS(FYD1,IPBO+IP,COFY,ICOLY,NCY)
C
C       Z - Momentum Equation
C       NCZ must be set already.
C
C       pb
        CALL ADCOFS(FZD1,IPBO+IP,COFZ,ICOLZ,NCZ)
C

C       The dependence on the area normals.
        IF(PLANER)THEN
          CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        ELSE
          CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1              FZD2,FZD3,FZD4)
        END IF

        RETURN
      END IF

C     Non-wall B Faces.

C     Get the mass flux and the velocities.
      CALL RMASB(MAS)
      IF(PLANER)THEN
        CALL RVELP(2,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(2,RWX,RWY,RWZ)
      END IF

C     Get the radius and x1 dummy value.
      CALL RRTHFC(R,DUM)

",filename]

/* Compressible flow */
rws : rwx^2 + rwy^2 + rwz^2
deriv["(rho W)^2",'rws,"mbc.sgf"]

or2 : (omg r)^2 / 2
deriv["(omega r)^2/2",'or2,"mbc.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
deriv["density",'rho,"mbc.sgf"]

fx : ( (mas rwx)/rho + px ) fm
deriv["Face contribution to X-momentum eq.",'fx,"mbc.sgf"]

fy : ( (mas rwy)/rho + py ) fm
deriv["Face contribution to Y-momentum eq.",'fy,"mbc.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
deriv["Face contribution to Z-momentum eq.",'fz,"mbc.sgf"]
```

```
/* Include in exprelist the dependence of fx, fy, and fz on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[fx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fz,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C     pb
      CALL ADCOFS(FXD1,IPBO+IP,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C     pb
      CALL ADCOFS(FYD1,IPBO+IP,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C     pb
      CALL ADCOFS(FZD1,IPBO+IP,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
C     The dependence on the area normals.
      IF(PLANER)THEN
        CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1            FZD2,FZD3,FZD4)
      ELSE
        CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1            FZD2,FZD3,FZD4)
      END IF

",filename]

Lpr["
C     The dependence on the mass flux.
      CALL LMASB(FXD5,FYD5,FZD5)

",filename]
```

```
Lpr["
C     The dependence on the velocities.
      IF(PLANER)THEN
        CALL LVELP(2,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
      ELSE
        CALL LVELC(2,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
      END IF

C     The dependence on the radius.
      CALL LRTHFC(FXD9,0.,FYD9,0.,FZD9,0.)

",filename]

Lpr["

      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lmbcdec.sgf;* Lmbcdec.osf
!dele Lmbcdec.sgf;*
!@cgf Lmbcdec.sgf

<"decfile.in"

decfile["Lmbcdec.sgf"]
```

## File MSC.IN

```
/* This session creates the subroutine which calculate the contribution
of an S face to the residual right hand side for each
of the equations:  x-momentum, y-momentum, and z momentum for compressible
flow.  The left hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy msc.sgf;* msc.osf
!dele msc.sgf;*
!@cgf msc.sgf

filename : "msc.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RMSC(ISF,FM)
C     Right Momentum equation for an S-face for compressible flow.
C
C     This routine calculates the contribution of the
C     S face ISF to the residuals
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER ISF
      REAL FM
      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RMSCDEC.FOR'

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ
C     Declare the volume variable.
      REAL VOL
C     Declare the r and theta at the face center.
      REAL R,X1

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the mass flux and the velocities.
      CALL RMASS(MAS)
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

Lpr["

C     Get the area normals and volume.  The volume is set to 0 if omega is 0.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
        IF(OMG.EQ.0.)THEN
          VOL = 0.
        ELSE
          CALL RVOLP(VOL)
        END IF
      ELSE
        CALL RANC(AX,AY,AZ)
        IF(OMG.EQ.0.)THEN
          VOL = 0.
        ELSE
          CALL RVOLC(VOL)
        END IF
      END IF

C     Get the r and theta.  If planer, the routine returns 0 for both.
      CALL RRTHFC(R,X1)

",filename]

/* Compressible flow */
rws : rwx^2 + rwy^2 + rwz^2
fort["(rho W)^2",'rws,"msc.sgf"]

or2 : (omg r)^2 / 2
```

```
fort["(omega r)^2/2",'or2,"msc.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
fort["density",'rho,"msc.sgf"]

ws : rws/(rho)^2
fort["velocity squared",'ws,"msc.sgf"]

ms : ws/((gam-1)(h - ws/2 + or2))
fort["Mach number squared.",'ms,"msc.sgf"]

vscase : {ms(1 - mcs/ms)/(1 + (gam-1)ms),\
          m1s(1 - mcs/m1s)/(1 + (gam-1)m1s),\
           0}
vscond : {"MS.GE.M1S .AND. MS.GT.MCS",\
          "MS.LT.M1S .AND. M1S.GT.MCS",\
          ".TRUE."}
casefort[3,"The artificial viscosity mu.",vs,vscase,vscond,"msc.sgf"]

pb : p + viscm vs(p - p1) - viscm vsc(p1 - p0)
fort["upwind pressure",'pb,"msc.sgf"]

Lpr["C     Pressure terms.",filename]

px : pb ax
fort["X component of pressure",'px,"msc.sgf"]

py : pb ay
fort["Y component of pressure",'py,"msc.sgf"]

pz : pb az
fort["Z component of pressure",'pz,"msc.sgf"]

Lpr["C     Source terms.",filename]

x : r Sin[x1]
fort["x at face",'x,"msc.sgf"]

y : r Cos[x1]
fort["y at face",'y,"msc.sgf"]

sx : vol (-rho omg^2 x - 2 omg rwy)/2
fort["X component of rotating source term",'sx,"msc.sgf"]

sy : vol (-rho omg^2 y + 2 omg rwx)/2
fort["Y component of rotating source term",'sy,"msc.sgf"]

Lpr["C     Face contibutions to momentum eqs.",filename]

fx : ( (mas rwx)/rho + px ) fm + sx
fort["Face contribution to X-momentum eq.",'fx,"msc.sgf"]

fy : ( (mas rwy)/rho + py ) fm + sy
fort["Face contribution to Y-momentum eq.",'fy,"msc.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
fort["Face contribution to Z-momentum eq.",'fz,"msc.sgf"]

Lpr["
      RESX = RESX - FX
      RESY = RESY - FY
      RESZ = RESZ - FZ

C     Save some of the calculated quantities.
      IF(ISAVE.EQ.1)CALL SAVES(ISF,RWX,RWY,RWZ,RHO)
```

```
C     Save the Mach number.
      CALL SAVEM(ISF,MS)

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rmscdec.sgf;* Rmscdec.osf
!dele Rmscdec.sgf;*
!@cgf Rmscdec.sgf

<"decfile.in"

decfile["Rmscdec.sgf"]

/* Copy and create the LMSC declaratin file. */
!copy Lmscdec.sgf;* Lmscdec.osf
!dele Lmscdec.sgf;*
!@cgf Lmscdec.sgf

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "msc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LMSC(ISF,FM)
C     Left Momentum equation for an S-face for compressible flow.
C
C     This routine calculates the contribution of the
C     S face ISF to the left hand sides
C     for the x, y, and z momentum equations.  FM indicates
C     whether the face normal points out of (+) or into (-) the
C     control volume.
C
      INTEGER ISF
      REAL FM

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LMSCDEC.FOR'
      REAL DERI

C     Declare the mass flux and the velocities.
      REAL MAS,RWX,RWY,RWZ
C     Declare the area normals.
      REAL AX,AY,AZ
C     Declare the volume variable.
      REAL VOL
C     Declare the r and theta at the face center.
      REAL R,X1
C     Declare the IP1 temporary.
      INTEGER IP1TMP
```

```
C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,ax,ay,az,mas,rwx,rwy,rwz,r,x1,vol,p1,m1s,p0}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_ax[Const] : 1 ; _ay[Const] : 1 ; _az[Const] : 1
_mas[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_gam[Const] : 1 ; _omg[Const] : 1 ; _fm[Const] : 1
_roc[Const] : 1
_vol[Const] : 1
_r[Const] : 1 ; _x1[Const] : 1
_vsc[Const] : 1 ; _mcs[Const] : 1 ;
_p1[Const] : 1 ; _m1s[Const] : 1 ; _p0[Const] : 1
_h[Const] : 1
_viscm[Const] : 1

/* Algorithm */

Lpr["

C     Get the mass flux and the velocities.
      CALL RMASS(MAS)
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

Lpr["

C     Get the area normals and volume.  The volume is set to 0 if omega is 0.
      IF(PLANER)THEN
        CALL RANP(AX,AY,AZ)
        IF(OMG.EQ.0.)THEN
          VOL = 0.
        ELSE
          CALL RVOLP(VOL)
        END IF
      ELSE
        CALL RANC(AX,AY,AZ)
        IF(OMG.EQ.0.)THEN
          VOL = 0.
        ELSE
          CALL RVOLC(VOL)
        END IF
      END IF

C     Get the r and theta.  If planer, the routine returns 0 for both.
      CALL RRTHFC(R,X1)

",filename]
```

```
/* Compressible flow */
rws : rwx^2 + rwy^2 + rwz^2
deriv["(rho W)^2",'rws,"msc.sgf"]

or2 : (omg r)^2 / 2
deriv["(omega r)^2/2",'or2,"msc.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
deriv["density",'rho,"msc.sgf"]

ws : rws/(rho)^2
deriv["velocity squared",'ws,"msc.sgf"]

ms : ws/((gam-1)(h - ws/2 + or2))
deriv["Mach number squared.",'ms,"msc.sgf"]

vscase : {ms(1 - mcs/ms)/(1 + (gam-1)ms),\
          m1s(1 - mcs/m1s)/(1 + (gam-1)m1s),\
          0}
vscond : {"MS.GE.M1S .AND. MS.GT.MCS",\
          "MS.LT.M1S .AND. M1S.GT.MCS",\
          ".TRUE."}
casederiv[3,"The artificial viscosity mu.",vs,vscase,vscond,"msc.sgf"]

pb : p + viscm vs(p - p1) - viscm vsc(p1 - p0)
deriv["upwind pressure",'pb,"msc.sgf"]

Lpr["C     Pressure terms.",filename]

px : pb ax
deriv["X component of pressure",'px,"msc.sgf"]

py : pb ay
deriv["Y component of pressure",'py,"msc.sgf"]

pz : pb az
deriv["Z component of pressure",'pz,"msc.sgf"]

Lpr["C     Source terms.",filename]

x : r Sin[x1]
deriv["x at face",'x,"msc.sgf"]

y : r Cos[x1]
deriv["y at face",'y,"msc.sgf"]

sx : vol (-rho omg^2 x - 2 omg rwy)/2
deriv["X component of rotating source term",'sx,"msc.sgf"]

sy : vol (-rho omg^2 y + 2 omg rwx)/2
deriv["Y component of rotating source term",'sy,"msc.sgf"]

Lpr["C     Face contibutions to momentum eqs.",filename]

fx : ( (mas rwx)/rho + px ) fm + sx
deriv["Face contribution to X-momentum eq.",'fx,"msc.sgf"]

fy : ( (mas rwy)/rho + py ) fm + sy
deriv["Face contribution to Y-momentum eq.",'fy,"msc.sgf"]

fz : ( (mas rwz)/rho + pz ) fm
deriv["Face contribution to Z-momentum eq.",'fz,"msc.sgf"]

/* Include in exprelist the dependence of fx, fy, and fz on all the
```

```
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[fx,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fy,dii]; \
  nel : nel + 1 ;\
  exprlist[nel] : Make[fz,dii] \
]

/* X - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      X - Momentum Equation
C      NCX must be set already.
C
C      ps
       CALL ADCOFS(FXD1,IPSO+IP,COFX,ICOLX,NCX)
C      p1
       CALL ADCOFS(FXD12,IPSO+IP1,COFX,ICOLX,NCX)
C      p0
       CALL ADCOFS(FXD14,IPSO+IPO,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      Y - Momentum Equation
C      NCY must be set already.
C
C      ps
       CALL ADCOFS(FYD1,IPSO+IP,COFY,ICOLY,NCY)
C      p1
       CALL ADCOFS(FYD12,IPSO+IP1,COFY,ICOLY,NCY)
C      p0
       CALL ADCOFS(FYD14,IPSO+IPO,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C      Put Jacobians in correct location in matrix.
C      Z - Momentum Equation
C      NCZ must be set already.
C
C      ps
       CALL ADCOFS(FZD1,IPSO+IP,COFZ,ICOLZ,NCZ)
C      p1
       CALL ADCOFS(FZD12,IPSO+IP1,COFZ,ICOLZ,NCZ)
C      p0
       CALL ADCOFS(FZD14,IPSO+IPO,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
C      The dependence on the area normals and volume.
       IF(PLANER)THEN
         CALL LANP(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1             FZD2,FZD3,FZD4)
         IF(OMG.NE.O.)CALL LVOLP(FXD11,FYD11,FZD11)
       ELSE
```

```
         CALL LANC(FXD2,FXD3,FXD4, FYD2,FYD3,FYD4,
     1             FZD2,FZD3,FZD4)
         IF(OMG.NE.O.)CALL LVOLC(FXD11,FYD11,FZD11)
       END IF

",filename]

Lpr["
C      The dependence on the mass flux.
       CALL LMASS(FXD5,FYD5,FZD5)

",filename]

Lpr["
C      The dependence on the velocities.
       IF(PLANER)THEN
         CALL LVELP(3,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
       ELSE
         CALL LVELC(3,FXD6,FXD7,FXD8,FYD6,FYD7,FYD8,FZD6,FZD7,FZD8)
       END IF
",filename]

Lpr["
C      The dependence on r and theta.
       CALL LRTHFC(FXD9,FXD10,FYD9,FYD10,FZD9,FZD10)

C      The dependence on m1s, the Mach number_1 squared.  IP1 is the
C      index for the upstream face.  Do only if used.  IP1 is in common
C      and LM1SS calls IMS to initialize this face.  This is why this routine
C      is called last and why IP1TMP is used as a tempory variable so
C      there is no conflict between what is passed through the argument
C      list and what is in common.
       IF(MS.LT.M1S .AND. M1S.GT.MCS)THEN
         IP1TMP = IP1
         CALL LM1SS(IP1TMP,FXD13,FYD13,FZD13)
       END IF

",filename]

Lpr["


       RETURN
       END
",filename]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["Lmscdec.sgf"]
```

## File M1S.IN

```
/* This session creates the subroutine which calculate the contribution
of an S face to the Mach number squared left hand side. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy m1s.sgf;* m1s.osf
!dele m1s.sgf;*
!@cgf m1s.sgf

/* Copy and create the LM1SS declaratin file. */
```

```
!copy Lm1ssdec.sgf;* Lm1ssdec.osf
!dele Lm1ssdec.sgf;*
!@cgf Lm1ssdec.sgf

filename : "m1s.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"

Lpr["
      SUBROUTINE LM1SS(ISF,FXDM1S,FYDM1S,FZDM1S)
C     Left M1 Squared at an S face.
C
C     This routine calculates the contribution of the
C     S face ISF to the Mach number squared.  It is used exclusivly by
C     LMCS for finding the dependence of M1S.
C
      INTEGER ISF
      REAL FXDM1S,FYDM1S,FZDM1S

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LM1SSDEC.FOR'
      REAL DERI

C     Declare the velocities.
      REAL RWX,RWY,RWZ
C     Declare the r and theta at the face center.
      REAL R,X1

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,rwx,rwy,rwz,r}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_gam[Const] : 1 ; _omg[Const] : 1
_r[Const] : 1 ; _x1[Const] : 1
_h[Const] : 1

/* Algorithm */

Lpr["

C     Get the velocities.
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

C     Get the r and theta.  If planer, the routine returns 0 for both.
      CALL RRTHFC(R,X1)
```

```
",filename]

/* Compressible flow */
rws : rwx^2 + rwy^2 + rwz^2
deriv["(rho W)^2",'rws,"m1s.sgf"]

or2 : (omg r)^2 / 2
deriv["(omega r)^2/2",'or2,"m1s.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
deriv["density",'rho,"m1s.sgf"]

ws : rws/(rho)^2
deriv["velocity squared",'ws,"m1s.sgf"]

ms : ws/((gam-1)(h - ws/2 + or2))
deriv["Mach number squared.",'ms,"m1s.sgf"]

/* Include in exprelist the dependence of ms on all the
dependent variables. */
Do[i,niv,\
  dii : Make[d,i];\
  nel : nel + 1 ;\
  exprlist[nel] : Make[ms,dii]; \
]

/* X - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation
C     NCX must be set already.
C
C     ps
      CALL ADCOFS(FXDM1S*MSD1,IPSO+IP,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation
C     NCY must be set already.
C
C     ps
      CALL ADCOFS(FYDM1S*MSD1,IPSO+IP,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Z - Momentum Equation
C     NCZ must be set already.
C
C     ps
      CALL ADCOFS(FZDM1S*MSD1,IPSO+IP,COFZ,ICOLZ,NCZ)
",filename]

Lpr["
C     The dependence on the velocities.
      IF(PLANER)THEN
```

```
         CALL LVELP(3,FXDM1S*MSD2,FXDM1S*MSD3,FXDM1S*MSD4,
     1              FYDM1S*MSD2,FYDM1S*MSD3,FYDM1S*MSD4,
     1              FZDM1S*MSD2,FZDM1S*MSD3,FZDM1S*MSD4)
       ELSE
         CALL LVELC(3,FXDM1S*MSD2,FXDM1S*MSD3,FXDM1S*MSD4,
     1              FYDM1S*MSD2,FYDM1S*MSD3,FYDM1S*MSD4,
     1              FZDM1S*MSD2,FZDM1S*MSD3,FZDM1S*MSD4)
       END IF
",filename]

Lpr["
C     The dependence on r and theta.
      CALL LRTHFC(FXDM1S*MSD5,0.,FYDM1S*MSD5,0.,FZDM1S*MSD5,0.)

",filename]

Lpr["


      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

<"decfile.in"

decfile["Lm1ssdec.sgf"]
```

## File AUXPC.IN

```
/* This session creates the subroutine which calculates the residuals to
the auxilliary pressure equations for compressible flow.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy auxpc.sgf;* auxpc.osf
!dele auxpc.sgf;*
!@cgf auxpc.sgf

filename : "auxpc.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RAUXPC(M)
C     Right AUXilliary Pressure equations for volume M for compressible flow.
C
C     This routine calculates the residual for the A and B auxilliary pressure
C     equations for volume M.  The A equation is in RESX and the B equation
C     is in RESY.
C
      INTEGER M

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RAUXPCDEC.FOR'

C     Declare the velocities.
      REAL RWX1,RWY1,RWZ1,RWX2,RWY2,RWZ2
C     Declare the psi1 and psi2 curvature terms.
      REAL SI1C,SI2C

C     Declare the r and theta at the face center.
      REAL R1,R2,DUM

C     Initialize the variables for this equation.
      CALL IAUXP(M)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the velocities for S face 1.
      CALL IMS(ISF1)
      IF(PLANER)THEN
        CALL RVELP(3,RWX1,RWY1,RWZ1)
      ELSE
        CALL RVELC(3,RWX1,RWY1,RWZ1)
      END IF

C     Get r for face 1.  Theta is not needed.
      CALL RRTHFC(R1,DUM)

C     Get the velocities for S face 2.
      CALL IMS(ISF2)
      IF(PLANER)THEN
        CALL RVELP(3,RWX2,RWY2,RWZ2)
      ELSE
        CALL RVELC(3,RWX2,RWY2,RWZ2)
      END IF

C     Get r for face 2.  Theta is not needed.
      CALL RRTHFC(R2,DUM)

C     Get the psi1 curvature term.
      CALL RSI1CT(SI1C)

C     Get the psi2 curvature term.
      CALL RSI2CT(SI2C)

",filename]

ws1 : rwx1^2 + rwy1^2 + rwz1^2
fort["(rho W)^2 at s face 1.",'ws1,"auxpc.sgf"]

ws2 : rwx2^2 + rwy2^2 + rwz2^2
fort["(rho W)^2 at s face 2.",'ws2,"auxpc.sgf"]

/* Compressible flow */
or1 : (omg r1)^2 / 2
fort["(omega r1)^2/2",'or1,"auxpc.sgf"]

ro1 : (gam ps1  + ((gam ps1)^2 + 2(gam-1)^2 (h1 + or1) ws1)^(1/2))/  \
      (2(gam-1)(h1 + or1))
fort["rho at 1",'ro1,"auxpc.sgf"]

or2 : (omg r2)^2 / 2
fort["(omega r2)^2/2",'or2,"auxpc.sgf"]

ro2 : (gam ps2  + ((gam ps2)^2 + 2(gam-1)^2 (h2 + or2) ws2)^(1/2))/  \
      (2(gam-1)(h2 + or2))
fort["rho at 2",'ro2,"auxpc.sgf"]

ms1 : ws1/( ro1^2 (gam-1)(h1 - ws1/(2 ro1^2) + or1) )
fort["Mach number at 1",'ms1,"auxpc.sgf"]

ms2 : ws2/( ro2^2 (gam-1)(h2 - ws2/(2 ro2^2) + or2) )
fort["Mach number at 2",'ms2,"auxpc.sgf"]

msb : (ms1 + ms2)/2
fort["Average Mach number.",'msb,"auxpc.sgf"]

pb : (ps1 + ps2)/2
fort["Average Pressure.",'pb,"auxpc.sgf"]

pcacase : {-4 ka gam pb msb (1-msb) si1c,\
           0}
pcacond : {"MSB.LT.1.",\
          ".TRUE."}
casefort[2,"2 P_c for the A equation.",pca,pcacase,pcacond,"auxpc.sgf"]

pcbcase : {-4 kb gam pb msb (1-msb) si2c,\
           0}
pcbcond : {"MSB.LT.1.",\
          ".TRUE."}
casefort[2,"2 P_c for the B equation.",pcb,pcbcase,pcbcond,"auxpc.sgf"]

aer : ps1 + ps2 - pa1 - pa2 - pca
fort["residual for A equation. (A error)",'aer,"auxpc.sgf"]

ber : ps1 + ps2 - pb1 - pb2 - pcb
fort["residual for B equation. (B error)",'ber,"auxpc.sgf"]

Lpr["
      RESX = -AER
      RESY = -BER

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rauxpcdec.sgf;* Rauxpcdec.osf
!dele Rauxpcdec.sgf;*
!@cgf Rauxpcdec.sgf

<"decfile.in"

decfile["Rauxpcdec.sgf"]

/* Create the declaration file for the variables in exprlist which
will be output at the end of this routine.  This is done here rather
than at the end of the routine because of an error that was occuring.*/

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lauxpcdec.sgf;* Lauxpcdec.osf
!dele Lauxpcdec.sgf;*
!@cgf Lauxpcdec.sgf
```

```
/* Reinitialize everything to ensure enough room. */
Set[]

filename : "auxpc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LAUXPC(M)
C     Left AUXilliary Pressure equations for volume M for compressible flow.
C
C     This routine calculates the left hand side coefficients of the Jaqcobian
C     matrix for the A and B auxilliary pressure
C     equations for volume M.  The A equation is in the COFX arrays and
C     the B equation is in the COFY arrays.
C
      INTEGER M

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LAUXPCDEC.FOR'
      REAL DERI

C     Declare the velocities.
      REAL RWX1,RWY1,RWZ1,RWX2,RWY2,RWZ2

C     Declare the psi1 and psi2 curvature terms.
      REAL SI1C,SI2C

C     Declare the r and theta at the face center.
      REAL R1,R2,DUM

C     Initialize the variables for this equation.
      CALL IAUXP(M)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {ps1,ps2,pa1,pa2,pb1,pb2,rwx1,rwy1,rwz1,rwx2,rwy2,rwz2,si1c,si2c,r1,r2}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_ps1[Const] : 1 ; _ps2[Const] : 1 ; _pa1[Const] : 1 ; _pa2[Const] : 1
_pb1[Const] : 1 ; _pb2[Const] : 1
_rwx1[Const] : 1 ; _rwy1[Const] : 1 ; _rwz1[Const] : 1
_rwx2[Const] : 1 ; _rwy2[Const] : 1 ; _rwz2[Const] : 1
_si1c[Const] : 1 ; _si2c[Const] : 1
_roc[Const] : 1 ; _ka[Const] : 1 ; _kb[Const] : 1
_h1[Const] : 1 ; _h2[Const] : 1
_gam[Const] : 1
_omg[Const] : 1
_r1[Const] : 1 ; _r2[Const] : 1

/* Algorithm */

Lpr["

C     Get the velocities for S face 1.
      CALL IMS(ISF1)
      IF(PLANER)THEN
        CALL RVELP(3,RWX1,RWY1,RWZ1)
      ELSE
        CALL RVELC(3,RWX1,RWY1,RWZ1)
      END IF

C     Get r for face 1.  Theta is not needed.
      CALL RRTHFC(R1,DUM)

C     Get the velocities for S face 2.
      CALL IMS(ISF2)
      IF(PLANER)THEN
        CALL RVELP(3,RWX2,RWY2,RWZ2)
      ELSE
        CALL RVELC(3,RWX2,RWY2,RWZ2)
      END IF

C     Get r for face 2.  Theta is not needed.
      CALL RRTHFC(R2,DUM)

C     Get the psi1 curvature term.
      CALL RSI1CT(SI1C)

C     Get the psi2 curvature term.
      CALL RSI2CT(SI2C)

",filename]

ws1 : rwx1^2 + rwy1^2 + rwz1^2
deriv["(rho W)^2 at s face 1.",'ws1,"auxpc.sgf"]

ws2 : rwx2^2 + rwy2^2 + rwz2^2
deriv["(rho W)^2 at s face 2.",'ws2,"auxpc.sgf"]

/* Compressible flow */
or1 : (omg r1)^2 / 2
deriv["(omega r1)^2/2",'or1,"auxpc.sgf"]

ro1 : (gam ps1  + ((gam ps1)^2 + 2(gam-1)^2 (h1 + or1) ws1)^(1/2))/  \
      (2(gam-1)(h1 + or1))
deriv["rho at 1",'ro1,"auxpc.sgf"]

or2 : (omg r2)^2 / 2
deriv["(omega r2)^2/2",'or2,"auxpc.sgf"]

ro2 : (gam ps2  + ((gam ps2)^2 + 2(gam-1)^2 (h2 + or2) ws2)^(1/2))/  \
      (2(gam-1)(h2 + or2))
deriv["rho at 2",'ro2,"auxpc.sgf"]

ms1 : ws1/( ro1^2 (gam-1)(h1 - ws1/(2 ro1^2) + or1) )
deriv["Mach number at 1",'ms1,"auxpc.sgf"]

ms2 : ws2/( ro2^2 (gam-1)(h2 - ws2/(2 ro2^2) + or2) )
deriv["Mach number at 2",'ms2,"auxpc.sgf"]

msb : (ms1 + ms2)/2
deriv["Average Mach number.",'msb,"auxpc.sgf"]

pb : (ps1 + ps2)/2
deriv["Average Pressure.",'pb,"auxpc.sgf"]

pcacase : {-4 ka gam pb msb (1-msb) si1c,\
           0}
pcacond : {"MSB.LT.1.",\
           ".TRUE."}
casederiv[2,"2 P_c for the A equation.",pca,pcacase,pcacond,"auxpc.sgf"]
```

```
pcbcase : {-4 kb gam pb msb (1-msb) si2c,\
           0}
pcbcond : {"MSB.LT.1.",\
           ".TRUE."}
casederiv[2,"2 P_c for the B equation.",pcb,pcbcase,pcbcond,"auxpc.sgf"]

aer : ps1 + ps2 - pa1 - pa2 - pca
deriv["residual for A equation. (A error)",'aer,"auxpc.sgf"]

ber : ps1 + ps2 - pb1 - pb2 - pcb
deriv["residual for B equation. (B error)",'ber,"auxpc.sgf"]

/* A axilliary equation stored in X - Momentum Equation arrays. */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation array
C     NCX must be set.
      NCX = 0
C
C     ps1
      CALL ADCOFS(AERD1,IPSO+ISF1,COFX,ICOLX,NCX)
C
C     ps2
      CALL ADCOFS(AERD2,IPSO+ISF2,COFX,ICOLX,NCX)
C
C     pa1
      CALL ADCOFS(AERD3,IPAO+IPA1,COFX,ICOLX,NCX)
C
C     pa2
      CALL ADCOFS(AERD4,IPAO+IPA2,COFX,ICOLX,NCX)
",filename]

/* B axilliary equation stored in Y - Momentum Equation arrays. */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     Y - Momentum Equation array
C     NCY must be set.
      NCY = 0
C
C     ps1
      CALL ADCOFS(BERD1,IPSO+ISF1,COFY,ICOLY,NCY)
C
C     ps2
      CALL ADCOFS(BERD2,IPSO+ISF2,COFY,ICOLY,NCY)
C
C     pb1
      CALL ADCOFS(BERD5,IPBO+IPB1,COFY,ICOLY,NCY)
C
C     pb2
      CALL ADCOFS(BERD6,IPBO+IPB2,COFY,ICOLY,NCY)
",filename]

/* Z - Momentum Equation arrays. */

Lpr["
C
C     NCZ must be set.
      NCZ = 0

",filename]
```

```
Lpr["
C     The dependence on the velocities for S face 2.
C     Put A equation in x-momentum eq arrays and B eq in y-momentum eq arrays.
C     S face 2 is still initialized.
      IF(PLANER)THEN
        CALL LVELP(3,AERD10,AERD11,AERD12, BERD10,BERD11,BERD12,
     1             0.,0.,0.)
      ELSE
        CALL LVELC(3,AERD10,AERD11,AERD12, BERD10,BERD11,BERD12,
     1             0.,0.,0.)
      END IF

C     The dependence on r2.
      CALL LRTHFC(AERD16,0.,BERD16,0.,0.,0.)

C     The dependence on the velocities for S face 1.
C     Put A equation in x-momentum eq arrays and B eq in y-momentum eq arrays.
      CALL IMS(ISF1)
      IF(PLANER)THEN
        CALL LVELP(3,AERD7,AERD8,AERD9, BERD7,BERD8,BERD9, 0.,0.,0.)
      ELSE
        CALL LVELC(3,AERD7,AERD8,AERD9, BERD7,BERD8,BERD9, 0.,0.,0.)
      END IF

C     The dependence on r1.
      CALL LRTHFC(AERD15,0.,BERD15,0.,0.,0.)

C     The dependence of the eqns on the psi1 curvature term.
      CALL LSI1CT(AERD13,0.)

C     The dependence of the eqns on the psi2 curvature term.
      CALL LSI2CT(0.,BERD14)

",filename]

Lpr["


      RETURN
      END
",filename]

<"decfile.in"

decfile["Lauxpcdec.sgf"]
```

## File ENTC.IN

```
/* This session creates the subroutine which calculates the entropy
at an S face for compressible flow.  The left hand side subroutine is
also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy entc.sgf;* entc.osf
!dele entc.sgf;*
!@cgf entc.sgf

filename : "entc.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"
```

```
Lpr["
      SUBROUTINE RENTC(ISF,ENT)
C     Right calculation of entropy for an s-face for compressible flow.
C
C     This routine calculates the compressible entropy at the S face ISF.
C
      INTEGER ISF
      REAL ENT

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RENTCDEC.FOR'

C     Declare the contravarient velocities.
      REAL RWX,RWY,RWZ

C     Declare the radius and theta.
      REAL R,X1V

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

C     Get the radius and theta.
      CALL RRTHFC(R,X1V)

",filename]

/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the velocities.
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

rws : rwx^2 + rwy^2 + rwz^2
fort["(rho W)^2",'rws,"entc.sgf"]

/* Compressible flow */
or2 : (omg r)^2 / 2
fort["(omega r)^2/2",'or2,"entc.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
fort["density",'rho,"entc.sgf"]

s : gam/(gam-1) Log[(h - rws/(2 rho^2) + or2)/href] - Log[p/pref]
fort["entropy",'s,"entc.sgf"]

Lpr["
      ENT = S

C
      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rentcdec.sgf;* Rentcdec.osf
!dele Rentcdec.sgf;*
!@cgf Rentcdec.sgf

<"decfile.in"

decfile["Rentcdec.sgf"]

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "entc.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LENTC(ISF,DEQDS)
C     Left Entropy for an s-face for compressible flow.
C
C     This routine multiplies DEQDS by the entropy coefficients and adds
C     them to the COLX arrays.
C
      INTEGER ISF
      REAL DEQDS

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LENTCDEC.FOR'
      REAL DERI
C     Declare the contravarient velocities.
      REAL RWX,RWY,RWZ

C     Declare the radius and theta.
      REAL R,X1V

C     Get the face variables for this face.  Data passed
C     through common.
      CALL IMS(ISF)

C     Get the radius and theta.
      CALL RRTHFC(R,X1V)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {p,rwx,rwy,rwz,r}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_p[Const] : 1
_rwx[Const] : 1 ; _rwy[Const] : 1 ; _rwz[Const] : 1
_roc[Const] : 1
```

```
_r[Const] : 1
_gam[Const] : 1
_h[Const] : 1
_href[Const] : 1
_pref[Const] : 1
_omg[Const] : 1

/* Algorithm */

Lpr["

C     Get the velocities.
      IF(PLANER)THEN
        CALL RVELP(3,RWX,RWY,RWZ)
      ELSE
        CALL RVELC(3,RWX,RWY,RWZ)
      END IF

",filename]

rws : rwx^2 + rwy^2 + rwz^2
deriv["(rho W)^2",'rws,"entc.sgf"]

/* Compressible flow */
or2 : (omg r)^2 / 2
deriv["(omega r)^2/2",'or2,"entc.sgf"]

rho : (gam p  + ((gam p)^2 + 2(gam-1)^2 (h + or2) rws)^(1/2))/  \
      (2(gam-1)(h + or2))
deriv["density",'rho,"entc.sgf"]

s : gam/(gam-1) Log[(h - rws/(2 rho^2) + or2)/href] - Log[p/pref]
deriv["entropy",'s,"entc.sgf"]

/* Coefficients stored in X - Momentum Equation arrays. */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation array
C     NCX must be set.
C
C     ps
      CALL ADCOFS(DEQDS*SD1,IPSO+IP,COFX,ICOLX,NCX)
",filename]

/* Y and Z - Momentum Equation arrays. */

Lpr["
C
C     NCY and NCZ must be set.
      NCY = 0
      NCZ = 0

",filename]

Lpr["
C     The dependence on the velocities.  Put in x-momentum eq arrays.
      IF(PLANER)THEN
        CALL LVELP(3,DEQDS*SD2,DEQDS*SD3,DEQDS*SD4, 0.,0.,0., 0.,0.,0.)
      ELSE
        CALL LVELC(3,DEQDS*SD2,DEQDS*SD3,DEQDS*SD4, 0.,0.,0., 0.,0.,0.)
      END IF

C     The dependence on the radius.
      CALL LRTHFC(DEQDS*SD5,0., 0.,0., 0.,0.)
",filename]

Lpr["


      RETURN
      END
",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lentcdec.sgf;* Lentcdec.osf
!dele Lentcdec.sgf;*
!@cgf Lentcdec.sgf

<"decfile.in"

decfile["Lentcdec.sgf"]
```

## File RADEQ.IN

```
/* This session creates the subroutine which calculates the residuals to
the radial equilibrium equation.  The left
hand side subroutine is also created. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy radeq.sgf;* radeq.osf
!dele radeq.sgf;*
!@cgf radeq.sgf

filename : "radeq.sgf"

/* Load in the fort routine. */
<"fort.in"
<"casefort.in"

Lpr["
      SUBROUTINE RRADEQ(I,J)
C     Right RADial EQuilibrium equation.
C
C     This routine calculates the residual for the radial equilibrium
C     equation integrated from the S face at I, J, nk to I, J+1, nk.
C     The equation is in RESX.
C
      INTEGER I,J

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RRADEQDEC.FOR'

C     Declare the velocities.
      REAL RWX1,RWY1,RWZ1,RWX2,RWY2,RWZ2

C     Declare the r and theta at the face centers.
      REAL R1,R2,X11,X12

C     Initialize the variables for this equation.
      CALL IRADEQ(I,J)

",filename]
```

```
/* Initialize expression list. */
nel : 0

/* Algorithm */

Lpr["

C     Get the velocities for S face 1.
      CALL IMS(ISF1)
      CALL RVELC(3,RWX1,RWY1,RWZ1)

C     Get r and theta for face 1.
      CALL RRTHFC(R1,X11)

C     Get the velocities for S face 2.
      CALL IMS(ISF2)
      CALL RVELC(3,RWX2,RWY2,RWZ2)

C     Get r and theta for face 2.
      CALL RRTHFC(R2,X12)

",filename]

ws1 : rwx1^2 + rwy1^2 + rwz1^2
fort["(rho W)^2 at s face 1.",'ws1,"radeq.sgf"]

ws2 : rwx2^2 + rwy2^2 + rwz2^2
fort["(rho W)^2 at s face 2.",'ws2,"radeq.sgf"]

or1 : (omg r1)^2 / 2
fort["(omega r1)^2/2",'or1,"radeq.sgf"]

or2 : (omg r2)^2 / 2
fort["(omega r2)^2/2",'or2,"radeq.sgf"]

ro1case : {(gam ps1  + ((gam ps1)^2 + 2(gam-1)^2 (h1 + or1) ws1)^(1/2))/  \
      (2(gam-1)(h1 + or1)) ,\
           roc}
ro1cond : {"CMPRS",\
           ".TRUE."}
casefort[2,"rho at 1",ro1,ro1case,ro1cond,"radeq.sgf"]

ro2case : {(gam ps2  + ((gam ps2)^2 + 2(gam-1)^2 (h2 + or2) ws2)^(1/2))/  \
      (2(gam-1)(h2 + or2)) ,\
           roc}
ro2cond : {"CMPRS",\
           ".TRUE."}
casefort[2,"rho at 2",ro2,ro2case,ro2cond,"radeq.sgf"]

dr : r2 - r1
fort["Delta r",'dr,"radeq.sgf"]

dz : z2 - z1
fort["Delta z",'dz,"radeq.sgf"]

ds : (dr^2 + dz^2)^(1/2)
fort["Delta q.",'ds,"radeq.sgf"]

w1s : ws1/ro1^2
fort["W1^2.",'w1s,"radeq.sgf"]

w2s : ws2/ro2^2
fort["W2^2.",'w2s,"radeq.sgf"]

w1t : Cos[x11] rwx1/ro1 - Sin[x11] rwy1/ro1
```

```
fort["W_theta_1.",'w1t,"radeq.sgf"]

w2t : Cos[x12] rwx2/ro2 - Sin[x12] rwy2/ro2
fort["W_theta_2.",'w2t,"radeq.sgf"]

sm1 : w1s - w1t^2
fort["W_m squared.",'sm1,"radeq.sgf"]

sm2 : w2s - w2t^2
fort["W_m squared.",'sm2,"radeq.sgf"]

res : ps1 - ps2 + ds (ro2 c2 sm2 + ro1 c1 sm1)/2 + \
      1/2 (ro2 w2t^2 + ro1 w1t^2) Log[r2/r1] + \
      omg (ro2 w2t + ro1 w1t)dr + \
      omg^2/4 (ro2 + ro1) (r2^2 - r1^2)
fort["residual",'res,"radeq.sgf"]

Lpr["
      RESX = -RES

C
      RETURN
      END


",filename]

/* Create the declaration file for the variables in exprlist. */

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Rradeqdec.sgf;* Rradeqdec.osf
!dele Rradeqdec.sgf;*
!@cgf Rradeqdec.sgf

<"decfile.in"

decfile["Rradeqdec.sgf"]

/* Create the declaration file for the variables in exprlist which
will be output at the end of this routine.  This is done here rather
than at the end of the routine because of an error that was occuring.*/

/* Copy the previous files to another location and delete them. */
/* sgf is Smp Generated Fortran ;  osf is Old Smp Fortran. */
!copy Lradeqdec.sgf;* Lradeqdec.osf
!dele Lradeqdec.sgf;*
!@cgf Lradeqdec.sgf

/* Reinitialize everything to ensure enough room. */
Set[]

filename : "radeq.sgf"

/* Load in the fort and deriv routines. */
<"fort.in"
<"deriv.in"
<"casederiv.in"

Lpr["
      SUBROUTINE LRADEQ(I,J)
C     Left RADial EQuilibrium equation.
C
C     This routine calculates the left hand side coeficients for the
C     radial equilibrium equation integrated from the S face at I, J, nk
C     to I, J+1, nk.  The equation is in COFX arrays.
C
      INTEGER I,J

      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LRADEQDEC.FOR'
      REAL DERI

C     Declare the velocities.
      REAL RWX1,RWY1,RWZ1,RWX2,RWY2,RWZ2

C     Declare the r and theta at the face centers.
      REAL R1,R2,X11,X12

C     Initialize the variables for this equation.
      CALL IRADEQ(I,J)

",filename]

/* Initialize expression list. */
nel : 0

/* Initialize the dependant variable list. */
invar : {ps1,ps2,rwx1,rwy1,rwz1,rwx2,rwy2,rwz2,x11,x12}
niv : Len[invar]

/* Set the properties of the dependent variables and constants to Const
so they are considered independent by the derivative operator. */
_ps1[Const] : 1 ; _ps2[Const] : 1
_rwx1[Const] : 1 ; _rwy1[Const] : 1 ; _rwz1[Const] : 1
_rwx2[Const] : 1 ; _rwy2[Const] : 1 ; _rwz2[Const] : 1
_roc[Const] : 1
_c1[Const] : 1 ; _c2[Const] : 1
_h1[Const] : 1 ; _h2[Const] : 1
_gam[Const] : 1
_omg[Const] : 1
_r1[Const] : 1 ; _r2[Const] : 1
_z1[Const] : 1 ; _z2[Const] : 1
_x11[Const] : 1 ; _x12[Const] : 1

/* Algorithm */

Lpr["

C     Get the velocities for S face 1.
      CALL IMS(ISF1)
      CALL RVELC(3,RWX1,RWY1,RWZ1)

C     Get r and theta for face 1.
      CALL RRTHFC(R1,X11)

C     Get the velocities for S face 2.
      CALL IMS(ISF2)
      CALL RVELC(3,RWX2,RWY2,RWZ2)

C     Get r and theta for face 2.
      CALL RRTHFC(R2,X12)

",filename]

ws1 : rwx1^2 + rwy1^2 + rwz1^2
deriv["(rho W)^2 at s face 1.",'ws1,"radeq.sgf"]

ws2 : rwx2^2 + rwy2^2 + rwz2^2
deriv["(rho W)^2 at s face 2.",'ws2,"radeq.sgf"]
```

```
or1 : (omg r1)^2 / 2
deriv["(omega r1)^2/2",'or1,"radeq.sgf"]

or2 : (omg r2)^2 / 2
deriv["(omega r2)^2/2",'or2,"radeq.sgf"]

ro1case : {(gam ps1  + ((gam ps1)^2 + 2(gam-1)^2 (h1 + or1) ws1)^(1/2))/  \
      (2(gam-1)(h1 + or1)) ,\
           roc}
ro1cond : {"CMPRS",\
           ".TRUE."}
casederiv[2,"rho at 1",ro1,ro1case,ro1cond,"radeq.sgf"]

ro2case : {(gam ps2  + ((gam ps2)^2 + 2(gam-1)^2 (h2 + or2) ws2)^(1/2))/  \
      (2(gam-1)(h2 + or2)) ,\
           roc}
ro2cond : {"CMPRS",\
           ".TRUE."}
casederiv[2,"rho at 2",ro2,ro2case,ro2cond,"radeq.sgf"]

dr : r2 - r1
deriv["Delta r",'dr,"radeq.sgf"]

dz : z2 - z1
deriv["Delta z",'dz,"radeq.sgf"]

ds : (dr^2 + dz^2)^(1/2)
deriv["Delta q.",'ds,"radeq.sgf"]

w1s : ws1/ro1^2
deriv["W1^2.",'w1s,"radeq.sgf"]

w2s : ws2/ro2^2
deriv["W2^2.",'w2s,"radeq.sgf"]

w1t : Cos[x11] rwx1/ro1 - Sin[x11] rwy1/ro1
deriv["W_theta_1.",'w1t,"radeq.sgf"]

w2t : Cos[x12] rwx2/ro2 - Sin[x12] rwy2/ro2
deriv["W_theta_2.",'w2t,"radeq.sgf"]

sm1 : w1s - w1t^2
deriv["W_m squared.",'sm1,"radeq.sgf"]

sm2 : w2s - w2t^2
deriv["W_m squared.",'sm2,"radeq.sgf"]

res : ps1 - ps2 + ds (ro2 c2 sm2 + ro1 c1 sm1)/2 + \
      1/2 (ro2 w2t^2 + ro1 w1t^2) Log[r2/r1] + \
      omg (ro2 w2t + ro1 w1t)dr + \
      omg^2/4 (ro2 + ro1) (r2^2 - r1^2)
deriv["residual",'res,"radeq.sgf"]

/* Equation stored in X - Momentum Equation arrays. */

Lpr["
C
C     Put Jacobians in correct location in matrix.
C     X - Momentum Equation array
C     NCX must be set.
      NCX = 0
C
C     ps1
      CALL ADCOFS(RESD1,IPSO+ISF1,COFX,ICOLX,NCX)
C
C     ps2
```

```
      CALL ADCOFS(RESD2,IPSO+ISF2,COFX,ICOLX,NCX)
",filename]

/* Y - Momentum Equation arrays. */

Lpr["
C
C     NCY must be set.
      NCY = 0

",filename]

/* Z - Momentum Equation arrays. */

Lpr["
C
C     NCZ must be set.
      NCZ = 0

",filename]

Lpr["
C     The dependence on the velocities for S face 2.
C     Put equation in x-momentum eq arrays.
C     S face 2 is still initialized.
      CALL LVELC(3,RESD6,RESD7,RESD8, 0.,0.,0.,
     1               0.,0.,0.)

C     The dependence on theta 2.
      CALL LRTHFC(0.,RESD10,0.,0.,0.,0.)

C     The dependence on the velocities for S face 1.
C     Put equation in x-momentum eq arrays.
      CALL IMS(ISF1)
      CALL LVELC(3,RESD3,RESD4,RESD5, 0.,0.,0., 0.,0.,0.)

C     The dependence on theta 1.
      CALL LRTHFC(0.,RESD9,0.,0.,0.,0.)

",filename]

Lpr["


      RETURN
      END
",filename]

<"decfile.in"

decfile["Lradeqdec.sgf"]
```

# Appendix F
# Solver FORTRAN Source Code Listing

## Include Files

The files in this section contain declarations, common blocks and common code and are "included" in the other routines listed in the following sections. FACE.FOR is also used in the SMP generated routines and this input is in Appendix E.

### File 3DCOM.FOR

```
      INTEGER MNN,MNB,MNJ,MNDTH,MNBP,MNBGP,MNMPRM
      PARAMETER (MNN=10000,MNB=500,MNJ=41,MNDTH=200*MNJ)
      PARAMETER (MNBP=200,MNBGP=120,MNMPRM=120)

C            Geometry.
      REAL X1N,X2N,X3N
      REAL EMPRM
      REAL PITCH
      REAL SBLD,X1BLD,MPBLD
      REAL DX1DSB,DMPDSB,SGL,SGU
      INTEGER NPBLD
      REAL SBLE,DSBLEC,DSBLE,NX1G,NX2G,NX3G,NMPG
      LOGICAL THINLE,THNLES
      REAL AMXC,AMYC,BMXC,BMYC,SMXC,SMYC
      REAL EMPRMT,X2MT,X3MT
      INTEGER NMPRMT
      REAL X1POS,X3POS
      REAL DSFISP,TLEMIN,TLEMAX
      COMMON /GEOM/X1N(MNN),X2N(MNN),X3N(MNN),EMPRM(MNN),PITCH,
     1 SBLD(MNBP,MNJ),X1BLD(MNBP,MNJ),MPBLD(MNBP,MNJ),
     1 DX1DSB(MNBP,MNJ),DMPDSB(MNBP,MNJ),NPBLD(MNJ),
     1 SGL(MNBGP,MNJ),SGU(MNBGP,MNJ),SBLE(MNJ),DSBLEC(MNJ),DSBLE(MNJ),
     1 NX1G(MNN),NX2G(MNN),NX3G(MNN),NMPG(MNN),THINLE,THNLES,
     1 AMXC(MNN),AMYC(MNN),BMXC(MNN),
     1 BMYC(MNN),SMXC(MNN),SMYC(MNN),
     1 EMPRMT(MNMPRM,MNJ),X2MT(MNMPRM,MNJ),X3MT(MNMPRM,MNJ),NMPRMT(MNJ),
     1 X1POS(MNDTH),X3POS(MNDTH),DSFISP(MNJ),TLEMIN,TLEMAX

      REAL XN(MNN),YN(MNN),ZN(MNN)
      EQUIVALENCE (X1N,XN),(X2N,YN),(X3N,ZN)

C            Flow.
      REAL PA,PB,PS,PSI1,PSI2,MSS,VSCS
      INTEGER IVSCSU
      COMMON /FLOW/PA(MNN),PB(MNN),PS(MNN),PSI1(MNN),PSI2(MNN),MSS(MNN),
     1 VSCS(MNN),IVSCSU
```

```
C           Counters.
      INTEGER NI,NJ,NK,NN,N1E,N2E,NM,NSF,NAF,NBF,NB,NB1,NB2,KLE
      INTEGER KTE,NDTH,NDPA,NSBLEI,KSBLE,ICNTSM
      COMMON /COUNT/NI,NJ,NK,NN,N1E,N2E,NM,NSF,NAF,NBF,NB,NB1,NB2,KLE,
     1 KTE,NDTH,NDPA,NSBLEI,KSBLE,ICNTSM

C           Convergence Criteria.
      DOUBLE PRECISION BSQ
      REAL EPS,EPSLE
      INTEGER ITER,MAXIT
      REAL KPA,KPB
      LOGICAL SMOMAP
      INTEGER DMPOPT
      REAL DAMP
      LOGICAL SUBSON
      REAL ASTIF,BSTIF,EPSOF
      COMMON /CONV/BSQ,ITER,MAXIT,EPS,KPA,KPB,SMOMAP,EPSLE,DMPOPT,DAMP,
     1  SUBSON,ASTIF,BSTIF,EPSOF
      DATA EPSLE/1.E-9/
      DATA EPSOF/5.0E-8/

C           Descriptors.
      CHARACTER*80 DESC1,DESC2,DESC3,DESC4,FN
      COMMON /DESC/ DESC1,DESC2,DESC3,DESC4,FN

C           Boundary Conditions.
      REAL HUP,PSI1DN,PSI2DN,PSI1MX,PSI2MX,SUP
      INTEGER IHUP,ISUP,IDWNBC
      REAL DPAA,ANG1,LNORM,VNORM,PNORM,PDWN,CURV
      LOGICAL CYLIND,CMPRES
      INTEGER IMUNK,IDSOPT
      INTEGER IUPBC
      REAL PHIUP,CTHUP,CZBAR
      COMMON /BOUND/IHUP,ISUP,HUP(MNB),PSI1DN(MNB),PSI2DN(MNB),
     1 PSI1MX,PSI2MX,SUP(MNB),DPAA(MNDTH),ANG1(MNJ),
     1 CYLIND,CMPRES,LNORM,VNORM,PNORM,IDWNBC,PDWN(MNB),IMUNK,
     1 CURV(MNJ),IDSOPT,IUPBC,PHIUP(MNJ),CTHUP(MNJ),CZBAR(MNJ)

C           Versions.
      REAL SLVVN,BRWVN
      INTEGER IUNIT
      COMMON /VSN/ SLVVN,BRWVN,IUNIT
```

## File BINRD.FOR

```
      REWIND(IUNIT)

      NBLOCK = 500

      DO 800 N=4,24
        ISPEC(N) = 0
800   CONTINUE

C     Problem description.
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      READ(IUNIT)DESC1,DESC2,DESC3,DESC4

C     Binary read write version number.
      LABEL = 'BINARY R/W VN    '
      TYPE = LREAL
      NREC = 1
      NDATA = 2
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      READ(IUNIT)BRWVN,SLVVN
```

```
C     Integer parameters.
      LABEL = 'PARAMETERS      '
      TYPE = LINT
      NREC = 1
      NDATA = 24
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      READ(IUNIT)NI,NJ,NK,NN,N1E,N2E,NM,NSF,NAF,NBF,NB,NB1,NB2,KLE,
     1      KTE,NDTH,NDPA,IHUP,ISUP,ITER,IDWNBC,IMUNK,IDSOPT,IVSCSU

C     Real constants.
      LABEL = 'CONSTANTS       '
      TYPE = LREAL
      NREC = 1
      NDATA = 14
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      READ(IUNIT)GAM,OMG,ROC,PSI1MX,PSI2MX,EPS,KPA,KPB,PITCH,
     1      HREF,PREF,LNORM,VNORM,PNORM

C     Logical switches.
      LABEL = 'SWITCHES        '
      TYPE = LINT
      NREC = 1
      NDATA = 4
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      READ(IUNIT)ICYLIND,ICMPRES,ISMOMAP,ITHINLE

      IF(ICYLIND.EQ.1)THEN
        CYLIND = .TRUE.
      ELSE
        CYLIND = .FALSE.
      END IF

      IF(ICMPRES.EQ.1)THEN
        CMPRES = .TRUE.
      ELSE
        CMPRES = .FALSE.
      END IF

      IF(ISMOMAP.EQ.1)THEN
        SMOMAP = .TRUE.
      ELSE
        SMOMAP = .FALSE.
      END IF

      IF(ITHINLE.EQ.1)THEN
        THINLE = .TRUE.
      ELSE
        THINLE = .FALSE.
      END IF

C     Convergence criteria.
      LABEL = 'CONVERG CRIT    '
      TYPE = LDBL
      NREC = 1
      NDATA = 1
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      READ(IUNIT)BSQ

C     NJ values of NMPRMT, NPBLD.
      TYPE = LINT
      NDATA = NJ
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'NMPRMT          '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,NMPRMT)

      LABEL = 'NPBLD                   '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,NPBLD)

C     The rest of the data is real.
      TYPE = LREAL

C     NAF values of PA.
      LABEL = 'PA                      '
      NDATA = NAF
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PA)

C     NBF values of PB.
      LABEL = 'PB                      '
      NDATA = NBF
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PB)

C     NSF values of PS.
      LABEL = 'PS                      '
      NDATA = NSF
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PS)

C     NSF values of VSCS if IVSCSU=1.
      IF(IVSCSU.EQ.1)THEN
        LABEL = 'VSCS                    '
        NDATA = NSF
        NREC = (NDATA + NBLOCK - 1)/NBLOCK
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,VSCS)
      ELSE
        DO ISFI=1,NSF
          VSCS(ISFI) = 0.
        END DO
      END IF

C     N1E values of PSI1.
      LABEL = 'PSI1                    '
      NDATA = N1E
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PSI1)

C     N2E values of PSI2.
      LABEL = 'PSI2                    '
      NDATA = N2E
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PSI2)

C     NN values of X1N, X2N, X3N, and EMPRM.
      NDATA = NN
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'X1N                     '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,X1N)

      LABEL = 'X2N                     '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,X2N)

      LABEL = 'X3N             '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,X3N)

      LABEL = 'EMPRM           '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,EMPRM)

C     NJ values of SBLE.
      NDATA = NJ
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'SBLE            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,SBLE)

C     NM values of AMXC, AMYC, BMXC, BMYC, SMXC, and SMYC.
      NDATA = NM
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'AMXC            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,AMXC)

      LABEL = 'AMYC            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,AMYC)

      LABEL = 'BMXC            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,BMXC)

      LABEL = 'BMYC            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,BMYC)

      LABEL = 'SMXC            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,SMXC)

      LABEL = 'SMYC            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,SMYC)

C     NB values of HUP, SUP, and PDWN.
      NDATA = NB
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'HUP             '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,HUP)

      LABEL = 'SUP             '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,SUP)

      LABEL = 'PDWN            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PDWN)

C     NB1 values of PSI1DN.
      NDATA = NB1
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'PSI1DN          '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PSI1DN)

C     NJ values of ANG1 and CURV.
      NDATA = NJ
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'ANG1            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,ANG1)

      LABEL = 'CURV            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,CURV)

C     NB2 values of PSI2DN.
      NDATA = NB2
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'PSI2DN          '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,PSI2DN)

C     NDPA values of DPAA.
      NDATA = NDPA
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'DPAA            '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,DPAA)

C     NI values of X1POS.
      NDATA = NI
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'X1POS           '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,X1POS)

C     NK values of X3POS.
      NDATA = NK
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'X3POS           '
      READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL READR(IUNIT,NREC,NDATA,NBLOCK,X3POS)

C     These are done for each spanwise station.
      DO 810 N=1,NJ
        NDATA = NPBLD(N)
        NREC = (NDATA + NBLOCK - 1)/NBLOCK

        LABEL = 'X1BLD           '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,X1BLD(1,N))

        LABEL = 'MPBLD           '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,MPBLD(1,N))

        NDATA = NMPRMT(N)
        NREC = (NDATA + NBLOCK - 1)/NBLOCK

        LABEL = 'EMPRMT          '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,EMPRMT(1,N))

        LABEL = 'X2MT            '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,X2MT(1,N))

        LABEL = 'X3MT            '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,X3MT(1,N))

        NDATA = KTE - KLE + 1
        NREC = (NDATA + NBLOCK - 1)/NBLOCK

        LABEL = 'SGL             '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,SGL(1,N))

        LABEL = 'SGU             '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,SGU(1,N))
810   CONTINUE

      IF(BRWVN .GE. 1.1)THEN
C       Version 1.1 stuff.

C       Integer upstream boundary condition flag.
        LABEL = 'IUPBC           '
        TYPE = LINT
        NREC = 1
        NDATA = 1
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        READ(IUNIT)IUPBC

C       NJ values of CTHUP.
        TYPE = LREAL
        NDATA = NJ
        NREC = (NDATA + NBLOCK - 1)/NBLOCK

        LABEL = 'CTHUP           '
        READ(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL READR(IUNIT,NREC,NDATA,NBLOCK,CTHUP)
      ELSE
C       Set upstream bc to ANG1.
        IUPBC = 0
      END IF

      CLOSE(IUNIT)
```

## File BINWRT.FOR

```
      REWIND(IUNIT)

      BRWVN = 1.1

      NBLOCK = 500
      FACTOR = UNITY

      DO 800 N=4,24
        ISPEC(N) = 0
800   CONTINUE

C     Problem description.
      LABEL = 'PROBLEM DESCR   '
      TYPE = LC80
      NREC = 1
      NDATA = 4
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      WRITE(IUNIT)DESC1,DESC2,DESC3,DESC4
```

```
C     Binary read write version number.
      LABEL = 'BINARY R/W VN    '
      TYPE = LREAL
      NREC = 1
      NDATA = 2
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      WRITE(IUNIT)BRWVN,SLVVN

C     Integer parameters.
      LABEL = 'PARAMETERS       '
      TYPE = LINT
      NREC = 1
      NDATA = 24
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      WRITE(IUNIT)NI,NJ,NK,NN,N1E,N2E,NM,NSF,NAF,NBF,NB,NB1,NB2,KLE,
     1      KTE,NDTH,NDPA,IHUP,ISUP,ITER,IDWNBC,IMUNK,IDSOPT,IVSCSU

C     Real constants.
      LABEL = 'CONSTANTS        '
      TYPE = LREAL
      NREC = 1
      NDATA = 14
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      WRITE(IUNIT)GAM,OMG,ROC,PSI1MX,PSI2MX,EPS,KPA,KPB,PITCH,
     1      HREF,PREF,LNORM,VNORM,PNORM

C     Logical switches.
      IF(CYLIND)THEN
        ICYLIND = 1
      ELSE
        ICYLIND = 0
      END IF

      IF(CMPRES)THEN
        ICMPRES = 1
      ELSE
        ICMPRES = 0
      END IF

      IF(SMOMAP)THEN
        ISMOMAP = 1
      ELSE
        ISMOMAP = 0
      END IF

      IF(THINLE)THEN
        ITHINLE = 1
      ELSE
        ITHINLE = 0
      END IF

      LABEL = 'SWITCHES         '
      TYPE = LINT
      NREC = 1
      NDATA = 4
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      WRITE(IUNIT)ICYLIND,ICMPRES,ISMOMAP,ITHINLE

C     Convergence criteria.
      LABEL = 'CONVERG CRIT     '
      TYPE = LDBL
      NREC = 1
      NDATA = 1
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      WRITE(IUNIT)BSQ
```

```
C     NJ values of NMPRMT, NPBLD.
      TYPE = LINT
      NDATA = NJ
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'NMPRMT          '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,NMPRMT)

      LABEL = 'NPBLD           '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,NPBLD)

C     Most of the rest of the data is real.
      TYPE = LREAL

C     NAF values of PA.
      LABEL = 'PA              '
      NDATA = NAF
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PA)

C     NBF values of PB.
      LABEL = 'PB              '
      NDATA = NBF
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PB)

C     NSF values of PS.
      LABEL = 'PS              '
      NDATA = NSF
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PS)

C     NSF values of VSCS if IVSCSU=1.
      IF(IVSCSU.EQ.1)THEN
        LABEL = 'VSCS            '
        NDATA = NSF
        NREC = (NDATA + NBLOCK - 1)/NBLOCK
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,VSCS)
      END IF

C     N1E values of PSI1.
      LABEL = 'PSI1            '
      NDATA = N1E
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PSI1)

C     N2E values of PSI2.
      LABEL = 'PSI2            '
      NDATA = N2E
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PSI2)

C     NN values of X1N, X2N, X3N, and EMPRM.
      NDATA = NN
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'X1N             '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X1N)

      LABEL = 'X2N             '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X2N)

      LABEL = 'X3N             '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X3N)

      LABEL = 'EMPRM           '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,EMPRM)

C     NJ values of SBLE.
      NDATA = NJ
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'SBLE            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,SBLE)

C     NM values of AMXC, AMYC, BMXC, BMYC, SMXC, and SMYC.
      NDATA = NM
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'AMXC            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,AMXC)

      LABEL = 'AMYC            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,AMYC)

      LABEL = 'BMXC            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,BMXC)

      LABEL = 'BMYC            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,BMYC)

      LABEL = 'SMXC            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,SMXC)

      LABEL = 'SMYC            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,SMYC)

C     NB values of HUP, SUP, and PDWN.
      NDATA = NB
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'HUP             '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,HUP)

      LABEL = 'SUP             '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,SUP)

      LABEL = 'PDWN            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PDWN)
```

```
C     NB1 values of PSI1DN.
      NDATA = NB1
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'PSI1DN          '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PSI1DN)

C     NJ values of ANG1 and CURV.
      NDATA = NJ
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'ANG1            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,ANG1)

      LABEL = 'CURV            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,CURV)

C     NB2 values of PSI2DN.
      NDATA = NB2
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'PSI2DN          '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,PSI2DN)

C     NDPA values of DPAA.
      NDATA = NDPA
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'DPAA            '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,DPAA)

C     NI values of X1POS.
      NDATA = NI
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'X1POS           '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X1POS)

C     NK values of X3POS.
      NDATA = NK
      NREC = (NDATA + NBLOCK - 1)/NBLOCK
      LABEL = 'X3POS           '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X3POS)

C     These are done for each spanwise station.
      DO 810 N=1,NJ
        NDATA = NPBLD(N)
        NREC = (NDATA + NBLOCK - 1)/NBLOCK

        LABEL = 'X1BLD           '
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X1BLD(1,N))

        LABEL = 'MPBLD           '
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,MPBLD(1,N))

        NDATA = NMPRMT(N)
        NREC = (NDATA + NBLOCK - 1)/NBLOCK

        LABEL = 'EMPRMT          '
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,EMPRMT(1,N))

        LABEL = 'X2MT            '
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X2MT(1,N))

        LABEL = 'X3MT            '
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,X3MT(1,N))

        NDATA = KTE - KLE + 1
        NREC = (NDATA + NBLOCK - 1)/NBLOCK

        LABEL = 'SGL             '
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,SGL(1,N))

        LABEL = 'SGU             '
        WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
        CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,SGU(1,N))
810   CONTINUE

C     Integer upstream boundary condition flag.
      LABEL = 'IUPBC           '
      TYPE = LINT
      NREC = 1
      NDATA = 1
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      WRITE(IUNIT)IUPBC

C     NJ values of CTHUP.
      TYPE = LREAL
      NDATA = NJ
      NREC = (NDATA + NBLOCK - 1)/NBLOCK

      LABEL = 'CTHUP           '
      WRITE(IUNIT)LABEL,TYPE,(ISPEC(J),J=1,24),FACTOR
      CALL WRTER(IUNIT,NREC,NDATA,NBLOCK,CTHUP)
```

## File FACE.FOR

```
C     This is the FACE common.
      INTEGER MNC
      PARAMETER (MNC=1000)

      REAL A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12
      REAL B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12
      REAL P,H,P1,M1S,PO,VSC,MCS,GAM,OMG,ROC,AO,BO
      INTEGER IA1,IA2,IA3,IA4,IA5,IA6,IA7,IA8,IA9,IA10,IA11,IA12
      INTEGER IB1,IB2,IB3,IB4,IB5,IB6,IB7,IB8,IB9,IB10,IB11,IB12
      INTEGER IP,IP1,IPO
      REAL RESX,RESY,RESZ
      REAL COFX,COFY,COFZ
      INTEGER ICOLX,ICOLY,ICOLZ,NCX,NCY,NCZ
      INTEGER IPSI10,IPSI20,IPAO,IPBO,IPSO,IDTHO,ISBLEO,IFTYP,ISAVE
      REAL SD,S1,S2,S3,S4,S5
      REAL ASD,AS1,AS2,AS3,AS4,AS5
      REAL BSD,BS1,BS2,BS3,BS4,BS5
      INTEGER ISFD,ISF1,ISF2,ISF3,ISF4,ISF5
      LOGICAL PLANER

      COMMON /FACE/A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,
     1 B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,
     1 P,H,P1,M1S,PO,VSC,MCS,GAM,OMG,ROC,AO,BO,
     1 IA1,IA2,IA3,IA4,IA5,IA6,IA7,IA8,IA9,IA10,IA11,IA12,
     1 IB1,IB2,IB3,IB4,IB5,IB6,IB7,IB8,IB9,IB10,IB11,IB12,
     1 IP,IP1,IPO, RESX,RESY,RESZ,
     1 COFX(MNC),COFY(MNC),COFZ(MNC),
     1 ICOLX(MNC),ICOLY(MNC),ICOLZ(MNC),NCX,NCY,NCZ,
     1 IPSI10,IPSI20,IPAO,IPBO,IPSO,IDTHO,ISBLEO,IFTYP,ISAVE,
     1 SD,S1,S2,S3,S4,S5, ASD,AS1,AS2,AS3,AS4,AS5,
     1 BSD,BS1,BS2,BS3,BS4,BS5, ISFD,ISF1,ISF2,ISF3,ISF4,ISF5,
     1 PLANER

C     Equivalence 2 arrays IA and IB to be the same as IAi and IBi.
      INTEGER IA(12),IB(12)
      EQUIVALENCE (IA,IA1),(IB,IB1)

C     New Face variables.  Separate into two commons so fewer continuations.
      INTEGER LA,LB,LC,LD,LE,LF,LG,LH,LI,LJ,LK,LL
      REAL X1AC,X1BC,X1CC,X1DC,X1EC,X1FC,X1GC,X1HC,X1IC,X1JC,X1KC,X1LC
      REAL X2AC,X2BC,X2CC,X2DC,X2EC,X2FC,X2GC,X2HC,X2IC,X2JC,X2KC,X2LC
      REAL X3AC,X3BC,X3CC,X3DC,X3EC,X3FC,X3GC,X3HC,X3IC,X3JC,X3KC,X3LC
      REAL D1DLEA,D1DLEB,D1DLEC,D1DLED
      REAL D1DLEE,D1DLEF,D1DLEG,D1DLEH,D1DLEI,D1DLEJ,D1DLEK,D1DLEL
      REAL D2DLEA,D2DLEB,D2DLEC,D2DLED
      REAL D2DLEE,D2DLEF,D2DLEG,D2DLEH,D2DLEI,D2DLEJ,D2DLEK,D2DLEL
      REAL D3DLEA,D3DLEB,D3DLEC,D3DLED
      REAL D3DLEE,D3DLEF,D3DLEG,D3DLEH,D3DLEI,D3DLEJ,D3DLEK,D3DLEL
      INTEGER JSLEAD,JSLEBC,NJJ,JSLEU,JSLEL

      COMMON /FACEN/LA,LB,LC,LD,LE,LF,LG,LH,LI,LJ,LK,LL,
     1 X1AC,X1BC,X1CC,X1DC,X1EC,X1FC,X1GC,X1HC,X1IC,X1JC,X1KC,X1LC,
     1 X2AC,X2BC,X2CC,X2DC,X2EC,X2FC,X2GC,X2HC,X2IC,X2JC,X2KC,X2LC,
     1 X3AC,X3BC,X3CC,X3DC,X3EC,X3FC,X3GC,X3HC,X3IC,X3JC,X3KC,X3LC,
     1 D1DLEA,D1DLEB,D1DLEC,D1DLED,
     1 D1DLEE,D1DLEF,D1DLEG,D1DLEH,D1DLEI,D1DLEJ,D1DLEK,D1DLEL,
     1 D2DLEA,D2DLEB,D2DLEC,D2DLED,
     1 D2DLEE,D2DLEF,D2DLEG,D2DLEH,D2DLEI,D2DLEJ,D2DLEK,D2DLEL,
     1 D3DLEA,D3DLEB,D3DLEC,D3DLED,
     1 D3DLEE,D3DLEF,D3DLEG,D3DLEH,D3DLEI,D3DLEJ,D3DLEK,D3DLEL,
     1 JSLEAD,JSLEBC,NJJ,JSLEU,JSLEL

      REAL X1A,X2A,X3A,X1B,X2B,X3B
      REAL DX11,DX12,DX13,DX14,DX15,DX16,DX17,DX18,DX19
      INTEGER IDX11,IDX12,IDX13,IDX14,IDX15,IDX16,IDX17,IDX18,IDX19
      INTEGER MGD,JVN,IDTH1,IDTH2,IDTH3,IDTH4
      REAL DX11N,DX12N,DX13N,DX14N
      REAL J12D,J22D,J32D,J42D,J52D,J62D
      REAL PS1,PS2,PA1,PA2,PB1,PB2
      REAL AP1,AP2,AP3,AP4,AP5,AP6
      REAL BP1,BP2,BP3,BP4,BP5,BP6
      INTEGER IPA1,IPA2,IPB1,IPB2
      INTEGER ISI11,ISI12,ISI13,ISI14,ISI15,ISI16
      INTEGER ISI21,ISI22,ISI23,ISI24,ISI25,ISI26
      REAL KA,KB,H1,H2
      INTEGER JBLEI1,JBLEI2,JBLEI3,JBLEI4
      REAL X1LEA,X2LEA,X3LEA,X1LEB,X2LEB,X3LEB
      REAL PREF,HREF
      REAL C1,C2,Z1,Z2
      LOGICAL CMPRS
      REAL VISCM

      COMMON /FACEN2/X1A(12),X2A(12),X3A(12),X1B(12),X2B(12),X3B(12),
     1 DX11,DX12,DX13,DX14,DX15,DX16,DX17,DX18,DX19,
     1 IDX11,IDX12,IDX13,IDX14,IDX15,IDX16,IDX17,IDX18,IDX19,
     1 MGD,JVN,IDTH1,IDTH2,IDTH3,IDTH4,
     1 DX11N,DX12N,DX13N,DX14N,
     1 J12D,J22D,J32D,J42D,J52D,J62D,
     1 PS1,PS2,PA1,PA2,PB1,PB2, AP1,AP2,AP3,AP4,AP5,AP6,
     1 BP1,BP2,BP3,BP4,BP5,BP6,
     1 IPA1,IPA2,IPB1,IPB2, ISI11,ISI12,ISI13,ISI14,ISI15,ISI16,
     1 ISI21,ISI22,ISI23,ISI24,ISI25,ISI26,
     1 KA,KB,H1,H2, JBLEI1,JBLEI2,JBLEI3,JBLEI4,
     1 X1LEA(12),X2LEA(12),X3LEA(12),X1LEB(12),X2LEB(12),X3LEB(12),
     1 PREF,HREF, C1,C2,Z1,Z2, CMPRS, VISCM

      DATA DX11,DX12,DX13,DX14,DX15,DX16,DX17,DX18,DX19/9*0./
      DATA DX11N,DX12N,DX13N,DX14N/4*0./
```

## File MATCOM.FOR

```
      INTEGER MNCR,MNEQ,MNSV,MNEV,MWS,MNTS
      PARAMETER (MNCR=1000,MNEQ=3*MNN)
      PARAMETER (MNSV=5*MNN,MNEV=10)
      PARAMETER (MWS=7100000)
      PARAMETER (MNTS=9)

C           Matrix Storage.
      REAL COFA
      INTEGER ICOLA,NA,IR,NEQ
      INTEGER IRHS,INDROW,ISSP,IISP,IWSMSU,ICWSB
      REAL WS
      INTEGER IWS(MWS)
      COMMON /MAT/COFA(MNCR),ICOLA(MNCR),NA,IR,NEQ,
     1      IRHS,INDROW,ISSP,IISP,IWSMSU,
     1      ICWSB,WS(MWS)
      EQUIVALENCE (IWS,WS)

C           Equation Ordering.
      INTEGER NCOL,IEQNO,IEQPTR
      INTEGER IOQNO(MNEQ),IOQPTR(MNEQ),IPERM(MNEQ),IP21(MNEQ)
      INTEGER LENR(MNEQ),IWSOEQ(4*MNEQ),ICN(10*MNEQ)
      COMMON /EQORD/NCOL,IEQNO(MNEQ),IEQPTR(MNEQ)
      EQUIVALENCE (WS(2*MNEQ+1),IOQNO)
      EQUIVALENCE (WS(3*MNEQ+1),IOQPTR)
      EQUIVALENCE (WS(4*MNEQ+1),IPERM)
      EQUIVALENCE (WS(5*MNEQ+1),IP21)
      EQUIVALENCE (WS(6*MNEQ+1),LENR)
      EQUIVALENCE (WS(7*MNEQ+1),IWSOEQ)
      EQUIVALENCE (WS(11*MNEQ+1),ICN)

C           Linesearch, SX is not used.
      REAL ALPREV,PNS,GC,MAXSTP,STPTOL
      INTEGER MAJERR
      COMMON /LNSRCH/ALPREV,PNS(MNSV),GC(MNSV),MAXSTP,STPTOL,
     1      MAJERR

C           Original Matrix Storage.
      REAL FVEC(MNSV)
      EQUIVALENCE (FVEC,WS)
      INTEGER IIWSOM,IAJ,IJCN,IJDROW,NSV
      COMMON /OMS/IIWSOM,IAJ,IJCN,IJDROW,NSV

C           Vector of Equations.
      REAL COFV,RHSV
      INTEGER ICOLV,NCV,NEV
      COMMON /VECTEQ/COFV(MNCR,MNEV),ICOLV(MNCR,MNEV),NCV(MNEV),
     1      RHSV(MNEV),NEV

C           Equation error statistics.
      REAL ESMMAX,EAMMAX,EBMMAX,EPAMAX,EPBMAX,EBCMAX
      REAL ESMAVG,EAMAVG,EBMAVG,EPAAVG,EPBAVG,EBCAVG
      INTEGER MSMMAX,MAMMAX,MBMMAX,MPAMAX,MPBMAX,IBCMAX
      COMMON /ERSTAT/ ESMMAX,EAMMAX,EBMMAX,EPAMAX,EPBMAX,EBCMAX,
     1 ESMAVG,EAMAVG,EBMAVG,EPAAVG,EPBAVG,EBCAVG,
     1 MSMMAX,MAMMAX,MBMMAX,MPAMAX,MPBMAX,IBCMAX

C              Entropy equation matrix indices.
      INTEGER ISD,IPDROW,ISPMSP,IIPMSP,IIWSE
      LOGICAL USEENT
      COMMON /ENTMAT/ISD,IPDROW,ISPMSP,IIPMSP,IIWSE,USEENT

C              GMRES storage.
      INTEGER KRYDIM
      REAL EPSM
      LOGICAL GMDISK
      COMMON /GMRES/KRYDIM,EPSM,GMDISK

C              Entropy and Rothalpy Tracking.
      LOGICAL SMMODE,ISSMAP
      REAL POLD,PSI1UT,PSI2UT,PSI1CT,PSI2CT,CST,PSI1S,PSI2S,S,SFUP
      COMMON /TRACK/SMMODE,ISSMAP(MNN),POLD(MNN),
     1        PSI1UT(MNB),PSI2UT(MNB),PSI1CT(MNB),PSI2CT(MNB),
     1        CST(MNB),PSI1S(MNB),PSI2S(MNB),S(MNN),SFUP(MNB)
```

## File MESSAG.FOR

```
C     This is the message common.

C     MSGLVG is the message level for the GMRESS routine.
      INTEGER MSGLVG
C     MSGLVM is the message level for the MATSLV routine.
      INTEGER MSGLVM
C     MSGLMI is the message level for the MATIT routine.
      INTEGER MSGLMI
C     MSGLVS is the message level for the SLVMAT routine.
      INTEGER MSGLVS
C     MSGLVO is the message level for the ONEIT routine.
      INTEGER MSGLVO
C     MSGLVR is the message level for the RHSSQ routine.
      INTEGER MSGLVR
C     MSGLVP is the message level for the PFRVSC routine.
      INTEGER MSGLVP
C     MSGLVE is the message level for the ELLIP routine.
      INTEGER MSGLVE
      COMMON /MESSAG/MSGLVG,MSGLVM,MSGLMI,MSGLVS,MSGLVO,MSGLVR,
     1               MSGLVP,MSGLVE
      DATA MSGLVG,MSGLVM,MSGLMI,MSGLVS,MSGLVO,MSGLVR/6*0/
      DATA MSGLVP,MSGLVE/0,0/
```

## File RWINC.FOR

```
C     This is the include file which contains the declarations used by the
C     TARA IO routines.
      INTEGER ISFI
      INTEGER ICYLIND,ICMPRES,ISMOMAP,ITHINLE
      CHARACTER*16 LABEL,TYPE
      DOUBLE PRECISION FACTOR,UNITY
      DATA UNITY/1.D0/
      INTEGER ISPEC(24),NREC,NBLOCK,NDATA
      EQUIVALENCE (ISPEC(1),NREC),(ISPEC(2),NDATA),(ISPEC(3),NBLOCK)
C+++++++++++++++++++  INCLUDE  FILE  'LTYP.INC' +++++++++++++++++++
C
      CHARACTER*16 LINT
      DATA         LINT    / 'INTEGER         ' /
      CHARACTER*16 LREAL
      DATA         LREAL   / 'REAL            ' /
      CHARACTER*16 LC80
      DATA         LC80    / 'CHARACTER*80    ' /
      CHARACTER*16 LTEXT
      DATA         LTEXT   / 'TEXT            ' /
      CHARACTER*16 LEOF
      DATA         LEOF    / 'END OF FILE     ' /
      CHARACTER*16 LDBL
      DATA         LDBL    / 'DOUBLE PRECISION' /
      CHARACTER*16 LLOG
      DATA         LLOG    / 'LOGICAL         ' /
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## Master Code Files

The files in this section are .MST files which have machine dependant directives in them which are read by a preprocessor to produce the code for a given computer. This allowed the same code to be used on both the CRAY and VAX.

### File TARAIO.MST

```
*&DEFINED VAX CRAY
      SUBROUTINE INIT
C     This routine initializes the algorithm.

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'
      INCLUDE 'MESSAG.FOR'
      INCLUDE 'RWINC.FOR'

      INTEGER I,J,M,N,N1,N2,IREAD,ITERAD,USCLMP
      REAL DS1,DS2
      REAL EMPAVG,DRDM,DZDM

      LOGICAL       LEND
      INTEGER ILSPF
      COMMON /CLSPF / ILSPF,LEND

      LEND = .TRUE.

C     Unit for reading input.
*&IF VAX
      IREAD = 5
*&ELSE
      IREAD = 10
*&END IF

*&IF VAX

      WRITE(6,10)
10    FORMAT(' Enter input file name')
      READ(5,20) FN
20    FORMAT(A80)

      IUNIT = 11
```

```
       OPEN(UNIT=IUNIT,FILE=FN,STATUS='OLD',
     &        FORM='UNFORMATTED',RECORDTYPE='VARIABLE',
     &        SHARED)

      GMDISK = .FALSE.

*&ELSE

      IUNIT = 11

      OPEN(UNIT=IUNIT,STATUS='OLD',FORM='UNFORMATTED',FILE='FINPUT')

      GMDISK = .TRUE.

*&END IF

      INCLUDE 'BINRD.FOR'

      WRITE(6,'(1X,A79)') DESC1
      WRITE(6,'(1X,A79)') DESC2
      PRINT*,'NI=,NJ=,NK=,KLE=,KTE=',NI,NJ,NK,KLE,KTE
      IF(CMPRES)THEN
        PRINT*,' Compressible.'
      ELSE
        PRINT*,' Incompressible.'
      END IF
      IF(THINLE)THEN
        PRINT*,' Thin leading edge.  Does not move.'
      ELSE
        PRINT*,' Leading edge allowed to move.'
      END IF
      IF(SMOMAP)THEN
        PRINT*,' S momentum equation to be applied.'
      ELSE
        PRINT*,' Entropy convection equaton will be applied.'
      END IF
      IF(ISUP.EQ.0)THEN
        PRINT*,' Entropy is uniform upstream.'
      ELSE
        PRINT*,' Entropy is variable upstream.'
      END IF
      IF(IHUP.EQ.0)THEN
        PRINT*,' Rothalpy is uniform upstream.'
        IF(IMUNK.EQ.1)PRINT*,' Munk and Primm being applied.'
      ELSE
        PRINT*,' Rothalpy is variable upstream.'
      END IF
      IF(CYLIND)THEN
        PRINT*,' Cylindrical Coordinate system. Omega = ',OMG
      ELSE
        PRINT*,' Cartesian Coordinate system.'
      END IF
      PRINT*,'IDWNBC = ',IDWNBC

*&IF VAX
C     Open the output file as file 2.  Write to it after each iteration.
      WRITE(6,30)
30    FORMAT(' Enter output file name')
      READ(5,20) FN

      IUNIT = 12

      OPEN(UNIT=IUNIT,FILE=FN,STATUS='NEW',
     &        FORM='UNFORMATTED',RECORDTYPE='VARIABLE')
```

```
*&ELSE

      IUNIT = 12

      OPEN(UNIT=IUNIT,STATUS='NEW',FORM='UNFORMATTED',FILE='FOUTPUT')

*&END IF

C     Initialize Iterative Matrix Solver Parameters.
      PRINT*,'Krylov Subspace dimension?'
      READ(IREAD,*)KRYDIM
      WRITE(6,*)KRYDIM

      PRINT*,'Convergence criteria in matrix solver?'
      READ(IREAD,*)EPSM
      WRITE(6,*)EPSM

      PRINT*,'Number of additional iterations?'
      READ(IREAD,*)ITERAD
      WRITE(6,*)ITERAD

C     Check if line search is desired.  If not, damping must be constant
C     on the CRAY.

      PRINT*,'Damping Option?  1-line search; 2-const damping;'
      PRINT*,'3-interactive damping factor (not on CRAY or in batch)'
      READ(IREAD,*)DMPOPT
      WRITE(6,*)DMPOPT

      IF(DMPOPT.EQ.2)THEN
        PRINT*,'Damping Factor?'
        READ(IREAD,*)DAMP
        WRITE(6,*)DAMP
      END IF

C     Momentum equation Stiffeners.
      ASTIF  = 0.
      BSTIF = 0.

C     Check if Ps will be updated  upstream based on rho W and entropy.
C     If it is best for a supersonic or transonic inlet to not use this.
C     It is always applied if iter=0.
      PRINT*,'Ps updated upstream based on entropy? (T/F)'
      READ(IREAD,*)SUBSON
      WRITE(6,*)SUBSON

      MAXIT = ITER + ITERAD

      NSBLEI = NJ - 1

      NEQ = N1E + N2E + NSF
      NSV = NEQ + NAF + NBF + NDTH + NSBLEI
      PRINT*,'Number of total unknowns = ',NSV

      IPSI10 = 0
      IPSI20 = N1E
      IPSO = N1E + N2E
      IPAO = N1E + N2E + NSF
      IPBO = N1E + N2E + NSF + NAF
      IDTHO = IPBO + NBF
      ISBLEO = IDTHO + NDTH

C     Get the A, B, and S unit vectors for each volume if iter = 0.
C     These are kept constant unless the grid is regenerated.
      IF(ITER.EQ.0)CALL GETVUV
      DO 40 N=1,NSBLEI
        DSBLEC(N) = 0.
40    CONTINUE

C     Define the k station where the leading edge arc lengths are stored.
      IF(NK.GT.KTE)THEN
C       There is a trailing edge so put it here.
        KSBLE = KTE
      ELSE
        KSBLE = NK - 1
      END IF


C     Check if leading edge clamping is desired.  EPSLE in a data statement
C     is used for this.
      PRINT*,'Is leading edge clamping desired? 0-no 1-yes.'
      PRINT*,'Input anything if not a leading edge.'
      READ(IREAD,*)USCLMP
      WRITE(6,*)USCLMP

      THNLES = THINLE

      IF(USCLMP.EQ.1)THEN
C       For iter=0, save the value of THINLE, and set it to .true.
C       until further converged.
        IF(ITER.EQ.0)THEN
          THNLES = THINLE
          THINLE = .TRUE.
        END IF
      END IF

      IF(IHUP.EQ.1)THEN
        PRINT*,'CANNOT HANDLE VARIABLE ROTHAPY YET.'
        STOP
      ELSE
        H = HUP(1)
      END IF

      SMMODE = SMOMAP
      CMPRS = CMPRES

      IF(SMMODE)THEN
        DO 60 M=1,NM
          ISSMAP(M) = .TRUE.
60      CONTINUE
      ELSE
        DO 70 M=1,NM
          ISSMAP(M) = .FALSE.
70      CONTINUE
      END IF

      ISAVE = 0

      IF(CYLIND)THEN
        PLANER = .FALSE.
      ELSE
         PLANER = .TRUE.
      END IF

C      Set up the arc length and derivatives of the blade splines.
      DO 80 J=1,NJ
        IF(NPBLD(J).GT.1)THEN
          CALL SCALC(X1BLD(1,J),MPBLD(1,J),SBLD(1,J),NPBLD(J))
          CALL SPLINE(X1BLD(1,J),DX1DSB(1,J),SBLD(1,J),NPBLD(J))
          CALL SPLINE(MPBLD(1,J),DMPDSB(1,J),SBLD(1,J),NPBLD(J))
        END IF
```

```
C       Get the first leading edge interval.
        IF(KLE.GT.1)THEN
          N1 = (KLE-1)*NI*NJ + (J-1)*NI + 1
          N2 = N1 + NI*NJ
          DS1 = SQRT( (X1N(N1)-X1N(N2))**2 + (EMPRM(N1)-EMPRM(N2))**2 )
          N1 = (KLE-1)*NI*NJ + (J-1)*NI + NI
          N2 = N1 + NI*NJ
          DS2 = SQRT( (X1N(N1)-X1N(N2))**2 + (EMPRM(N1)-EMPRM(N2))**2 )
          DSFISP(J) = 0.5*(DS1 + DS2)
C         Get phi upstream for C_theta boundary condition.
          IF(IUPBC.EQ.1)THEN
            IF(PLANER)THEN
              PHIUP(J) = 0.
            ELSE
C             Phi is arc tan( dr / dz ).  Evaluate it midway between first two
C             k stations.
              N1 = (J-1)*NI + 1
              N2 = N1 + NI*NJ
              EMPAVG = 0.5*(EMPRM(N1) + EMPRM(N2))
              CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1          EMPAVG,DRDM,1, 1)
              CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1          EMPAVG,DZDM,1, 1)
              PHIUP(J) = ATAN2(DRDM,DZDM)
            END IF
          END IF
        END IF
80    CONTINUE

C     The first leading edge interval tolerances.
      TLEMIN = 0.6
      TLEMAX = 1.4

      PRINT*,'Critical Mach number squared M_c^2? '
      PRINT*,'Input anything if incompressible.'
      READ(IREAD,*)MCS
      WRITE(6,*)MCS

      VISCM = 1.0

      NJJ = NJ

C     Set up what to do if there is trouble with convergence.
      MAJERR = 0

C     Message levels.
      MSGLVS = 2
      MSGLVO = 1
      MSGLVR = 1

      RETURN
      END

      SUBROUTINE OUT
C     This routine writes the output file.

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'
      INCLUDE 'RWINC.FOR'

      INTEGER I,J,N
      LOGICAL THNSS

C     Write out the correct thinle, but change back to what it was
C     temporarily.
      THNSS = THINLE
      THINLE = THNLES

      IUNIT = 12

      INCLUDE 'BINWRT.FOR'

      THINLE = THNSS

      RETURN
      END
```

## File TARAMAT2.MST

```
*&DEFINED VAX CRAY
C     *********************************************************************
C
C     FROM SKYLIN.FOR
C
C     *********************************************************************
C         SKYLINE MATRIX SOLUTION ROUTINES.

C     THESE ROUTINES CONSIST OF A PREPROCESSOR, FACTORIZATION AND
C     FORWARD-BACKWARD SOLUTION SUBROUTINES FOR SOLVING A MATRIX
C     STORED IN COMPACT FORM USING A SKYLINE OR PROFILE STORAGE SCHEME.
C     THE MATRICES ARE ASSUMED NONSYMMETRIC BOTH IN VALUES AND STRUCTURE.

C     THERE IS ONLY ONE REQUIREMENT FOR THE COLUMN ORDERING IN THE
C     SP AND ISP ARRAYS FOR THE FACTORIZATION PREPROCESSORS.  THAT IS
C     THAT THE FIRST COLUMN FOR A GIVEN ROW IS IN THE FIRST LOCATION
C     IN THE SP AND ISP ARRAYS FOR THAT ROW (NAMELY NDROW(IR-1) + 1).
C     THE REST OF THE ORDERING DOES NOT MATTER FOR CREATING THE KDIAG
C     ARRAY AND INITIALIZING THE MATRIX A IN SKYLINE MODE.

      SUBROUTINE FACS(A,N,KDIAG, IERR)
C     MATRIX FACTORIZATION SINGLE.
C
C     THIS ROUTINE FACTORS THE LU MATRIX STORED IN SKYLINE FORMAT.
C
C     INPUT:
C         A   COEFICIENT MATRIX IN NONSYMMETRIC SKYLINE
C         STORAGE - MUST BE DIMENSIONED TO (KDIAG(2,N));
C         N   ORDER OF THE MATRIX;
C     KDIAG   COEFFICIENT POINTER - DIMENSIONED TO (2,N);
C         KDIAG(1,I) POINTS TO ROW I-1, COLUMN I, UNLESS COLUMN IS
C         EMPTY IN WHICH CASE IT POINTS TO ROW I-1, COLUMN I-1.
C         KDIAG(2,I) POINTS TO ROW I, COLUMN I.
C         MUST HAVE KDIAG(1,1)=0,
C         MUST HAVE KDIAG(2,1)=1.
C     OUTPUT:
C         A   FACTORED COEFICIENT MATRIX IN NONSYMMETRIC SKYLINE
C         STORAGE - MUST BE DIMENSIONED TO (KDIAG(2,N)).
C        IERR IS THE ERROR CODE.
C         = 0 SUCCESSFUL.
C         NOT ZERO:  A ZERO DIAGONAL IS CALCULATED AT ROW= IERR.
C     A IS SINGLE PRECISION.
C
      REAL A(1)
      INTEGER N,KDIAG(2,N),IERR

*&IF VAX
      REAL DSCCSS
      INTEGER I,KIR,LR,KIC,LC,LAST,J,KJ,LBAR,KICPJ,KIRPJ
      REAL SUM
```

```
*&ELSE
      REAL SDOT
      INTEGER I,KIR,LR,KIC,LC,J,KJ,LBAR,KICPJ,KIRPJ,K,LA1,LA2
*&END IF


C
C     REPLACE COLUMN I ABOVE DIAGONAL AND ROW I LEFT OF AND INCLIDING
C       (I,I) BY UPPER AND LOWER FACTOR RESPECTIVELY.  FIRST COLUMN,
C       THEN ROW.
C

C       ROW 1 IS UNCHANGED
        DO 100 I=2,N
C         POINT TO ROW I, COLUMN 0
          KIR=KDIAG(2,I)-I

C         FIRST STORED COLUMN OF ROW I
          LR=KDIAG(1,I)-KIR+1

C         POINTER TO ROW 0, COLUMN I
          KIC=KDIAG(1,I)-I+1

C         FIRST STORED ROW OF COLUMN I
          LC=KDIAG(2,I-1)-KIC+1

*&IF VAX
        LAST = LC
*&END IF

C         CALCULATE COLUMN I OF UPPER DIAGONAL MATRIX U
          DO 10 J=LC,I-1
C           POINT TO ROW J, COLUMN 0
            KJ=KDIAG(2,J)-J
C           FIRST COLUMN AND ROW OF DOT PRODUCT
            LBAR=MAXO(KDIAG(1,J)-KJ+1,LC)

          KICPJ = KIC + J

*&IF CRAY

          A(KICPJ) = ( A(KICPJ) -
     1   SDOT(J-LBAR,A(LBAR+KIC),1,A(LBAR+KJ),1) )/A(KDIAG(2,J))

*&ELSE
          SUM = A(KICPJ)
          A(KICPJ) = 0.

C         CHECK IF LBAR IS GREATER THAN COLUMN WITH LAST NON-ZERO RESULT.
          IF(LAST.GE.LBAR)
     1        SUM = SUM - DSCCSS(A(LBAR+KIC),A(LBAR+KJ),J-LBAR)

          IF(SUM.NE.0.)THEN
            A(KICPJ) = SUM/A(KDIAG(2,J))
            LAST = J
          END IF
*&END IF

10      CONTINUE

*&IF VAX
        LAST = LR
*&END IF
C         CALCULATE ROW I OF LOWER DIAGONAL MATRIX L, INCLUDING L(I,I)
          DO 20 J=LR+1,I
C           POINT TO ROW 0, COLUMN J
```

```
            KJ=KDIAG(1,J)-J+1
C           FIRST ROW AND COLUMN FOR DOT PRODUCT
            LBAR=MAX0(KDIAG(2,J-1)-KJ+1,LR)

          KIRPJ = KIR + J

*&IF CRAY

          A(KIRPJ) = A(KIRPJ) -
     1   SDOT(J-LBAR,A(LBAR+KIR),1,A(LBAR+KJ),1)

*&ELSE
          SUM = A(KIRPJ)
          A(KIRPJ) = 0.

C         CHECK IF LBAR IS GREATER THAN COLUMN WITH LAST NON-ZERO RESULT.
          IF(LAST.GE.LBAR)
     1    SUM = SUM - DSCCSS(A(LBAR+KIR),A(LBAR+KJ),J-LBAR)

          IF(SUM.NE.0.)THEN
            A(KIRPJ) = SUM
            LAST = J
          ELSE
C           CHECK THAT DIAGONAL IS NOT 0.
            IF(J.EQ.I)THEN
              IERR = J
          RETURN
            END IF
          END IF
*&END IF

20      CONTINUE

100   CONTINUE

      RETURN
      END

      SUBROUTINE SOLS(A,N,KDIAG,B)
C     MATRIX FORWARD-BACKWARD SUBSTITION SOLUTION, SINGLE.
C
C     THIS PERFORMS THE FOWARD REDUCTION AND BACK SUBSTITUTION TO SOLVE
C     THE MATRIX EQUATION WITH THE RIGHT HAND SIDE B.  THE SOLUTION
C     VECTOR IS RETURNED IN B, AND A IS NOT MODIFIED.  THEREFORE
C     MULTIPLE CALLS CAN BE MADE TO THIS ROUTINE FOR MULTIPLE RIGHT
C     HAND SIDES.
C     INPUT:
C         A   FACTORED COEFICIENT MATRIX IN NONSYMMETRIC SKYLINE
C         STORAGE - MUST BE DIMENSIONED TO (KDIAG(2,N)).  THE
C         FACTORIZATION MUST HAVE BEEN SUCCESSFUL BECAUSE NO CHECK
C         FOR A ZERO IN THE DIAGONAL WILL BE MADE;
C         N   ORDER OF THE MATRIX;
C     KDIAG   COEFFICIENT POINTER - DIMENSIONED TO (2,N);
C         B   RIGHT HAND SIDE VECTOR - DIMENSIONED TO (N).
C     OUTPUT:
C         B   SOLUTION VECTOR - DIMENSIONED TO (N).
C     A AND B ARE SINGLE PRECISION.
C
      REAL A(1)
      INTEGER N,KDIAG(2,1)
      REAL B(1)

*&IF VAX
      REAL DSCCSS
      INTEGER LAST,I,KI,L
      REAL SUM
      LAST = 0
*&ELSE
      REAL SDOT
      INTEGER I,KI,L
*&END IF

C       CALCULATE LINV B
        DO 10 I=1,N
C         INDEX OF ROW I COLUMN 0
          KI=KDIAG(2,I)-I
C         FIRST STORED COLUMN OF ROW I
          L=KDIAG(1,I)-KI+1

*&IF CRAY

          B(I) = ( B(I) - SDOT(I-L,A(KI+L),1,B(L),1) )/A(KDIAG(2,I))

*&ELSE
        SUM = B(I)
        B(I) = 0.

C       CHECK IF L IS GREATER THAN COLUMN WITH LAST NON-ZERO RESULT.
        IF(LAST.GE.L)SUM = SUM - DSCCSS(A(KI+L),B(L),I-L)

        IF(SUM.NE.0.)THEN
          B(I) = SUM/A(KDIAG(2,I))
          LAST = I
        END IF
*&END IF

10    CONTINUE

C       CALCULATE UINV B
      DO 20 I=N,2,-1
          KI=KDIAG(1,I)-I+1
          L=KDIAG(2,I-1)-KI+1
C         GAUSSIAN BACK SUBSTITUTION
*&IF CRAY

          IF(I.GT.L .AND.B(I).NE.0.)THEN
            CALL SAXPY(I-L, -B(I), A(KI+L),1, B(L),1)
          END IF
20        CONTINUE

*&ELSE

20      IF(I.GT.L .AND. B(I).NE.0.)CALL VPIVS(B(L),A(KI+L),-B(I),I-L)

*&END IF

      RETURN
      END
C     **********************************************************************
C
C     FROM SRT.FOR
C
C     **********************************************************************
C             SORT ROUTINES.

C           THESE ARE THE MATRIX SORT ROUTINES.

      SUBROUTINE CASRTS(AR,ICR,NC,IR, SP,ISP,NDROW, IS)
C     COLUMN ADD SORT, SINGLE PRECISION.
C
C     THIS ROUTINE IS SIMILAR TO CSRTS EXCEPT MORE THAN ONE VALUE IN THE
```

```
C      AR ARRAY CAN HAVE THE SAME COLUMN NUMBER.  THESE MULTIPLE ENTRIES
C      FOR THE SAME COLUMN ARE SUMMED, AND THEN APPENDED TO THE SP AND ISP
C      ARRAYS.  THIS ROUTINE IS ESPECIALLY USEFUL IF THE CHAIN RULE IS
C      USED TO CALCULATE THE INFLUENCE COEFFICIENTS AND THE COEFFICIENT IS
C      A COLLECTION OF SUMS.
C      INPUT:
C          AR  ARRAY OF NON-ZERO COEFFICIENTS FOR ROW IR OF THE
C          MATRIX.  ZERO COEFFICIENTS MAY BE INCLUDED FOR
C          CONVENIENCE WHICH ARE DELETED FROM THE MATRIX
C          STRUCTURE IF NC IS POSITIVE.  DIMENSIONED TO THE
C          ABSOLUTE VALUE OF NC.
C          ICR COLUMN NUMBERS OF EACH ELEMENT OF AR.
C          DIMENSIONED TO THE ABSOLUTE VALUE OF NC.
C          NC  THE ABSOLUTE VALUE IS THE NUMBER OF ENTRIES
C          IN AR & ICR FOR THIS ROW.  IF NC IS POSITIVE, ZERO
C          ENTRIES IN AR ARE DELETED. IF IT IS NEGATIVE, THESE
C          ARE RETAINED TO KEEP THE STRUCTURE OF THE MATRIX.
C          IR  ROW NUMBER OF THE PARTICULAR ROW.
C      INPUT & OUTPUT:
C          SP, ISP, AND NDROW WHICH IS THE MATRIX IN COMPACT STORAGE.
C          ON INPUT IT CONTAINS THE FIRST IR-1 ROWS OF THE MATRIX,
C          AND ON OUTPUT, IT IS THE FIRST IR ROWS OF THE MATRIX.
C          SP AND ISP MUST BE DIMENSIONED LARGE ENOUGH TO ACCOMODATE
C          THE ABS(NC) NEW VALUES IN THIS ROW.
C      WORKING STORAGE:
C          IS  USED AS THE SORTED COLUMN ARRAY - MUST BE DIMENSIONED
C          TO AT LEAST ABS(NC).
C
C      AR AND SP IS IN SINGLE PRECISION.
C
       REAL AR(1)
       INTEGER ICR(1),NC,IR
       REAL SP(1)
       INTEGER ISP(1),NDROW(1),IS(1)

       INTEGER NNC,IP,I,ISN
       REAL COEF

       NNC = IABS(NC)

C      SORT THE ICR VECTOR BY COLUMNS FOR ENTRIES 1 TO NC.
       CALL ISRTI(ICR,IS,NNC)

C      PUT THE COEFICIENTS AND COLUMNS INTO THE SP AND ISP ARRAY.
       IF(IR.EQ.1)THEN
         IP = 0
       ELSE
         IP = NDROW(IR-1)
       END IF

       IF(NC.GE.0)THEN
C          Discard 0 coefficient entries.
           I = 1
10         CONTINUE
           ISN = IS(I)
           COEF = AR(ISN)
20         CONTINUE
           IF(I.LT.NNC)GOTO 25
C            Last value of I, I = NNC.
             IF(COEF.NE.0.)THEN
               IP = IP + 1
               SP(IP) = COEF
               ISP(IP) = ICR(ISN)
             END IF
C            UPDATE NDROW.
             NDROW(IR) = IP
```

```
          RETURN

25        CONTINUE
C         CHECK FOR REPEATED COLUMN NUMBER AND SUM IF FOUND.
          IF(ICR(IS(I+1)) .EQ. ICR(ISN))THEN
            I = I+1
            ISN = IS(I)
            COEF = COEF + AR(ISN)
            GOTO 20
          END IF
C         DISCARD ZERO ENTRIES.
          IF(COEF.NE.0.)THEN
            IP = IP + 1
            SP(IP) = COEF
            ISP(IP) = ICR(ISN)
          END IF
          I = I+1
          GOTO 10

      ELSE
C       Save each coefficient.
        I = 1
30      CONTINUE
        ISN = IS(I)
        COEF = AR(ISN)
40      CONTINUE
        IF(I.LT.NNC)GOTO 45
C         Last value of I, I=NNC.
          IP = IP + 1
          SP(IP) = COEF
          ISP(IP) = ICR(ISN)
C         UPDATE NDROW.
          NDROW(IR) = IP
          RETURN

45        CONTINUE
C         CHECK FOR REPEATED COLUMN NUMBER AND SUM IF FOUND.
          IF(ICR(IS(I+1)) .EQ. ICR(ISN))THEN
            I = I+1
            ISN = IS(I)
            COEF = COEF + AR(ISN)
            GOTO 40
          END IF
          IP = IP + 1
          SP(IP) = COEF
          ISP(IP) = ICR(ISN)
          I = I+1
          GOTO 30

      END IF

      END

    SUBROUTINE ADCOFS(COF,ICOL,COFA,ICOLA,NA)
C   ADD COEFFICIENTS, SINGLE
C
C   THIS ROUTINE APPENDS THE COEFFICIENT TO THE ARRAY OF COEFFICIENTS.
C   NA IS THE NUMBER OF COEFFICIENTS IN THE ARRAYS BOTH ON INPUT AND OUTPUT.
C   IF COF = 0, NOTHING IS DONE.
C   COF AND COFA ARE SINGLE PRECISION.
C
    REAL COF,COFA(1)
    INTEGER ICOL,ICOLA(1),NA

    INTEGER I
```

```
      IF(COF.EQ.O.)RETURN

      NA = NA + 1
      COFA(NA) = COF
      ICOLA(NA) = ICOL

      RETURN
      END

      SUBROUTINE PTIME(PT)
C     This routine returns a real number representing processor time.
      REAL PT

      COMMON /SAVVAL/IENT
      DATA IENT/0/

*&IF VAX
C   UNDER VMS 4.1 YOU MUST CALL LIB$INIT_TIMER

      INTEGER*4 LIB$INIT_TIMER,LIB$STAT_TIMER

      IF(IENT.EQ.0) THEN
        CALL LIB$INIT_TIMER()
        IENT=1
      ENDIF

      CALL LIB$STAT_TIMER(2,NT)
      PT=FLOAT(NT)*0.01
*&ELSE
      CALL SECOND(PT)
*&END IF

      RETURN
      END
```

## FORTRAN Files

The following files are the rest of the code used for the solver except, the GMRES code which is available from Boeing and described in references [8] and [9], the SPARSPAK code which is available from the University of Waterloo [12], the Reverse Cuthill-Mckee code which is listed in reference [23], and the MA21A code which is listed in reference [20]. These routines can run on any machine by using a preprocessor which recognizes INCLUDE statements and DO–END DO statements.

### Main Routine File TARA.FOR

```
C             TARA (Turner's Aerodynamic Research Algorithm)

        PROGRAM TARA

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'MESSAG.FOR'

        LOGICAL ISUPW
        INTEGER ISF

C       ITER is set in input file.

        CALL INIT

C       Stream function and pressure are solved simultaneously.
        USEENT = .FALSE.

C       Set POLD to PS which is used in RHSSQ.
        DO ISf=1,NSF
          POLD(ISF) = PS(ISF)
        END DO

C       Get current right hand side and function value.
        CALL RHSSQ(-1.,PNS,BSQ)
        CALL RHSSQ(0.,PNS,BSQ)

        IF(MSGLVO.EQ.1)THEN
          PRINT*,' '
          PRINT*,'ITER, SQRT(BSQ/NSV) ',ITER,' ',SQRT(BSQ/NSV)
        END IF

        CALL PRSTAT

C       Output the results because flow has been initialized.
        CALL OUT

C       Check if there is a kinked grid.  If so, get new grid, and new right
C       hand side and reinitialize some of the other variables.  Do not do
C       duct or design option.
        IF(KLE.GT.1 .AND. KLE.NE.KTE)CALL CHKGRD

        MAXSTP = 1000.*SQRT(FLOAT(NSV))

C       Order the equations.
        CALL ORDEQ

10      CONTINUE

C       Check that solution is converged enough to have the leading edge
C       move.
        IF(BSQ/NSV.LE.EPSLE .AND. THINLE .AND. (.NOT.THNLES))THEN
          PRINT*,' '
          PRINT*,'Moving leading edge now being allowed.'
          PRINT*,' '
          THINLE = THNLES

C         Re-Order the equations.
          CALL ORDEQ

C         Re-evaluate the right hand side.
          CALL RHSSQ(-1.,PNS,BSQ)
          CALL RHSSQ(0.,PNS,BSQ)

          IF(MSGLVO.EQ.1)THEN
            PRINT*,' '
            PRINT*,'ITER, SQRT(BSQ/NSV) ',ITER,' ',SQRT(BSQ/NSV)
          END IF

          CALL PRSTAT
        END IF
        IF(BSQ/NSV.GT.EPS .AND. ITER.LT.MAXIT)THEN

          ITER = ITER + 1
          CALL ONEIT

C         Output the results every iteration.
          CALL OUT

C         Check if dsble cumulative is too large or if there is a kinked
C         grid.  If so, get new grid, and new right hand side and
C         reinitialize some of the other variables.  Do not do duct or design
C         option.
          IF(KLE.GT.1 .AND. KLE.NE.KTE)CALL CHKGRD

          GOTO 10

        END IF

C       Close the output file
        CLOSE(12)

        STOP
        END

        SUBROUTINE CHKUPW(ISUPW)
C       This routine checks if any value of MSS > MSC, the critical
C       Mach number which turns on upwinding.  If yes, ISUPW = .true.,
C       if not, ISUPW = .false.
        LOGICAL ISUPW

        INCLUDE '3DCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER ISF

        ISUPW = .FALSE.
        DO ISF=1,NSF
          IF(MSS(ISF).GE.MCS)THEN
            ISUPW = .TRUE.
            RETURN
          END IF
        END DO

        RETURN
        END

        SUBROUTINE CHKGRD
C       This routine makes sure the cumulative arc length for each spanwise
C       station has not exceeded the average upstream value or that the
C       grid is not too kinked.  If so, it is sent through the elliptic
C       grid generator again.  If any grid is changed, the momentum equation
C       multipliers will also be updated.  Also the first interval in front
C       of the stagnation point is checked that it has not stretched too
C       big or too small.

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'MESSAG.FOR'

        INTEGER I,J,K,N,N1,N2
        LOGICAL MODGRD,MOD2D
        REAL DSUP,MAXCP,DS1

        MODGRD = .FALSE.

C       Check each spanwise station.
```

```
      DO J=1,NJ
        MOD2D = .FALSE.

C         Check leading edge movement.
        IF(.NOT. THINLE)THEN
          DSUP = SBLD(NPBLD(J),J)*(SGL(2,J) + SGU(2,J))/4.
          IF(ABS(DSBLEC(J)).GT.DSUP)THEN
            MOD2D = .TRUE.
            PRINT*,'Ds average at le =',DSUP,
     1        'Dsble cumulative=',DSBLEC(J)
          END IF
        END IF

C         Check the first interval of the stagnation point.
        N1 = (KLE-1)*NI*NJ + (J-1)*NI + 1
        N2 = N1 - NI*NJ
        DS1 = SQRT( (X1N(N1)-X1N(N2))**2 + (EMPRM(N1)-EMPRM(N2))**2 )
        IF(DS1.GT.TLEMAX*DSFISP(J) .OR. DS1.LT.TLEMIN*DSFISP(J))THEN
C           Fix the inlet stagnation streamline.
          PRINT*,'Adjusting inlet stagnation streamline.'
          PRINT*,'First delta s desired = ',DSFISP(J)
          PRINT*,'First delta s on stagnation sl = ',DS1
          CALL STGFIX(J)
C           Fix the second station m' values and adjust the theta.
          K = 2
          N1 = (K-1)*NI*NJ + (J-1)*NI + 1
          DO I=2,NI-1
            N = (K-1)*NI*NJ + (J-1)*NI + I
            EMPRM(N) = EMPRM(N1)
            X1N(N) = X1N(N1) +
     1        (X1POS(I) - X1POS(1))*(X1N(N1+NI-1) - X1N(N1))/
     1        (X1POS(NI)-X1POS(1))
          END DO
          MOD2D = .TRUE.
        END IF

C         Test for kinked grid.  Too kinked if cp>0.8.
        IF(.NOT. MOD2D)THEN
          CALL CHKINK(J,MAXCP)
          IF(MAXCP.GT.0.8)THEN
            MOD2D = .TRUE.
            PRINT*,'Max cross prod=',MAXCP
          END IF
        END IF

C         Modify grid if needed.
        IF(MOD2D)THEN
          PRINT*,'Grid being regenerated at j=',J
          MODGRD = .TRUE.
          CALL ELLIP(NI,NJ,NK,J,X1N,EMPRM,X1POS,X3POS,ANG1(J),MSGLVE)

C           Calculate x2 and x3 from m' and table.
          DO K=1,NK
            DO I=1,NI
              N = (K-1)*NI*NJ + (J-1)*NI + I
              CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1                EMPRM(N),X2N(N),1,0)
              CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1                EMPRM(N),X3N(N),1,0)
            END DO
          END DO

          DSBLEC(J) = 0.
        END IF
      END DO
```

```
        IF(MODGRD)THEN
C         Set up new volume unit vectors and calculate a new right hand side.
          CALL GETVUV
          CALL RHSSQ(-1.,PNS,BSQ)
          CALL RHSSQ(0.,PNS,BSQ)

          IF(MSGLVO.EQ.1)THEN
            PRINT*,' '
            PRINT*,'ITER, SQRT(BSQ/NSV) ',ITER,' ',SQRT(BSQ/NSV)
          END IF

C         Output file with new geometry.
          CALL OUT

        END IF

        RETURN
        END



      SUBROUTINE CHKINK(J,MAXCP)
C     This routine was taken from UPDATE.FOR of ISES and modified for 3D.

C....................................
C
C     Returns maximum cross-product
C     of two adjacent grid segments
C     as a measure of grid distortion.
C....................................
      INTEGER J
      REAL MAXCP

      INCLUDE '3DCOM.FOR'

      INTEGER I,K,N,NIM,NIP,NKM,NKP
      REAL DX1KM,DMPKM,DX1KP,DMPKP
      REAL DX1IM,DMPIM,DX1IP,DMPIP
      REAL CPKSQ,CPISQ

C
      MAXCP = 0.
      DO 10 I=2, NI-1
        DO 110 K=2, NK-1
          N = (K-1)*NI*NJ + (J-1)*NI + I
          NIP = N + 1
          NIM = N - 1
          NKP = N + NI*NJ
          NKM = N - NI*NJ
          DX1KM = XN(N) - X1N(NKM)
          DMPKM = EMPRM(N) - EMPRM(NKM)
          DX1KP = XN(N) - XN(NKP)
          DMPKP = EMPRM(N) - EMPRM(NKP)
          CPKSQ = (DX1KM*DMPKP - DMPKM*DX1KP)**2
     1          / ((DX1KM**2 + DMPKM**2)*(DX1KP**2 + DMPKP**2))
          DX1IM = X1N(N) - X1N(NIM)
          DMPIM = EMPRM(N) - EMPRM(NIM)
          DX1IP = X1N(N) - X1N(NIP)
          DMPIP = EMPRM(N) - EMPRM(NIP)
          CPISQ = (DX1IM*DMPIP - DMPIM*DX1IP)**2
     1          / ((DX1IM**2 + DMPIM**2)*(DX1IP**2 + DMPIP**2))
          MAXCP = AMAX1( MAXCP , CPKSQ , CPISQ )
  110   CONTINUE
   10 CONTINUE
C
      MAXCP = SQRT(MAXCP)
      RETURN
      END

      SUBROUTINE STGFIX(J)
C     This routine was taken from UPDATE.FOR of ISES and modified for 3D.
C     It adjusts the inlet stagnation streamline for station J so that the
C     first arc length is fixed.
      INTEGER J

      INCLUDE '3DCOM.FOR'

      REAL SLEN,SNEW(MNBGP),X1OLD,MPOLD
      REAL DX1P,DMPP,DX1M,DMPM,SOLD,DX1,DMP,DELTS
      INTEGER K,N1,N2

C     Keep the first two stations the same.

C
C---- calculate the new inlet stagnation streamline arc length.
      SLEN = 0.
      DO 10 K=3, KLE
        N1 = (K-1)*NI*NJ + (J-1)*NI + 1
        N2 = N1 - NI*NJ
        SLEN = SLEN + SQRT( (X1N(N1)-X1N(N2))**2 +
        1        (EMPRM(N1)-EMPRM(N2))**2 )
   10 CONTINUE

C     Get the new exponential stretching by fixing the first interval length.
C     SNEW will start at le and go upstream.
      CALL SETEXP(SNEW,KLE-1,DSFISP(J),SLEN)
C
C---- sweep upstream from LE
      N1 = (KLE-1)*NI*NJ + (J-1)*NI + 1
      X1OLD = X1N(N1)
      MPOLD = EMPRM(N1)
      SOLD = 0.
      DO 20 K=KLE-1, 3, -1
        N1 = (K-1)*NI*NJ + (J-1)*NI + 1
        N2 = N1 - NI*NJ
C
C------ recalculate local arc length
        DX1P = X1OLD - X1N(N1)
        DMPP = MPOLD - EMPRM(N1)
        DX1M = X1N(N1) - X1N(N2)
        DMPM = EMPRM(N1) - EMPRM(N2)
C
        SOLD = SOLD + SQRT(DX1P*DX1P + DMPP*DMPP)
C
C------ distance point must move to match SNEW.  The sign is
C       different than expected because S measured from le.
        DELTS = SOLD - SNEW(KLE-K+1)
C
C------ move point along streamline tangent
        DX1 = DX1M + DX1P
        DMP = DMPM + DMPP
        X1OLD = X1N(N1)
        MPOLD = EMPRM(N1)
        X1N(N1) = X1N(N1) + DELTS * DX1/SQRT(DX1*DX1 + DMP*DMP)
        EMPRM(N1) = EMPRM(N1) + DELTS * DMP/SQRT(DX1*DX1 + DMP*DMP)
        X1N(N1 + NI-1) = X1N(N1) + PITCH
        EMPRM(N1 + NI-1) = EMPRM(N1)
   20 CONTINUE
C

      RETURN
      END
```

## File BOUND.FOR

```
      SUBROUTINE RBPSI1(I1E,PSI1V)
C     This routine calculates the residual for the boundary condition
C     which sets psi1 = psi1v at edge I1E.
      INTEGER I1E
      REAL PSI1V

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = PSI1(I1E) - PSI1V

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LBPSI1(I1E,PSI1V)
C     This routine calculates the left hand side for the boundary condition
C     which sets psi1 = psi1v at edge I1E.
      INTEGER I1E
      REAL PSI1V

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSI10+I1E,COFA,ICOLA,NA)

      RETURN
      END

      SUBROUTINE RBPSI2(I2E,PSI2V)
C     This routine calculates the residual for the boundary condition
C     which sets psi2 = psi2v at edge I2E.
      INTEGER I2E
      REAL PSI2V

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = PSI2(I2E) - PSI2V

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LBPSI2(I2E,PSI2V)
C     This routine calculates the left hand side for the boundary condition
C     which sets psi2 = psi2v at edge I2E.
      INTEGER I2E
      REAL PSI2V
      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSI20+I2E,COFA,ICOLA,NA)

      RETURN
      END


      SUBROUTINE RBDP1(I1E1,I1E2)
C     This routine calculates the residual for the boundary condition
C     which sets delta psi1 = 0 for edges i1e1 and i1e2.
      INTEGER I1E1,I1E2

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = PSI1(I1E1) - PSI1(I1E2)

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LBDP1(I1E1,I1E2)
C     This routine calculates the residual for the boundary condition
C     which sets delta psi1 = 0 for edges i1e1 and i1e2.
      INTEGER I1E1,I1E2

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSI10+I1E1,COFA,ICOLA,NA)

C     Coefficient for this equation is -1.
      CALL ADCOFS(-1.,IPSI10+I1E2,COFA,ICOLA,NA)

      RETURN
      END

      SUBROUTINE RBDP2(I2E1,I2E2)
C     This routine calculates the residual for the boundary condition
C     which sets delta psi2 = 0 for edges i2e1 and i2e2.
      INTEGER I2E1,I2E2

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = PSI2(I2E1) - PSI2(I2E2)

      FVEC(IR) = -RES
      RETURN
      END

      SUBROUTINE LBDP2(I2E1,I2E2)
C     This routine calculates the residual for the boundary condition
C     which sets delta psi2 = 0 for edges i2e1 and i2e2.
      INTEGER I2E1,I2E2

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSI20+I2E1,COFA,ICOLA,NA)

C     Coefficient for this equation is -1.
      CALL ADCOFS(-1.,IPSI20+I2E2,COFA,ICOLA,NA)

      RETURN
      END

      SUBROUTINE RBDPS(ISF1B,ISF2B)
C     This routine calculates the residual for the boundary condition
C     which sets delta Ps = 0 for faces ISF1B and ISF2B.
      INTEGER ISF1B,ISF2B

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = PS(ISF2B) - PS(ISF1B)

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LBDPS(ISF1B,ISF2B)
C     This routine calculates the left hand side for the boundary condition
C     which sets delta Ps = 0 for faces ISF1B and ISF2B.
      INTEGER ISF1B,ISF2B

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSO+ISF2B,COFA,ICOLA,NA)

C     Coefficient for this equation is -1.
      CALL ADCOFS(-1.,IPSO+ISF1B,COFA,ICOLA,NA)

      RETURN
      END

      SUBROUTINE RDPA(IAFI1,IAFNI,DPA)
C     This routine calculates the residual for the boundary condition
C     which sets Pa(IAFNI) - Pa(IAFI1) - DPA.
      INTEGER IAFI1,IAFNI
      REAL DPA

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = PA(IAFNI) - PA(IAFI1) - DPA

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LDPA(IAFI1,IAFNI,DPA)
C     This routine calculates the left hand side for the boundary condition
C     which sets Pa(IAFNI) - Pa(IAFI1) - DPA.
      INTEGER IAFI1,IAFNI
      REAL DPA

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for Pa(IAFNI) is 1.
      CALL ADCOFS(1.,IPAO+IAFNI,COFA,ICOLA,NA)

C     Coefficient for Pa(IAFI1) is -1.
      CALL ADCOFS(-1.,IPAO+IAFI1,COFA,ICOLA,NA)

      RETURN
      END

      SUBROUTINE RDTHO(IDTH)
C     This routine calculates the residual for the boundary condition
C     which sets DTH = 0 at edge IDTH.
      INTEGER IDTH

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = 0.

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LDTHO(IDTH)
C     This routine calculates the left hand side for the boundary condition
C     which sets DTH = 0 at edge IDTH.
      INTEGER IDTH

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1.
```

```
      CALL ADCOFS(1.,IDTHO+IDTH,COFA,ICOLA,NA)

      RETURN
      END

      SUBROUTINE RDTHUP(J,TANGJ,TANGJP)
C     This routine calculates the residual for the boundary condition
C     which sets up the Delta theta angle boundary condition upstream
C     at J.  TANGJ and TANGJP are the tangent of the angle boundary
C     condition for grid lines J and J+1.

      INTEGER J
      REAL TANGJ,TANGJP

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER NNA,NNB,NNC,NND
      REAL RESJ,RESJP,RES
      REAL DX1,DX3,DMP

      NNA = (J-1)*NI + 1
      NNB = NNA + NI
      NNC = NNA + NI*NJ
      NND = NNC + NI

      IF(CYLIND)THEN
        RESJ = (X1N(NNC) - X1N(NNA)) -
     1        TANGJ*(EMPRM(NNC) - EMPRM(NNA))
        RESJP = (X1N(NND) - X1N(NNB)) -
     1        TANGJP*(EMPRM(NND) - EMPRM(NNB))
      ELSE
        RESJ = (X1N(NNC) - X1N(NNA)) -
     1        TANGJ*(X3N(NNC) - X3N(NNA))
        RESJP = (X1N(NND) - X1N(NNB)) -
     1        TANGJP*(X3N(NND) - X3N(NNB))
      END IF

      RES = RESJ + RESJP

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LDTHUP(J,TANGJ,TANGJP)
C     This routine calculates the left side for the boundary condition
C     which sets up the Delta theta angle boundary condition upstream
C     at J.  TANGJ and TANGJP are the tangent of the angle boundary
C     condition for grid lines J and J+1.

      INTEGER J
      REAL TANGJ,TANGJP

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER NNA,NNB,NNC,NND

      NA = 0

      NNA = (J-1)*NI + 1
      NNB = NNA + NI
      NNC = NNA + NI*NJ
      NND = NNC + NI

      IF(CYLIND)THEN
C       Variation at node A.
        CALL LGNODE(1,1,J,1,NNA, -1., TANGJ, 0.)
C       Variation at node B.
        CALL LGNODE(1,1,J+1,1,NNB, -1., TANGJP, 0.)
C       Variation at node C.
        CALL LGNODE(1,1,J,2,NNC, 1., -TANGJ, 0.)
C       Variation at node D.
        CALL LGNODE(1,1,J+1,2,NND, 1., -TANGJP, 0.)
      ELSE
C       Variation at node A.
        CALL LGNODE(0,1,J,1,NNA, -1., 0., TANGJ)
C       Variation at node B.
        CALL LGNODE(0,1,J+1,1,NNB, -1., 0., TANGJP)
C       Variation at node C.
        CALL LGNODE(0,1,J,2,NNC, 1., 0., -TANGJ)
C       Variation at node D.
        CALL LGNODE(0,1,J+1,2,NND, 1., 0., -TANGJP)
      END IF

      RETURN
      END

      SUBROUTINE RCTHUP(J)
C     This routine calculates the residual for the boundary condition
C     which sets up the C_theta boundary condition upstream
C     at J.  ANG1(J) and ANG1(J+1) are updated, as are CZBAR(J) and
C     CZBAR(J+1).  This is only valid for cylindrical coordinates.

      INTEGER J

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER NNA,NNB,NNC,NND
      REAL TANGJ,TANGJP
      REAL RESJ,RESJP,RES
      REAL DX1,DX3,DMP
      REAL RJ,RJP
      REAL CZAVG
      REAL WTAVG

      NNA = (J-1)*NI + 1
      NNB = NNA + NI
      NNC = NNA + NI*NJ
      NND = NNC + NI

C     Get CZBAR at J and J+1 to calculate ANG1's.
      CALL RCZBAR(J,CZBAR(J))
      CALL RCZBAR(J+1,CZBAR(J+1))

C     Calculate ANG1 which is tan (betam).
      RJ = (X2N(NNA) + X2N(NNC))/2.
      RJP = (X2N(NNB) + X2N(NND))/2.
      ANG1(J) = COS(PHIUP(J))*(CTHUP(J) + OMG*RJ)/CZBAR(J)
      ANG1(J+1) = COS(PHIUP(J+1))*(CTHUP(J+1) + OMG*RJP)/CZBAR(J+1)

C     Calculate residual.
      CALL RWTAVG(J,WTAVG)

      RES = WTAVG -
     1      ( (CTHUP(J) + CTHUP(J+1))/2. + OMG*(RJ + RJP)/2. )
```

```
      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LCTHUP(J)
C     This routine calculates the left side for the boundary condition
C     which sets up the C_theta boundary condition upstream
C     at J.  It needs the left side of the average C_z.  It therefore
C     will use COFX and then put the coefficients into COFA.
C     This is only valid for cylindrical coordinates.

      INTEGER J

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER L

      NCX = 0

      CALL LWTAVG(J,1.)

C     Put coefficients COLX in COLA.
      NA = NCX
      DO L=1,NA
        COFA(L) = COFX(L)
        ICOLA(L) = ICOLX(L)
      END DO

      RETURN
      END

      SUBROUTINE LGNODE(IOPT,I,J,K,N,DRDX1,DRDX2,DRDX3)
C     This routine calculates the left hand side for the grid node
C     coordinates at I, J, K which is node N.  DRDX1, DRDX2, and DRDX3
C     are the derivatives of the residual wrt x1, x2, and x3 if IOPT=0.
C     And if IOPT=1, DRDX1 and DRDX2 are the derivatives of the residual
C     wrt x1 and mprime, and DRDX3 is not used.

      INTEGER IOPT,I,J,K,N
      REAL DRDX1,DRDX2,DRDX3

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL DRDS

      IF(IOPT.EQ.0)THEN
        DRDS = DRDX1*NX1G(N) + DRDX2*NX2G(N) + DRDX3*NX3G(N)
      ELSE
        DRDS = DRDX1*NX1G(N) + DRDX2*NMPG(N)
      END IF

      IF(NJ.EQ.2)THEN
C       w.r.t dth(1,k)
        CALL ADCOFS(DRDX1,IDTHO+K,COFA,ICOLA,NA)
C       w.r.t dsble(1)
        CALL ADCOFS(DRDS,ISBLEO+1,COFA,ICOLA,NA)
      ELSE IF(J.EQ.1)THEN
C       w.r.t dth(1,k)
        CALL ADCOFS(1.5*DRDX1,IDTHO+(K-1)*(NJ-1) + 1,COFA,ICOLA,NA)
C       w.r.t dth(2,k)
        CALL ADCOFS(-0.5*DRDX1,IDTHO+(K-1)*(NJ-1) + 2,COFA,ICOLA,NA)
```

```
C         w.r.t dsble(1)
          CALL ADCOFS(1.5*DRDS,ISBLEO+1,COFA,ICOLA,NA)
C         w.r.t dsble(2)
          CALL ADCOFS(-0.5*DRDS,ISBLEO+2,COFA,ICOLA,NA)
        ELSE IF(J.EQ.NJ)THEN
C         w.r.t dth(nj-1,k)
          CALL ADCOFS(1.5*DRDX1,IDTHO+(K-1)*(NJ-1) + NJ-1,COFA,ICOLA,NA)
C         w.r.t dth(nj-2,k)
          CALL ADCOFS(-0.5*DRDX1,IDTHO+(K-1)*(NJ-1) + NJ-2,COFA,ICOLA,NA)
C         w.r.t dsble(nj-1)
          CALL ADCOFS(1.5*DRDS,ISBLEO+NJ-1,COFA,ICOLA,NA)
C         w.r.t dsble(nj-2)
          CALL ADCOFS(-0.5*DRDS,ISBLEO+NJ-2,COFA,ICOLA,NA)
        ELSE
C         w.r.t dth(j,k)
          CALL ADCOFS(0.5*DRDX1,IDTHO+(K-1)*(NJ-1) + J,COFA,ICOLA,NA)
C         w.r.t dth(j-1,k)
          CALL ADCOFS(0.5*DRDX1,IDTHO+(K-1)*(NJ-1) + J-1,COFA,ICOLA,NA)
C         w.r.t dsble(j)
          CALL ADCOFS(0.5*DRDS,ISBLEO+J,COFA,ICOLA,NA)
C         w.r.t dsble(j-1)
          CALL ADCOFS(0.5*DRDS,ISBLEO+J-1,COFA,ICOLA,NA)
        END IF

        RETURN
        END

        SUBROUTINE RCZBAR(J,CZBARJ)
C       This routine calculates CZBAR at J.
        INTEGER J
        REAL CZBARJ

        INCLUDE '3DCOM.FOR'

        REAL CZAVGA,CZAVGB

        IF(NJ.EQ.2)THEN
          CALL RCZAVG(1,CZBARJ)
        ELSE IF(J.EQ.NJ)THEN
c         CALL RCZAVG(J-1,CZBARJ)
          CALL RCZAVG(J-1,CZAVGA)
          CALL RCZAVG(J-2,CZAVGB)
          CZBARJ = 1.5*CZAVGA - 0.5*CZAVGB
        ELSE IF(J.EQ.1)THEN
c         CALL RCZAVG(1,CZBARJ)
          CALL RCZAVG(1,CZAVGA)
          CALL RCZAVG(2,CZAVGB)
          CZBARJ = 1.5*CZAVGA - 0.5*CZAVGB
        ELSE
          CALL RCZAVG(J,CZAVGA)
          CALL RCZAVG(J-1,CZAVGB)
          CZBARJ = 0.5*(CZAVGA + CZAVGB)
        END IF

        RETURN
        END


      SUBROUTINE RCZAVG(J,CZAVG)
C     This routine calculates the area weighted average of C_z upstream
C     at mid grid line J.  The routine is a modified version of RENTC.
C     It is only for cylindrical cases.
C
      INTEGER J
      REAL CZAVG
      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'RENTCDEC.FOR'

      INTEGER ISF
      REAL DTHETA

C     Declare the contravarient velocities.
      REAL RWX,RWY,RWZ

C     Declare the radius and theta.
      REAL R,X1V

      CZAVG = 0.

C     Get the area weighted average.
      DO ISF=(J-1)*(NI-1)+1,J*(NI-1)

C       Get the face variables for this face.  Data passed
C       through common.
        CALL IMS(ISF)

C       Get DTHETA as average difference of 4 volume nodes.
        DTHETA = 0.5*(X1CC + X1DC - X1AC - X1BC)

C       Get the radius and theta.
        CALL RRTHFC(R,X1V)

C       Get the velocities.
        CALL RVELC(3,RWX,RWY,RWZ)

C       (rho W)^2
        RWS = RWX ** 2 + RWY ** 2 + RWZ ** 2

C       (omega r)^2/2
        OR2 = 0.5E0 * OMG ** 2 * R ** 2

C       density
      RHO = 0.5E0 * ((GAM * P + (GAM ** 2 * P ** 2 + 2 * RWS * ((-1) +
     $    GAM) ** 2 * (H + OR2)) ** 0.5E0) / (((-1) + GAM) * (H + OR2))
     $    )

        CZAVG = CZAVG + DTHETA*RWZ/RHO

      END DO

      CZAVG = CZAVG/PITCH
C
      RETURN
      END


      SUBROUTINE LCZAVG(J,DEQDCZ)
C     This routine calculates the left side of the area weighted average
C     of C_z upstream calculation at mid grid line J.  The routine is a
C     modified version of LENTC.
C     It is only for cylindrical cases.
C     This routine multiplies DEQDCZ by the C_z coefficients and adds
C     them to the COLX arrays.
C
      INTEGER J
      REAL DEQDCZ

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'
```

```
C      These declarations are generated by SMP:
       INCLUDE 'LENTCDEC.FOR'
       REAL DERI
C      Declare the contravarient velocities.
       REAL RWX,RWY,RWZ

C      Declare the radius and theta.
       REAL R,X1V

       INTEGER ISF
       REAL DTHETA,CZD1,CZD2,CZD3,CZD4

C      There won't be any variation of radius at this boundary.

       DO ISF=(J-1)*(NI-1)+1,J*(NI-1)

C        Get the face variables for this face.  Data passed
C        through common.
         CALL IMS(ISF)

C        Get DTHETA as average difference of 4 volume nodes.
         DTHETA = 0.5*(X1CC + X1DC - X1AC - X1BC)

C        Get the radius and theta.
         CALL RRTHFC(R,X1V)

C        Get the velocities.
         CALL RVELC(3,RWX,RWY,RWZ)

C        (rho W)^2
         RWS = RWX ** 2 + RWY ** 2 + RWZ ** 2

C        Derivatives
         DERI = 2 * RWX

         rwsd2 = deri
C
C        Derivatives
         DERI = 2 * RWY

         rwsd3 = deri
C
C        Derivatives
         DERI = 2 * RWZ

         rwsd4 = deri
C
C        (omega r)^2/2
         OR2 = 0.5E0 * OMG ** 2 * R ** 2

C        density
        RHO = 0.5E0 * ((GAM * P + (GAM ** 2 * P ** 2 + 2 * RWS * ((-1) +
     $    GAM) ** 2 * (H + OR2)) ** 0.5E0) / (((-1) + GAM) * (H + OR2))
     $    )

C        Derivatives
        DERI = 0.5E0 * ((GAM + (GAM ** 2 * P) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0) / (((-1) + GAM
     $    ) * (H + OR2)))

         rhod1 = deri
C
C        Derivatives
        DERI = 0.5E0 * ((RWSD2 * ((-1) + GAM)) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0)
```

```
         rhod2 = deri
C
C        Derivatives
       DERI = 0.5E0 * ((RWSD3 * ((-1) + GAM)) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0)

         rhod3 = deri
C
C        Derivatives
       DERI = 0.5E0 * ((RWSD4 * ((-1) + GAM)) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0)

         rhod4 = deri


C
C              CZAVG = CZAVG + DTHETA*RWZ/(RHO*PITCH)
         if(mss(isf).gt.0.9 .and. mss(isf).lt.1.1)then
C          At sonic, do not account for the dependence of rho on velocity.
           CZD1 = -DTHETA/PITCH*RWZ/RHO**2*RHOD1
           CZD2 = 0.
           CZD3 = 0.
           CZD4 = DTHETA/(PITCH*RHO)
         else
           CZD1 = -DTHETA/PITCH*RWZ/RHO**2*RHOD1
           CZD2 = -DTHETA/PITCH*RWZ/RHO**2*RHOD2
           CZD3 = -DTHETA/PITCH*RWZ/RHO**2*RHOD3
           CZD4 = DTHETA/(PITCH*RHO) -
     $          DTHETA/PITCH*RWZ/RHO**2*RHOD4
         end if
C
C        Put Jacobians in correct location in matrix.
C        X - Momentum Equation array
C        NCX must be set.
C
C        ps
         CALL ADCOFS(DEQDCZ*CZD1,IPSO+IP,COFX,ICOLX,NCX)
C
C        NCY and NCZ must be set.
         NCY = 0
         NCZ = 0

C        The dependence on the velocities.  Put in x-momentum eq arrays.
         CALL LVELC(3,DEQDCZ*CZD2,DEQDCZ*CZD3,DEQDCZ*CZD4,
     $               0.,0.,0., 0.,0.,0.)

        END DO


        RETURN
        END

        SUBROUTINE RWTAVG(J,WTAVG)
C       This routine calculates the area weighted average of W_theta upstream
C       at mid grid line J.  The routine is a modified version of RENTC and VELC.
C       It is only for cylindrical cases.
C
        INTEGER J
        REAL WTAVG

        INCLUDE '3DCOM.FOR'
        INCLUDE 'FACE.FOR'
C       These declarations are generated by SMP:
        INCLUDE 'RENTCDEC.FOR'

        INTEGER ISF
```

```
      REAL DTHETA

C     Declare the derivatives of psi1 and psi2.
      REAL SI11,SI12,SI13,SI21,SI22,SI23

C     Declare the velocities.
      REAL RWX,RWY,RWZ,RW1

C     Declare the radius and theta.
      REAL R,X1V

      WTAVG = 0.

C     Get the area weighted average.
      DO ISF=(J-1)*(NI-1)+1,J*(NI-1)

C       Get the face variables for this face.  Data passed
C       through common.
        CALL IMS(ISF)

CC        Get the derivatives of psi1 and psi2.
C         CALL RPDSI(3,SI11,SI12,SI13)
C         CALL RPDSI(6,SI21,SI22,SI23)
C
CC        Rho W(theta).
C         RW1 = SI12 * SI23 + (-1) * SI13 * SI22

C       Get DTHETA as average difference of 4 volume nodes.
        DTHETA = 0.5*(X1CC + X1DC - X1AC - X1BC)

C       Get the radius and theta.
        CALL RRTHFC(R,X1V)

C       Get the velocities.
        CALL RVELC(3,RWX,RWY,RWZ)

C       Get the W_theta.
        RW1 = COS(X1V)*RWX - SIN(X1V)*RWY

C       (rho W)^2
        RWS = RWX ** 2 + RWY ** 2 + RWZ ** 2

C       (omega r)^2/2
        OR2 = 0.5E0 * OMG ** 2 * R ** 2

C       density
      RHO = 0.5E0 * ((GAM * P + (GAM ** 2 * P ** 2 + 2 * RWS * ((-1) +
     $    GAM) ** 2 * (H + OR2)) ** 0.5E0) / (((-1) + GAM) * (H + OR2))
     $    )

        WTAVG = WTAVG + DTHETA/RHO*RW1

      END DO

      WTAVG = WTAVG/PITCH
C
      RETURN
      END


      SUBROUTINE LWTAVG(J,DEQDWT)
C     This routine calculates the left side of the area weighted average
C     of W_theta upstream calculation at mid grid line J.  The routine is a
C     modified version of LENTC and VELC.
C     It is only for cylindrical cases.
C     This routine multiplies DEQDWT by the W_theta coefficients and adds
```

```
C     them to the COLX arrays.
C
      INTEGER J
      REAL DEQDWT

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'
C     These declarations are generated by SMP:
      INCLUDE 'LENTCDEC.FOR'
      REAL DERI

C     Declare the derivatives of psi1 and psi2.
      REAL SI11,SI12,SI13,SI21,SI22,SI23

C     Declare the contravarient velocities.
      REAL RWX,RWY,RWZ

C     Declare the radius and theta.
      REAL R,X1V

      INTEGER ISF
      REAL DTHETA,WTD1,WTD2,WTD3,WTD4,WTD5,RW1

C     The changes in the physical derivatives.
      REAL DFXD11,DFXD12,DFXD13
      REAL DFXD21,DFXD22,DFXD23


C     There won't be any variation of radius at this boundary.

      DO ISF=(J-1)*(NI-1)+1,J*(NI-1)

C       Get the face variables for this face.  Data passed
C       through common.
        CALL IMS(ISF)

CC        Get the derivatives of psi1 and psi2.
C         CALL RPDSI(3,SI11,SI12,SI13)
C         CALL RPDSI(6,SI21,SI22,SI23)
C
CC        Rho W(theta).
C         RW1 = SI12 * SI23 + (-1) * SI13 * SI22

C       Get DTHETA as average difference of 4 volume nodes.
        DTHETA = 0.5*(X1CC + X1DC - X1AC - X1BC)

C       Get the radius and theta.
        CALL RRTHFC(R,X1V)

C       Get the velocities.
        CALL RVELC(3,RWX,RWY,RWZ)

C       Get the W_theta.
        RW1 = COS(X1V)*RWX - SIN(X1V)*RWY

C       (rho W)^2
        RWS = RWX ** 2 + RWY ** 2 + RWZ ** 2

C       Derivatives
        DERI = 2 * RWX

        rwsd2 = deri
C
C       Derivatives
        DERI = 2 * RWY
        rwsd3 = deri
C
C       Derivatives
        DERI = 2 * RWZ

        rwsd4 = deri
C
C       (omega r)^2/2
        OR2 = 0.5E0 * OMG ** 2 * R ** 2

C       density
       RHO = 0.5E0 * ((GAM * P + (GAM ** 2 * P ** 2 + 2 * RWS * ((-1) +
     $    GAM) ** 2 * (H + OR2)) ** 0.5E0) / (((-1) + GAM) * (H + OR2))
     $    )

C       Derivatives
       DERI = 0.5E0 * ((GAM + (GAM ** 2 * P) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0) / (((-1) + GAM
     $    ) * (H + OR2))

        rhod1 = deri
C
C       Derivatives
       DERI = 0.5E0 * ((RWSD2 * ((-1) + GAM)) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0)

        rhod2 = deri
C
C       Derivatives
       DERI = 0.5E0 * ((RWSD3 * ((-1) + GAM)) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0)

        rhod3 = deri
C
C       Derivatives
       DERI = 0.5E0 * ((RWSD4 * ((-1) + GAM)) / (GAM ** 2 * P ** 2 + 2 *
     $    RWS * ((-1) + GAM) ** 2 * (H + OR2)) ** 0.5E0)

        rhod4 = deri


C
C               WTAVG = WTAVG + DTHETA/(PITCH*RHO)*RW1
C         wrt P.
          WTD1 = -DTHETA/PITCH*RW1/RHO**2*RHOD1
C         wrt rho Wx.
          WTD2 = -DTHETA/PITCH*RW1/RHO**2*RHOD2
C         wrt rho Wy.
          WTD3 = -DTHETA/PITCH*RW1/RHO**2*RHOD3
C         wrt rho Wz.
          WTD4 = -DTHETA/PITCH*RW1/RHO**2*RHOD4

C         wrt rho W_theta.
          WTD5 = DTHETA/(PITCH*RHO)

C         d rho W_theta/d theta = -rho W_x sin theta - rho W_y cos theta
C         d rho W_theta/d rho W_x = cos theta
C         d rho W_theta/d rho W_y = -sin theta
          WTD2 = WTD2 + COS(X1V)*WTD5
          WTD3 = WTD3 - SIN(X1V)*WTD5

C
C        Put Jacobians in correct location in matrix.
C        X - Momentum Equation array
C        NCX must be set.
C
```

```
C       ps
        CALL ADCOFS(DEQDWT*WTD1,IPSO+IP,COFX,ICOLX,NCX)
C
C       NCY and NCZ must be set.
        NCY = 0
        NCZ = 0

C       The dependence on the velocities.  Put in x-momentum eq arrays.
        CALL LVELC(3,DEQDWT*WTD2,DEQDWT*WTD3,DEQDWT*WTD4,
     $              0.,0.,0., 0.,0.,0.)

        CALL LRTHFC(0.,DEQDWT*WTD5*(-RWX*SIN(X1V) - RWY*COS(X1V)),
     $              0.,0., 0.,0. )

CC       The dependence on the psi derivatives.
C        DFXD11 = 0.
C        DFXD12 = DEQDWT*WTD5*SI23
C        DFXD13 = -DEQDWT*WTD5*SI22
C
C        DFXD21 = 0.
C        DFXD22 = -DEQDWT*WTD5*SI13
C        DFXD23 = DEQDWT*WTD5*SI12
C
C        CALL LPDSI(3,DFXD11,DFXD12,DFXD13,
C     1             0.,0.,0., 0.,0.,0.)
C        CALL LPDSI(6,DFXD21,DFXD22,DFXD23,
C     1             0.,0.,0., 0.,0.,0.)

      END DO


      RETURN
      END


        SUBROUTINE RBENT(ISF,SDES)
C       This routine calculates the residual for the entropy boundary condition
C       which sets the entropy to Sdes for face ISF.
        INTEGER ISF
        REAL SDES

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        REAL SACT,RES

        IF(CMPRES)THEN
          CALL RENTC(ISF,SACT)
        ELSE
          CALL RENT(ISF,SACT)
        END IF

        RES = SACT - SDES

        RESX = -RES

        RETURN
        END

        SUBROUTINE LBENT(ISF)
C       This routine calculates the left hand side for the entropy
C       boundary condition for the face ISF.
        INTEGER ISF

        INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1 for the entropy.
C     The left hand side is stored in the x momentum eq.
      NCX = 0
      IF(CMPRES)THEN
        CALL LENTC(ISF,1.)
      ELSE
        CALL LENT(ISF,1.)
      END IF

      RETURN
      END

      SUBROUTINE RBPS(ISF,PSV)
C     This routine calculates the residual for the boundary condition
C     which sets ps = psv at s face ISF.
      INTEGER ISF
      REAL PSV

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      RES = PS(ISF) - PSV

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LBPS(ISF)
C     This routine calculates the left hand side for the boundary condition
C     which sets ps = psv at s face ISF.
      INTEGER ISF
      REAL PSV

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      NA = 0

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSO+ISF,COFA,ICOLA,NA)

      RETURN
      END

      SUBROUTINE RBDPS2(I,J)
C     This routine calculates the residual for the boundary condition
C     which sets d^2 Ps/dxi2 dxi3 = 0 for faces the s face i,j to i,j+1.
      INTEGER I,J

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      INTEGER K
```

```
C       Get the face indices.
        K = NK - 1
C         s1 : i = I, j = J, k = K
          ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C         s2 : s1 + (ni-1),  i = I, j = J+1, k = K
          ISF2 = ISF1 + (NI-1)
        K = NK - 2
C         s3 : i = I, j = J, k = K
          ISF3 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C         s4 : s3 + (ni-1),  i = I, j = J+1, k = K
          ISF4 = ISF3 + (NI-1)

        RES = PS(ISF2) - PS(ISF1) - PS(ISF4) + PS(ISF3)

        FVEC(IR) = -RES

        RETURN
        END

        SUBROUTINE LBDPS2(I,J)
C       This routine calculates the left hand side for the boundary condition
C       which sets d^2 Ps/dxi2 dxi3 = 0 for faces the s face i,j to i,j+1.
        INTEGER I,J

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER K

        NA = 0

C       Get the face indices.
        K = NK - 1
C         s1 : i = I, j = J, k = K
          ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C         s2 : s1 + (ni-1),  i = I, j = J+1, k = K
          ISF2 = ISF1 + (NI-1)
        K = NK - 2
C         s3 : i = I, j = J, k = K
          ISF3 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C         s4 : s3 + (ni-1),  i = I, j = J+1, k = K
          ISF4 = ISF3 + (NI-1)

C       Coefficient for this equation is 1.
        CALL ADCOFS(1.,IPSO+ISF2,COFA,ICOLA,NA)

C       Coefficient for this equation is -1.
        CALL ADCOFS(-1.,IPSO+ISF1,COFA,ICOLA,NA)

C       Coefficient for this equation is -1.
        CALL ADCOFS(-1.,IPSO+ISF4,COFA,ICOLA,NA)

C       Coefficient for this equation is 1.
        CALL ADCOFS(1.,IPSO+ISF3,COFA,ICOLA,NA)

        RETURN
        END

        SUBROUTINE RBDPS1(I,J)
C       This routine calculates the residual for the boundary condition
C       which sets d^2 Ps/dxi1 dxi3 = 0 for faces the s face i,j to i+1,j.
        INTEGER I,J

        INCLUDE '3DCOM.FOR'
```

```
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      REAL RES

      INTEGER K

C     Get the face indices.
      K = NK - 1
C       s1 : i = I, j = J, k = K
        ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C       s2 : s1 + 1,  i = I+1, j = J, k = K
        ISF2 = ISF1 + 1
      K = NK - 2
C       s3 : i = I, j = J, k = K
        ISF3 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C       s4 : s3 + 1,  i = I+1, j = J, k = K
        ISF4 = ISF3 + 1

      RES = PS(ISF2) - PS(ISF1) - PS(ISF4) + PS(ISF3)

      FVEC(IR) = -RES

      RETURN
      END

      SUBROUTINE LBDPS1(I,J)
C     This routine calculates the left hand side for the boundary condition
C     which sets d^2 Ps/dxi1 dxi3 = 0 for faces the s face i,j to i+1,j.
      INTEGER I,J

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER K

      NA = 0

C     Get the face indices.
      K = NK - 1
C       s1 : i = I, j = J, k = K
        ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C       s2 : s1 + 1,  i = I+1, j = J, k = K
        ISF2 = ISF1 + 1
      K = NK - 2
C       s3 : i = I, j = J, k = K
        ISF3 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C       s4 : s3 + 1,  i = I+1, j = J, k = K
        ISF4 = ISF3 + 1

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSO+ISF2,COFA,ICOLA,NA)

C     Coefficient for this equation is -1.
      CALL ADCOFS(-1.,IPSO+ISF1,COFA,ICOLA,NA)

C     Coefficient for this equation is -1.
      CALL ADCOFS(-1.,IPSO+ISF4,COFA,ICOLA,NA)

C     Coefficient for this equation is 1.
      CALL ADCOFS(1.,IPSO+ISF3,COFA,ICOLA,NA)

      RETURN
      END
```

## File COMBEQ.FOR

```
      SUBROUTINE COM2EQ(COF1,ICOL1,N1,RHS1,COF2,ICOL2,N2,RHS2,
     1       EQ1M,EQ2M,COFC,ICOLC,NC,RHSC)
C     This routine combines two equations stored in compact format,
C     eq1 and eq2, by adding EQ1M*eq1 and EQ2M*eq2.
C     The result is put in the combined equation, eqc, also
C     stored in compact format.
C     eq1 uses COF1,ICOL1,N1,RHS1;
C     eq2 uses COF2,ICOL2,N2,RHS2;
C     eqc uses COFC,ICOLC,NC,RHSC.
C     On input, eq1 and eq2 should be sorted, and on output, eqc will
C     be sorted.
      REAL COF1(1),RHS1,COF2(1),RHS2,COFC(1),RHSC,EQ1M,EQ2M
      INTEGER ICOL1(1),N1,ICOL2(1),N2,ICOLC(1),NC

      INTEGER I,I1,I2

C     Put equation 1 and 2 into combined equation making sure equation
C     c is ordered.

      I1 = 1
      I2 = 1
      NC = 0
      RHSC = EQ1M*RHS1 + EQ2M*RHS2

10    CONTINUE

      IF(ICOL1(I1).LT.ICOL2(I2))THEN
C       Column in eq1 is first.
        NC = NC + 1
        ICOLC(NC) = ICOL1(I1)
        COFC(NC) = EQ1M*COF1(I1)
        IF(I1.EQ.N1)GOTO 20
        I1 = I1 + 1
        GOTO 10
      ELSE IF(ICOL1(I1).GT.ICOL2(I2))THEN
C       Column in eq2 is first.
        NC = NC + 1
        ICOLC(NC) = ICOL2(I2)
        COFC(NC) = EQ2M*COF2(I2)
        IF(I2.EQ.N2)GOTO 30
        I2 = I2 + 1
        GOTO 10
      ELSE
C       Column in eq1 and eq2 are the same.
        NC = NC + 1
        ICOLC(NC) = ICOL1(I1)
        COFC(NC) = EQ1M*COF1(I1) + EQ2M*COF2(I2)
        IF(I1.EQ.N1 .AND. I2.EQ.N2)RETURN
        IF(I1.EQ.N1)THEN
          I2 = I2 + 1
          GOTO 20
        END IF
        IF(I2.EQ.N2)THEN
          I1 = I1 + 1
          GOTO 30
        END IF
        I1 = I1 + 1
        I2 = I2 + 1
        GOTO 10
      END IF
```

```
20      CONTINUE

C       Equation 1 used up.  Finish equation 2.
        DO I=I2,N2
          NC = NC + 1
          ICOLC(NC) = ICOL2(I)
          COFC(NC) = EQ2M*COF2(I)
        END DO

        RETURN

30      CONTINUE

C       Equation 2 used up.  Finish equation 1.
        DO I=I1,N1
          NC = NC + 1
          ICOLC(NC) = ICOL1(I)
          COFC(NC) = EQ1M*COF1(I)
        END DO

        RETURN

        END

        SUBROUTINE COM3EQ(COF1,ICOL1,N1,RHS1,COF2,ICOL2,N2,RHS2,
        1       COF3,ICOL3,N3,RHS3,EQ1M,EQ2M,EQ3M,
        1       COFC,ICOLC,NC,RHSC)
C       This routine combines three equations stored in compact format,
C       eq1, eq2 and eq3, by adding EQ1M*eq1 + EQ2M*eq2 + EQ3M*eq3.
C       The result is put in the combined equation, eqc, also
C       stored in compact format.
C       eq1 uses COF1,ICOL1,N1,RHS1;
C       eq2 uses COF2,ICOL2,N2,RHS2;
C       eq3 uses COF3,ICOL3,N3,RHS3;
C       eqc uses COFC,ICOLC,NC,RHSC.
C       On input, eq1, eq2 and eq3 should be sorted, and on output, eqc will
C       be sorted.
        REAL COF1(1),RHS1,COF2(1),RHS2,COF3(1),RHS3,EQ1M,EQ2M,EQ3M
        REAL COFC(1),RHSC
        INTEGER ICOL1(1),N1,ICOL2(1),N2,ICOL3(1),N3,ICOLC(1),NC

        INTEGER I,I1,I2,I3,IC1,IC2,IC3,NCA
        REAL DUM

C       Put equation 1, 2 and 3 into combined equation making sure equation
C       c is ordered.

        I1 = 1
        I2 = 1
        I3 = 1
        NC = 0
        RHSC = EQ1M*RHS1 + EQ2M*RHS2 + EQ3M*RHS3

10      CONTINUE

        IC1 = ICOL1(I1)
        IC2 = ICOL2(I2)
        IC3 = ICOL3(I3)

        IF(IC1.LT.IC2)THEN
          IF(IC1.LT.IC3)THEN
C           Column in eq1 is first.  ic1 < ic2 & ic1 < ic3
            NC = NC + 1
            ICOLC(NC) = IC1
            COFC(NC) = EQ1M*COF1(I1)
            IF(I1.EQ.N1)GOTO 20
            I1 = I1 + 1
            GOTO 10
          ELSE IF(IC1.GT.IC3)THEN
C           Column in eq3 is first.  ic3 < ic1 & ic3 < ic2
            NC = NC + 1
            ICOLC(NC) = IC3
            COFC(NC) = EQ3M*COF3(I3)
            IF(I3.EQ.N3)GOTO 40
            I3 = I3 + 1
            GOTO 10
          ELSE
C           Column 1 and 3 are the same and first.  ic1 = ic3 & ic1 < ic2
            NC = NC + 1
            ICOLC(NC) = IC1
            COFC(NC) = EQ1M*COF1(I1) + EQ3M*COF3(I3)
            IF(I1.EQ.N1 .AND. I3.EQ.N3)GOTO 60
            IF(I1.EQ.N1)THEN
              I3 = I3 + 1
              GOTO 20
            END IF
            IF(I3.EQ.N3)THEN
              I1 = I1 + 1
              GOTO 40
            END IF
            I1 = I1 + 1
            I3 = I3 + 1
            GOTO 10
          END IF
        ELSE IF(IC1.GT.IC2)THEN
          IF(IC2.LT.IC3)THEN
C           Column in eq2 is first.  ic2 < ic3 & ic2 < ic1
            NC = NC + 1
            ICOLC(NC) = IC2
            COFC(NC) = EQ2M*COF2(I2)
            IF(I2.EQ.N2)GOTO 30
            I2 = I2 + 1
            GOTO 10
          ELSE IF(IC2.GT.IC3)THEN
C           Column in eq3 is first.  ic3 < ic1 & ic3 < ic2
            NC = NC + 1
            ICOLC(NC) = IC3
            COFC(NC) = EQ3M*COF3(I3)
            IF(I3.EQ.N3)GOTO 40
            I3 = I3 + 1
            GOTO 10
          ELSE
C           Column 2 and 3 are the same and first.  ic2 = ic3 & ic2 < ic1
            NC = NC + 1
            ICOLC(NC) = IC2
            COFC(NC) = EQ2M*COF2(I2) + EQ3M*COF3(I3)
            IF(I2.EQ.N2 .AND. I3.EQ.N3)GOTO 50
            IF(I2.EQ.N2)THEN
              I3 = I3 + 1
              GOTO 30
            END IF
            IF(I3.EQ.N3)THEN
              I2 = I2 + 1
              GOTO 40
            END IF
            I2 = I2 + 1
            I3 = I3 + 1
            GOTO 10
          END IF
        ELSE
C         Column in eq1 and eq2 are the same.
          IF(IC1.LT.IC3)THEN
```

```
C           Column 1 and 2 are the same and first.  ic1 = ic2 & ic1 < ic3
            NC = NC + 1
            ICOLC(NC) = IC1
            COFC(NC) = EQ1M*COF1(I1) + EQ2M*COF2(I2)
            IF(I1.EQ.N1 .AND. I2.EQ.N2)GOTO 70
            IF(I1.EQ.N1)THEN
              I2 = I2 + 1
              GOTO 20
            END IF
            IF(I2.EQ.N2)THEN
              I1 = I1 + 1
              GOTO 30
            END IF
            I1 = I1 + 1
            I2 = I2 + 1
            GOTO 10
          ELSE IF(IC1.GT.IC3)THEN
C           Column in eq3 is first.  ic3 < ic1 & ic3 < ic2
            NC = NC + 1
            ICOLC(NC) = IC3
            COFC(NC) = EQ3M*COF3(I3)
            IF(I3.EQ.N3)GOTO 40
            I3 = I3 + 1
            GOTO 10
          ELSE
C           Column 1, 2 and 3 are the same.
            NC = NC + 1
            ICOLC(NC) = IC1
            COFC(NC) = EQ1M*COF1(I1) + EQ2M*COF2(I2) + EQ3M*COF3(I3)
            IF(I1.EQ.N1 .AND. I2.EQ.N2 .AND. I3.EQ.N3)RETURN
            IF(I2.EQ.N2 .AND. I3.EQ.N3)THEN
              I1 = I1 + 1
              GOTO 50
            END IF
            IF(I1.EQ.N1 .AND. I3.EQ.N3)THEN
              I2 = I2 + 1
              GOTO 60
            END IF
            IF(I1.EQ.N1 .AND. I2.EQ.N2)THEN
              I3 = I3 + 1
              GOTO 70
            END IF
            IF(I1.EQ.N1)THEN
              I2 = I2 + 1
              I3 = I3 + 1
              GOTO 20
            END IF
            IF(I2.EQ.N2)THEN
              I1 = I1 + 1
              I3 = I3 + 1
              GOTO 30
            END IF
            IF(I3.EQ.N3)THEN
              I1 = I1 + 1
              I2 = I2 + 1
              GOTO 40
            END IF
            I1 = I1 + 1
            I2 = I2 + 1
            I3 = I3 + 1
            GOTO 10
          END IF
        END IF

 20     CONTINUE
```

```
C       Equation 1 used up.  Finish equation 2 and 3.
        CALL COM2EQ(COF2(I2),ICOL2(I2),N2-I2+1,0.,
        1       COF3(I3),ICOL3(I3),N3-I3+1,0.,
        1       EQ2M,EQ3M,COFC(NC+1),ICOLC(NC+1),NCA,DUM)
        NC = NC + NCA

        RETURN

30      CONTINUE

C       Equation 2 used up.  Finish equation 1 and 3.
        CALL COM2EQ(COF1(I1),ICOL1(I1),N1-I1+1,0.,
        1       COF3(I3),ICOL3(I3),N3-I3+1,0.,
        1       EQ1M,EQ3M,COFC(NC+1),ICOLC(NC+1),NCA,DUM)
        NC = NC + NCA

        RETURN

40      CONTINUE

C       Equation 3 used up.  Finish equation 1 and 2.
        CALL COM2EQ(COF1(I1),ICOL1(I1),N1-I1+1,0.,
        1       COF2(I2),ICOL2(I2),N2-I2+1,0.,
        1       EQ1M,EQ2M,COFC(NC+1),ICOLC(NC+1),NCA,DUM)
        NC = NC + NCA

        RETURN

50      CONTINUE

C       Equation 2  and 3 used up.  Finish equation 1.
        DO I=I1,N1
          NC = NC + 1
          ICOLC(NC) = ICOL1(I)
          COFC(NC) = EQ1M*COF1(I)
        END DO

        RETURN

60      CONTINUE

C       Equation 1 and 3 used up.  Finish equation 2.
        DO I=I2,N2
          NC = NC + 1
          ICOLC(NC) = ICOL2(I)
          COFC(NC) = EQ2M*COF2(I)
        END DO

        RETURN

70      CONTINUE

C       Equation 1 and 2 used up.  Finish equation 3.
        DO I=I3,N3
          NC = NC + 1
          ICOLC(NC) = ICOL3(I)
          COFC(NC) = EQ3M*COF3(I)
        END DO

        RETURN

        END
```

## File ELIM.FOR

```
      SUBROUTINE ELIM(COF1,ICOL1,N1,RHS1,COF2,ICOL2,N2,RHS2,
     1       ICOLEL,COFC,ICOLC,NC,RHSC,IERR)
C     This routine uses the two equations stored in compact format,
C     eq1 and eq2, and combines them to eliminate the dependence on
C     column ICOLEL.  The result is put in combined equation, eqc, also
C     stored in compact format.
C     eq1 uses COF1,ICOL1,N1,RHS1;
C     eq2 uses COF2,ICOL2,N2,RHS2;
C     eqc uses COFC,ICOLC,NC,RHSC.
C     On input, eq1 and eq2 should be sorted, and on output, eqc will
C     be sorted.
C     IERR is the error return code;  = 0 : successful
C                                     = 1 : no coefs for column ICOLC
C                                               in either eq.
      REAL COF1(1),RHS1,COF2(1),RHS2,COFC(1),RHSC
      INTEGER ICOL1(1),N1,ICOL2(1),N2,ICOLC(1),NC,ICOLEL,IERR

      INTEGER I,I1,I2
      REAL EQ1M,EQ2M

C     Search through equation 1 to find the coeficient for column ICOLEL.
      EQ2M = 0.
      DO I=1,N1
        IF(ICOL1(I).EQ.ICOLEL)THEN
C         Multiply equation 2 by minus this coefficient.
          EQ2M = -COF1(I)
          GOTO 10
        END IF
      END DO

10    CONTINUE

C     Search through equation 2 to find the coeficient for column ICOLEL.
      EQ1M = 0.
      DO I=1,N2
        IF(ICOL2(I).EQ.ICOLEL)THEN
C         Multiply equation 1 by this coefficient.
          EQ1M = COF2(I)
          GOTO 20
        END IF
      END DO

20    CONTINUE

C     The combined right hand side.
      RHSC = EQ1M*RHS1 + EQ2M*RHS2

      IF(EQ1M.EQ.0. .AND. EQ2M.EQ.0.)THEN
        IERR = 1
        RETURN
      END IF

      IF(EQ1M.EQ.0.)THEN
C       Make the combined equation equal to equation 2 but leave out ICOLEL.
        NC = 0
        DO I=1,N2
          IF(ICOL2(I).NE.ICOLEL)THEN
            NC = NC + 1
            ICOLC(NC) = ICOL2(I)
            COFC(NC) = EQ2M*COF2(I)
          END IF
        END DO
        IERR = 0
        RETURN
        END IF

        IF(EQ2M.EQ.O.)THEN
C         Make the combined equation equal to equation 1 but leave out ICOLEL.
          NC = 0
          DO I=1,N1
            IF(ICOL1(I).NE.ICOLEL)THEN
              NC = NC + 1
              ICOLC(NC) = ICOL1(I)
              COFC(NC) = EQ1M*COF1(I)
            END IF
          END DO
          IERR = 0
          RETURN
        END IF

C       Put equation 1 and 2 into combined equation making sure equation
C       c is ordered.

        I1 = 1
        I2 = 1
        NC = 0

30      CONTINUE

        IF(ICOL1(I1).LT.ICOL2(I2))THEN
C         Column in eq1 is first.
          NC = NC + 1
          ICOLC(NC) = ICOL1(I1)
          COFC(NC) = EQ1M*COF1(I1)
          IF(I1.EQ.N1)GOTO 40
          I1 = I1 + 1
          GOTO 30
        ELSE IF(ICOL1(I1).GT.ICOL2(I2))THEN
C         Column in eq2 is first.
          NC = NC + 1
          ICOLC(NC) = ICOL2(I2)
          COFC(NC) = EQ2M*COF2(I2)
          IF(I2.EQ.N2)GOTO 50
          I2 = I2 + 1
          GOTO 30
        ELSE
C         Column in eq1 and eq2 are the same.
          IF(ICOL1(I1).NE.ICOLEL)THEN
            NC = NC + 1
            ICOLC(NC) = ICOL1(I1)
            COFC(NC) = EQ1M*COF1(I1) + EQ2M*COF2(I2)
          END IF
          IF(I1.EQ.N1 .AND. I2.EQ.N2)THEN
            IERR = 0
            RETURN
          END IF
          IF(I1.EQ.N1)THEN
            I2 = I2 + 1
            GOTO 40
          END IF
          IF(I2.EQ.N2)THEN
            I1 = I1 + 1
            GOTO 50
          END IF
          I1 = I1 + 1
          I2 = I2 + 1
          GOTO 30
        END IF

40      CONTINUE
```

```
C       Equation 1 used up.  Finish equation 2.
        DO I=I2,N2
          NC = NC + 1
          ICOLC(NC) = ICOL2(I)
          COFC(NC) = EQ2M*COF2(I)
        END DO

        IERR = 0
        RETURN

50      CONTINUE

C       Equation 2 used up.  Finish equation 1.
        DO I=I1,N1
          NC = NC + 1
          ICOLC(NC) = ICOL1(I)
          COFC(NC) = EQ1M*COF1(I)
        END DO

        IERR = 0
        RETURN

        END
```

## File **ELLIP.FOR**

```
C
      SUBROUTINE ELLIP(NI,NJ,NK,J,X1,X3,X1POS,X3POS,TANANG,MSGLVL)
C     This is the ISES grid generator modified for 3D arrays of x and z.
C     The grid is updated for the spanwise station J.
C     ANGUP is the upstream grid angle.  The second axial grid station is
C     held fixed.  The angle is then applied to the first
C     upstream station after the grid has been updated.
C     Convergence history is printed when MSGLVL=1.
      INTEGER NI,NJ,NK,J
      REAL X1(1),X3(1),X1POS(1),X3POS(1)
      REAL TANANG
      INTEGER MSGLVL

      INTEGER MNK
      PARAMETER (MNK=200)
      REAL C(MNK),D(2,MNK)

      INTEGER I,K,N
      INTEGER NMP,NOP,NPP,NMO,NPO,NMM,NOM,NPM
      REAL XOO,XMP,XOP,XPP,XMO,XPO,XMM,XOM,XPM
      REAL YOO,YMP,YOP,YPP,YMO,YPO,YMM,YOM,YPM

      INTEGER ITMAX,ITER
      REAL DSET1,DSET2,DSET3,RLX,RLX1,RLX2,RLX3,DMAX
      REAL DXIM,DXIP,DXIAV,DETM,DETP,DETAV
      REAL DXDET,DYDET,DXDXI,DYDXI,ALF,BET,GAM,CXIM,CXIP,CETM,CETP
      REAL B,A,AINV,AD1,AD2
      INTEGER KFIN,KBACK

C     The stencil in the x3, x1 plane looks like:
C
C             x1 ^
C                |
C               nmp ------- nop ------- npp
C                |           |           |
C                |           |           |
C                |           |           |
C          ---- nmo -------  n  ------- npo -----> x3
C               |            |           |
C               |            |           |
C               |            |           |
C              nmm ------- nom ------- npm
C
 
C
      ITMAX = 50
C
      DSET1 = 1.0E-1
      DSET2 = 5.0E-3
      DSET3 = 2.0E-4
C
      RLX1 = 1.30          !          DMAX > DSET1
      RLX2 = 1.50          !  DSET1 > DMAX > DSET2
      RLX3 = 1.60          !  DSET2 > DMAX > DSET3
CCC   STOP                 !  DSET3 > DMAX
C
      RLX = RLX1
C
      DO 1 ITER = 1, ITMAX
C
        DMAX = 0.
        DO 5 I=2, NI-1
C
C         Keep first 2 and last 4 stations fixed.
          DO 6 K=3, NK-5
            N = (K-1)*NI*NJ + (J-1)*NI + I
            NOP = N + 1
            NOM = N - 1
            NPO = N + NI*NJ
            NPP = NPO + 1
            NPM = NPO - 1
            NMO = N - NI*NJ
            NMP = NMO + 1
            NMM = NMO - 1
C
            XMM = X3(NMM)
            XOM = X3(NOM)
            XPM = X3(NPM)
            XMO = X3(NMO)
            XOO = X3(N)
            XPO = X3(NPO)
            XMP = X3(NMP)
            XOP = X3(NOP)
            XPP = X3(NPP)

            YMM = X1(NMM)
            YOM = X1(NOM)
            YPM = X1(NPM)
            YMO = X1(NMO)
            YOO = X1(N)
            YPO = X1(NPO)
            YMP = X1(NMP)
            YOP = X1(NOP)
            YPP = X1(NPP)
C
            DXIM = X3POS(K)-X3POS(K-1)
            DXIP = X3POS(K+1)-X3POS(K)
            DXIAV = 0.5*(DXIM+DXIP)
C
            DETM = X1POS(I)-X1POS(I-1)
            DETP = X1POS(I+1)-X1POS(I)
            DETAV = 0.5*(DETM+DETP)
C
```

```
           DXDET = ( XOP - XOM ) / DETAV
           DYDET = ( YOP - YOM ) / DETAV
           DXDXI = ( XPO - XMO ) / DXIAV
           DYDXI = ( YPO - YMO ) / DXIAV
C
           ALF = DXDET**2 + DYDET**2
           BET = DXDET*DXDXI + DYDET*DYDXI
           GAM = DXDXI**2 + DYDXI**2
C
           CXIM = 1.0 / (DXIM*DXIAV)
           CXIP = 1.0 / (DXIP*DXIAV)
           CETM = 1.0 / (DETM*DETAV)
           CETP = 1.0 / (DETP*DETAV)
C
           B =      -ALF*CXIM
           A = ALF*(CXIM+CXIP) + GAM*(CETM+CETP)
           C(K) =  -ALF*CXIP
           IF(K.EQ.2) B = 0
C
           D(1,K) = ALF*((XMO-XOO)*CXIM + (XPO-XOO)*CXIP)
     &             - 2.0*BET*(XPP-XMP-XPM+XMM) / (4.0*DXIAV*DETAV)
     &             + GAM*((XOM-XOO)*CETM + (XOP-XOO)*CETP)
C
           D(2,K) = ALF*((YMO-YOO)*CXIM + (YPO-YOO)*CXIP)
     &             - 2.0*BET*(YPP-YMP-YPM+YMM) / (4.0*DXIAV*DETAV)
     &             + GAM*((YOM-YOO)*CETM + (YOP-YOO)*CETP)
C
           AINV = 1.0/(A - B*C(K-1))
           C(K) = C(K) * AINV
           D(1,K) = ( D(1,K) - B*D(1,K-1) ) * AINV
           D(2,K) = ( D(2,K) - B*D(2,K-1) ) * AINV
C
  6      CONTINUE
C
         D(1,NK) = 0.
         D(2,NK) = 0.
C
         KFIN = NK-2
         DO 8 KBACK=2, KFIN
           K = NK-KBACK+1
           D( 1,K) = D( 1,K) - C(K)*D(1,K+1)
           D( 2,K) = D( 2,K) - C(K)*D(2,K+1)
           N = (K-1)*NI*NJ + (J-1)*NI + I
           X3(N) = X3(N) + RLX*D(1,K)
           X1(N) = X1(N) + RLX*D(2,K)
           AD1 = ABS(D(1,K))
           AD2 = ABS(D(2,K))
           DMAX = AMAX1(DMAX,AD1,AD2)
  8      CONTINUE
C
  5    CONTINUE
C
       IF(MSGLVL.EQ.1)WRITE(6,*) 'Dmax = ', DMAX, RLX
C
       RLX = RLX1
       IF(DMAX.LT.DSET1) RLX = RLX2
       IF(DMAX.LT.DSET2) RLX = RLX3
       IF(DMAX.LT.DSET3) THEN
C        Set up the upstream boundary.
         K = 1
         DO I=1,NI
           N = (K-1)*NI*NJ + (J-1)*NI + I
           X1(N) = X1(N+NI*NJ) - TANANG*(X3(N+NI*NJ) - X3(N))
         END DO
         RETURN
       END IF
```

```
C
   1  CONTINUE
C
      RETURN
      END
```

## File FGDER.FOR

```
C       This file contains the Fixed Grid derivative calculation subroutines
C       for A, B and S faces which are used for calculating psi1 and psi2
C       derivatives.

C       Theses routines have a similar input and output.
C       Input:  CRD(12) is an array of coordinate values dimensioned to 12.
C       Output: DDXI1, DDXI2, and DDXI3 are the xi1, xi2, and xi3 derivatives
C               of the physical coordinates using the same stencil as
C               the corresponding psi1 and psi2 derivatives.

        SUBROUTINE FGDA1(CRD,DDXI1,DDXI2,DDXI3)
C       Fixed Grid Derivate for an A face used for psi1 calculation.
        REAL CRD(12),DDXI1,DDXI2,DDXI3

        INCLUDE 'FACE.FOR'

C      Dcoord/Dxi1
       DDXI1 = 0.5*(CRD(7) - CRD(5))

C      Dcoord/Dxi2
       IF(IFTYP.EQ.0)THEN
         DDXI2 = 0.5*(CRD(8) - CRD(4))
       ELSE IF(IFTYP.EQ.1)THEN
         DDXI2 = 0.5*(-3*CRD(6) + 4*CRD(8) - CRD(4))
       ELSE IF(IFTYP.EQ.2)THEN
         DDXI2 = 0.5*(3*CRD(6) - 4*CRD(4) + CRD(8))
       ELSE IF(IFTYP.EQ.3)THEN
         DDXI2 = 0
       END IF

C      Dcoord/Dxi3
       DDXI3 = 0.5*(CRD(10) - CRD(2))

        RETURN
        END


        SUBROUTINE FGDB1(CRD,DDXI1,DDXI2,DDXI3)
C       Fixed Grid Derivate for a B face used for psi1 calculation.
        REAL CRD(12),DDXI1,DDXI2,DDXI3

        INCLUDE 'FACE.FOR'

C      Dcoord/Dxi1
       DDXI1 = 0.5*((CRD(6) + CRD(8)) - (CRD(5) + CRD(7)))

C      Dcoord/Dxi2
       DDXI2 = 0.5*((CRD(7) + CRD(8)) - (CRD(5) + CRD(6)))

C      Dcoord/Dxi3
       DDXI3 = ((CRD(9) + CRD(10) + CRD(11) + CRD(12)) -
     1          (CRD(1) + CRD(2) + CRD(3) + CRD(4)))/8.

        RETURN
        END

        SUBROUTINE FGDS1(CRD,DDXI1,DDXI2,DDXI3)
```

```
C        Fixed Grid Derivate for an S face used for psi1 calculation.
         REAL CRD(12),DDXI1,DDXI2,DDXI3

         INCLUDE 'FACE.FOR'

C      Dcoord/Dxi1
       DDXI1 = 0.5*((CRD(4) + CRD(10)) - (CRD(3) + CRD(9)))

C      Dcoord/Dxi2
       IF(IFTYP.EQ.1 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.6)THEN
         DDXI2 = (-3*(CRD(3) + CRD(4) + CRD(9) + CRD(10)) +
     1             4*(CRD(5) + CRD(6) + CRD(11) + CRD(12)) -
     1               (CRD(1) + CRD(2) + CRD(7) + CRD(8)))/8.
       ELSE IF(IFTYP.EQ.2 .OR. IFTYP.EQ.7 .OR. IFTYP.EQ.8)THEN
         DDXI2 = (3*(CRD(3) + CRD(4) + CRD(9) + CRD(10)) -
     1            4*(CRD(1) + CRD(2) + CRD(7) + CRD(8)) +
     1              (CRD(5) + CRD(6) + CRD(11) + CRD(12)))/8.
       ELSE IF(IFTYP.EQ.9)THEN
         DDXI2 = 0
       ELSE IF(.TRUE.)THEN
         DDXI2 = ((CRD(5) + CRD(6) + CRD(11) + CRD(12)) -
     1            (CRD(1) + CRD(2) + CRD(7) + CRD(8)))/8.
       END IF

C      Dcoord/Dxi3
       DDXI3 =0.5*((CRD(9) + CRD(10)) - (CRD(3) + CRD(4)))

         RETURN
         END

         SUBROUTINE FGDA2(CRD,DDXI1,DDXI2,DDXI3)
C        Fixed Grid Derivate for an A face used for psi2 calculation.
         REAL CRD(12),DDXI1,DDXI2,DDXI3

         INCLUDE 'FACE.FOR'

C      Dcoord/Dxi1
       DDXI1 = 0.5*((CRD(6) + CRD(8)) - (CRD(5) + CRD(7)))

C      Dcoord/Dxi2
       DDXI2 = 0.5*((CRD(7) + CRD(8)) - (CRD(5) + CRD(6)))

C      Dcoord/Dxi3
       DDXI3 = ((CRD(9) + CRD(10) + CRD(11) + CRD(12)) -
     1          (CRD(1) + CRD(2) + CRD(3) + CRD(4)))/8.

         RETURN
         END

         SUBROUTINE FGDB2(CRD,DDXI1,DDXI2,DDXI3)
C        Fixed Grid Derivate for an B face used for psi2 calculation.
         REAL CRD(12),DDXI1,DDXI2,DDXI3

         INCLUDE 'FACE.FOR'

C      Dcoord/Dxi1
       IF(IFTYP.EQ.0)THEN
         DDXI1 = 0.5*(CRD(7) - CRD(5))
       ELSE IF(IFTYP.EQ.1)THEN
         DDXI1 = 0.5*(-3*CRD(6) + 4*CRD(7) - CRD(5))
       ELSE IF(IFTYP.EQ.2)THEN
         DDXI1 = 0.5*(3*CRD(6) - 4*CRD(5) + CRD(7))
       ELSE IF(IFTYP.EQ.3)THEN
         DDXI1 = 0
       END IF
```

```
C     Dcoord/Dxi2
      DDXI2 = 0.5*(CRD(8) - CRD(4))

C     Dcoord/Dxi3
      DDXI3 = 0.5*(CRD(10) - CRD(2))

        RETURN
        END

        SUBROUTINE FGDS2(CRD,DDXI1,DDXI2,DDXI3)
C       Fixed Grid Derivate for an S face used for psi2 calculation.
        REAL CRD(12),DDXI1,DDXI2,DDXI3

        INCLUDE 'FACE.FOR'

C     Dcoord/Dxi1
      IF(IFTYP.EQ.3 .OR. IFTYP.EQ.5 .OR. IFTYP.EQ.7)THEN
        DDXI1 = (-3*(CRD(2) + CRD(5) + CRD(8) + CRD(11)) +
     1            4*(CRD(3) + CRD(6) + CRD(9) + CRD(12)) -
     1             (CRD(1) + CRD(4) + CRD(7) + CRD(10)))/8.
      ELSE IF(IFTYP.EQ.4 .OR. IFTYP.EQ.6 .OR. IFTYP.EQ.8)THEN
        DDXI1 = (3*(CRD(2) + CRD(5) + CRD(8) + CRD(11)) -
     1           4*(CRD(1) + CRD(4) + CRD(7) + CRD(10)) +
     1             (CRD(3) + CRD(6) + CRD(9) + CRD(12)))/8.
      ELSE IF(IFTYP.EQ.9)THEN
        DDXI1 = 0
      ELSE IF(.TRUE.)THEN
        DDXI1 = ((CRD(3) + CRD(6) + CRD(9) + CRD(12)) -
     1           (CRD(1) + CRD(4) + CRD(7) + CRD(10)))/8
      END IF

C     Dcoord/Dxi2
      DDXI2 = 0.5*((CRD(5) + CRD(11)) - (CRD(2) + CRD(8)))

C     Dcoord/Dxi3
      DDXI3 = 0.5*((CRD(8) + CRD(11)) - (CRD(2) + CRD(5)))

        RETURN
        END
```

## File GETENT.FOR

```
C       This routine is for compressible and incompressible flow.
C       For incompressible flow where the entropy variable,
C               S = PTrel - rho (omega r)**2/2.
C                 = P + rho V**2/2 - rho (omega r)**2/2.
C       For compressible flow the entropy variable is,
C               S = gam/(gam-1) ln{ (I - W^2/2 + (omega r)^2/2)/Href }
C                   - ln{p/Pref}
C
C
        SUBROUTINE GETENT(SDC,PMSP,IPMSP,NPDROW,IWSENT)
C       This routine sets up the entropy matrix which is stored such that
C               Ps = PM (psi1,psi2) + SDC Ent.,
C       where Ps is the pressure vector, PM is the psi matrix, and SDC is
C       the entropy diagonal matrix.
        REAL SDC(1),PMSP(1)
        INTEGER IPMSP(1),NPDROW(1),IWSENT(1)
        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'
```

```
      INTEGER NEQCAS
      COMMON /CASRT/NEQCAS

      INTEGER ISF,I

      NEQCAS = NSV

C     Entropy throughout grid.
      DO ISF=1,NSF
        NCX = 0
        IF(CMPRES)THEN
          CALL LENTC(ISF,1.)
        ELSE
          CALL LENT(ISF,1.)
        END IF
C       Get the coefficient for the pressure and normalize the equation
C       by it.
        DO I=1,NCX
          IF(ICOLX(I).EQ.IPSO+ISF)THEN
            SDC(ISF) = 1./COFX(I)
            COFX(I) = 0.
            GOTO 10
          END IF
        END DO
10      CONTINUE
        DO I=1,NCX
          COFX(I) = -COFX(I)*SDC(ISF)
        END DO
        CALL CASRTS(COFX,ICOLX,NCX,ISF,PMSP,IPMSP,NPDROW,IWSENT)
      END DO

      RETURN
      END
```

## File INIMOM.FOR

```
      SUBROUTINE IMA(IAF)
C     This routine initializes the variables in the FACE commons
C     which are not global to the algorithm for the A face IAF.
      INTEGER IAF

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER I,J,K,N1,N2

C     Initialize those quantities independent of face type.
      P = PA(IAF)

      IP = IAF

      K = (IAF-1)/(NI*(NJ-1)) + 2
      J = (IAF - (K-2)*NI*(NJ-1) - 1)/NI + 1
      I = IAF - (K-2)*NI*(NJ-1) - (J-1)*NI

C     4 corner volume nodes at face.  At constant i.  a and b are at k.
C     c and d are at k-1.  a and d are at j.  b and c are at j+1.
      LA = (K-1)*NI*NJ + (J-1)*NI + I
      LB = LA + NI
      LC = LB - NI*NJ
      LD = LC - NI

C     Get the coordinate values at the volume nodes a-d of face.
      CALL CRDFVN(LA,X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA)
      CALL CRDFVN(LB,X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB)
      CALL CRDFVN(LC,X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC)
      CALL CRDFVN(LD,X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED)

C     The index of the J values for a-d and b-c.
      JSLEAD = J
      JSLEBC = J + 1

C     Check if this is a wall.
      IF(I.EQ.1 .OR. I.EQ.NI)THEN
        IFTYP = -1
        RETURN
      END IF

C     Psi2 indices.
      IB6 = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
      IB5 = IB6 - 1
      IB8 = IB6 + (NI-1)
      IB7 = IB8 - 1
      IB2 = IB6 - (NI-1)*NJ
      IB1 = IB2 - 1
      IB4 = IB2 + (NI-1)
      IB3 = IB4 - 1
      IB10 = IB6 + (NI-1)*NJ
      IB9 = IB10 - 1
      IB12 = IB10 + (NI-1)
      IB11 = IB12 - 1

C     Psi2 values.
      B1 = PSI2(IB1)
      B2 = PSI2(IB2)
      B3 = PSI2(IB3)
      B4 = PSI2(IB4)
      B5 = PSI2(IB5)
      B6 = PSI2(IB6)
      B7 = PSI2(IB7)
      B8 = PSI2(IB8)
      B9 = PSI2(IB9)
      B10 = PSI2(IB10)
      B11 = PSI2(IB11)
      B12 = PSI2(IB12)

C     Psi1 indices.
      IA6 = (K-1)*NI*(NJ-1) + (J-1)*NI + I
      IA2 = IA6 - NI*(NJ-1)
      IA10 = IA6 + NI*(NJ-1)

C     Assign these psi1 values.
      A2 = PSI1(IA2)
      A6 = PSI1(IA6)
      A10 = PSI1(IA10)

      IA5 = IA6 - 1
      A5 = PSI1(IA5)
      IA7 = IA6 + 1
      A7 = PSI1(IA7)

      IF(NJ.EQ.2)THEN
C       2-D problem.
        IFTYP = 3
        GOTO 10
      ELSE IF(J.EQ.1)THEN
C       Face next to bottom wall.
        IFTYP = 1
        IA8 = IA6 + NI
        IA4 = IA8 + NI
      ELSE IF(J.EQ.NJ-1)THEN
```

```
C         Face next to top wall.
          IFTYP = 2
          IA4 = IA6 - NI
          IA8 = IA4 - NI
        ELSE
C         Interior face.
          IFTYP = 0
          IA8 = IA6 + NI
          IA4 = IA6 - NI
        END IF

        IA3 = IA8 - NI*(NJ-1)
        IA11 = IA8 + NI*(NJ-1)
        IA1 = IA4 - NI*(NJ-1)
        IA9 = IA4 + NI*(NJ-1)

C       Assign psi1.  There is no a12 on this face.
        A1 = PSI1(IA1)
        A3 = PSI1(IA3)
        A4 = PSI1(IA4)
        A8 = PSI1(IA8)
        A9 = PSI1(IA9)
        A11 = PSI1(IA11)

10      CONTINUE
C       Coordinates for grid derivatives.  The subroutines do not check
C       for periodic boundaries.  These must be added here.
        CALL CRDSI1
        CALL CRDSI2

C       Jacobians if 2-D.
        IF(IFTYP.EQ.3)THEN
          N1 = (K-1)*NI*NJ + (J-1)*NI + 1
          N2 = N1 + NI
          J42D = X1N(N2) - X1N(N1)
          J52D = X2N(N2) - X2N(N1)
          J62D = X3N(N2) - X3N(N1)
        END IF

C       Moving Grid Derivative Initialization.

        MGD = IFTYP
        IDX15 = (K - 1)*(NJ-1) + J
        JBLEI2 = J
        IF(MGD.EQ.0)THEN
          IDX14 = IDX15 - 1
          IDX16 = IDX15 + 1
          JBLEI1 = JBLEI2 - 1
          JBLEI3 = JBLEI2 + 1
        ELSE IF(MGD.EQ.1)THEN
          IDX16 = IDX15 + 1
          IDX14 = IDX16 + 1
          JBLEI3 = JBLEI2 + 1
          JBLEI1 = JBLEI3 + 1
        ELSE IF(MGD.EQ.2)THEN
          IDX14 = IDX15 - 1
          IDX16 = IDX14 - 1
          JBLEI1 = JBLEI2 - 1
          JBLEI3 = JBLEI1 - 1
        ELSE
C         Use dummy values for 4 and 6 for MGD = 3.
          IDX14 = 2
          IDX16 = 2
C         Use dummy values for 1 and 3 for MGD = 3.
          JBLEI1 = 1
          JBLEI3 = 1
```

```
      END IF

      IDX17 = IDX14 + (NJ-1)
      IDX18 = IDX15 + (NJ-1)
      IDX19 = IDX16 + (NJ-1)

      IDX11 = IDX14 - (NJ-1)
      IDX12 = IDX15 - (NJ-1)
      IDX13 = IDX16 - (NJ-1)

      IF(CMPRES)THEN
        IF(IHUP.EQ.1)THEN
C           When rothalpy tracking works, get rothalpy.
          CONTINUE
        END IF
      END IF

      RETURN
      END


      SUBROUTINE IMB(IBF)
C     This routine initializes the variables in the FACE commons
C     which are not global to the algorithm for the B face IBF.
      INTEGER IBF

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER I,J,K,N1,N2

C     Initialize those quantities independent of face type.
      P = PB(IBF)

      IP = IBF

      K = (IBF-1)/((NI-1)*NJ) + 2
      J = (IBF - (K-2)*(NI-1)*NJ - 1)/(NI-1) + 1
      I = IBF - (K-2)*(NI-1)*NJ - (J-1)*(NI-1)

C     4 corner volume nodes at face.  At constant j.  a and b are at k-1.
C     c and d are at k.  a and d are at i.  b and c are at i+1.
      LD = (K-1)*NI*NJ + (J-1)*NI + I
      LC = LD + 1
      LA = LD - NI*NJ
      LB = LA + 1

C     Get the coordinate values at the volume nodes a-d of face.
      CALL CRDFVN(LA,X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA)
      CALL CRDFVN(LB,X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB)
      CALL CRDFVN(LC,X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC)
      CALL CRDFVN(LD,X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED)

C     The index of the J values for a-d and b-c.
      JSLEAD = J
      JSLEBC = J

C     Check if this is a wall.
      IF(J.EQ.1 .OR. J.EQ.NJ)THEN
        IFTYP = -1
        RETURN
      END IF

C     Psi1 indices.
      IA7 = (K-1)*NI*(NJ-1) + (J-1)*NI + I
      IA8 = IA7 + 1
      IA5 = IA7 - NI
      IA6 = IA5 + 1
      IA3 = IA7 - NI*(NJ-1)
      IA4 = IA3 + 1
      IA1 = IA3 - NI
      IA2 = IA1 + 1
      IA11 = IA7 + NI*(NJ-1)
      IA12 = IA11 + 1
      IA9 = IA11 - NI
      IA10 = IA9 + 1

C     Assign psi1.
      A1 = PSI1(IA1)
      A2 = PSI1(IA2)
      A3 = PSI1(IA3)
      A4 = PSI1(IA4)
      A5 = PSI1(IA5)
      A6 = PSI1(IA6)
      A7 = PSI1(IA7)
      A8 = PSI1(IA8)
      A9 = PSI1(IA9)
      A10 = PSI1(IA10)
      A11 = PSI1(IA11)
      A12 = PSI1(IA12)

C     Psi2 indices.
      IB6 = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
      IB2 = IB6 - (NI-1)*NJ
      IB4 = IB6 - (NI-1)
      IB8 = IB6 + (NI-1)
      IB10 = IB6 + (NI-1)*NJ

C     Assign these psi2's.
      B2 = PSI2(IB2)
      B4 = PSI2(IB4)
      B6 = PSI2(IB6)
      B8 = PSI2(IB8)
      B10 = PSI2(IB10)

C     Rest of B's depend on type.
      IF(NI.EQ.2)THEN
C       2-D problem.
        IFTYP = 3
        GOTO 20
      ELSE IF(I.EQ.1)THEN
C       Face next to right wall.
        IFTYP = 1
        IB7 = IB6 + 1
        IB5 = IB7 + 1
      ELSE IF(I.EQ.NI-1)THEN
C       Face next to left wall.
        IFTYP = 2
        IB5 = IB6 - 1
        IB7 = IB5 - 1
      ELSE
C       Interior face.
        IFTYP = 0
        IB7 = IB6 + 1
        IB5 = IB6 - 1
      END IF

      IB1 = IB5 - (NI-1)*NJ
      IB3 = IB7 - (NI-1)*NJ
      IB9 = IB5 + (NI-1)*NJ
      IB11 = IB7 + (NI-1)*NJ
```

```
C       Assign rest of Psi2.
        B1 = PSI2(IB1)
        B3 = PSI2(IB3)
        B5 = PSI2(IB5)
        B7 = PSI2(IB7)
        B9 = PSI2(IB9)
        B11 = PSI2(IB11)

20      CONTINUE
C       Coordinates for grid derivatives.  The subroutines do not check
C       for periodic boundaries.  These must be added here.
        CALL CRDSI1
        CALL CRDSI2

C       Jacobians if 2-D.
        IF(IFTYP.EQ.3)THEN
          N1 = (K-1)*NI*NJ + (J-1)*NI + 1
          N2 = N1 + 1
          J12D = X1N(N2) - X1N(N1)
          J22D = X2N(N2) - X2N(N1)
          J32D = ZN(N2) - ZN(N1)
        END IF

C       Moving Grid Derivative Initialization.
        IDX14 = (K - 1)*(NJ-1) + J
        IDX13 = IDX14 - 1
        JBLEI2 = J
        JBLEI1 = JBLEI2 - 1
        IF(J.EQ.2)THEN
          MGD = 0
          IDX17 = IDX14 + 1
C         Use dummy values for 8.
          IDX18 = 1
          JBLEI3 = JBLEI2 + 1
        ELSE IF(J.EQ.NJ-1)THEN
          MGD = 1
          IDX17 = IDX13 - 1
C         Use dummy values for 8.
          IDX18 = 1
          JBLEI3 = JBLEI1 - 1
        ELSE
          MGD = 2
          IDX17 = IDX13 - 1
          IDX18 = IDX14 + 1
          JBLEI3 = JBLEI1 - 1
          JBLEI4 = JBLEI2 + 1
        END IF

        IDX15 = IDX13 + (NJ-1)
        IDX16 = IDX14 + (NJ-1)

        IDX11 = IDX13 - (NJ-1)
        IDX12 = IDX14 - (NJ-1)

        IF(CMPRES)THEN
          IF(IHUP.EQ.1)THEN
C           When rothalpy tracking works, get rothalpy.
            CONTINUE
          END IF
        END IF

        RETURN
        END


        SUBROUTINE IMS(ISF)
```

```
C       This routine initializes the variables in the FACE commons
C       which are not global to the algorithm for the S face ISF.
        INTEGER ISF

        INCLUDE '3DCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER I,J,K

C       Initialize those quantities independent of face type.
        P = PS(ISF)

        IP = ISF

        K = (ISF-1)/((NI-1)*(NJ-1)) + 1
        J = (ISF - (K-1)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
        I = ISF - (K-1)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C       Get the volume node info for this face.
        CALL FVNSF(I,J,K,LA,LB,LC,LD,
        1       X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA,
        1       X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB,
        1       X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC,
        1       X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED)

C       The index of the J values for a-d and b-c.
        JSLEAD = J
        JSLEBC = J + 1

C       Psi1 indices.
        IA3 = (K-1)*NI*(NJ-1) + (J-1)*NI + I
        IA4 = IA3 + 1
        IA9 = IA3 + NI*(NJ-1)
        IA10 = IA9 + 1

C       These psi1's.
        A3 = PSI1(IA3)
        A4 = PSI1(IA4)
        A9 = PSI1(IA9)
        A10 = PSI1(IA10)

C       Psi2 indices.
        IB2 = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
        IB5 = IB2 + (NI-1)
        IB8 = IB2 + (NI-1)*NJ
        IB11 = IB8 + (NI-1)

C       These psi2's.
        B2 = PSI2(IB2)
        B5 = PSI2(IB5)
        B8 = PSI2(IB8)
        B11 = PSI2(IB11)

C       Rest of psi1 and psi2 indices depend on the type.
        IF(NI.EQ.2 .OR. NJ.EQ.2)THEN
C         2-D problem.
          IFTYP = 9
          GOTO 10
        ELSE IF(J.EQ.1)THEN
          IA5 = IA3 + NI
          IA1 = IA5 + NI
          IF(I.EQ.1)THEN
C           Face next to bottom right wall.
            IFTYP = 5
            IB3 = IB2 + 1
            IB1 = IB3 + 1
          ELSE IF(I.EQ.NI-1)THEN
C           Face next to bottom left wall.
            IFTYP = 6
            IB1 = IB2 - 1
            IB3 = IB1 - 1
          ELSE
C           Face next to bottom wall.
            IFTYP = 1
            IB1 = IB2 - 1
            IB3 = IB2 + 1
          END IF
        ELSE IF(J.EQ.NJ-1)THEN
          IA1 = IA3 - NI
          IA5 = IA1 - NI
          IF(I.EQ.1)THEN
C           Face next to top right wall.
            IFTYP = 7
            IB3 = IB2 + 1
            IB1 = IB3 + 1
          ELSE IF(I.EQ.NI-1)THEN
C           Face next to top left wall.
            IFTYP = 8
            IB1 = IB2 - 1
            IB3 = IB1 - 1
          ELSE
C           Face next to top wall.
            IFTYP = 2
            IB1 = IB2 - 1
            IB3 = IB2 + 1
          END IF
        ELSE
          IA5 = IA3 + NI
          IA1 = IA3 - NI
          IF(I.EQ.1)THEN
C           Face next to right wall.
            IFTYP = 3
            IB3 = IB2 + 1
            IB1 = IB3 + 1
          ELSE IF(I.EQ.NI-1)THEN
C           Face next to left wall.
            IFTYP = 4
            IB1 = IB2 - 1
            IB3 = IB1 - 1
          ELSE
C           Interior face.
            IFTYP = 0
            IB1 = IB2 - 1
            IB3 = IB2 + 1
          END IF
        END IF

        IA2 = IA1 + 1
        IA7 = IA1 + NI*(NJ-1)
        IA8 = IA7 + 1
        IA6 = IA5 + 1
        IA11 = IA5 + NI*(NJ-1)
        IA12 = IA11 + 1

        IB4 = IB1 + (NI-1)
        IB7 = IB1 + (NI-1)*NJ
        IB10 = IB7 + (NI-1)
        IB6 = IB3 + (NI-1)
        IB9 = IB3 + (NI-1)*NJ
        IB12 = IB9 + (NI-1)

C       Assign the rest of the psi1's and psi2's.
```

```
        A1 = PSI1(IA1)
        A2 = PSI1(IA2)
        A5 = PSI1(IA5)
        A6 = PSI1(IA6)
        A7 = PSI1(IA7)
        A8 = PSI1(IA8)
        A11 = PSI1(IA11)
        A12 = PSI1(IA12)

        B1 = PSI2(IB1)
        B3 = PSI2(IB3)
        B4 = PSI2(IB4)
        B6 = PSI2(IB6)
        B7 = PSI2(IB7)
        B9 = PSI2(IB9)
        B10 = PSI2(IB10)
        B12 = PSI2(IB12)

10      CONTINUE
C       Coordinates for grid derivatives.  The subroutines do not check
C       for periodic boundaries.  These must be added here.
        CALL CRDSI1
        CALL CRDSI2

C       Jacobians if 2-D.
        IF(IFTYP.EQ.9)THEN
          J12D = 0.5*((X1CC + X1DC) - (X1AC + X1BC))
          J22D = 0.5*((X2CC + X2DC) - (X2AC + X2BC))
          J32D = 0.5*((X3CC + X3DC) - (X3AC + X3BC))
          J42D = 0.5*((X1BC + X1CC) - (X1AC + X1DC))
          J52D = 0.5*((X2BC + X2CC) - (X2AC + X2DC))
          J62D = 0.5*((X3BC + X3CC) - (X3AC + X3DC))
        END IF

C       Moving Grid Derivative Initialization.

        IDX12 = (K - 1)*(NJ-1) + J
        IDX15 = IDX12 + (NJ-1)
        JBLEI2 = J
        IF(NJ.EQ.2)THEN
          MGD = 3
C         Use dummy values for 1 an 3 for MGD = 3.
          IDX11 = 1
          IDX13 = 1
C         Use dummy values for 1 an 3 for MGD = 3.
          JBLEI1 = 1
          JBLEI3 = 1
        ELSE IF(J.EQ.1)THEN
          MGD = 1
          IDX13 = IDX12 + 1
          IDX11 = IDX13 + 1
          JBLEI3 = JBLEI2 + 1
          JBLEI1 = JBLEI3 + 1
        ELSE IF(J.EQ.NJ-1)THEN
          MGD = 2
          IDX11 = IDX12 - 1
          IDX13 = IDX11 - 1
          JBLEI1 = JBLEI2 - 1
          JBLEI3 = JBLEI1 - 1
        ELSE
          MGD = 0
          IDX13 = IDX12 + 1
          IDX11 = IDX12 - 1
          JBLEI1 = JBLEI2 - 1
          JBLEI3 = JBLEI2 + 1
        END IF
```

```
      IDX14 = IDX11 + (NJ-1)
      IDX16 = IDX13 + (NJ-1)

      IF(CMPRES)THEN
        IF(IHUP.EQ.1)THEN
C         When rothalpy tracking works, get rothalpy.
          CONTINUE
        END IF
C       Get upstream faces for upwinding.
        IP1 = ISF - (NI-1)*(NJ-1)
        IP0 = IP1 - (NI-1)*(NJ-1)
C       Check if goes beyond upstream boundary.  For a supersonic inlet,
C       make the ustream pressure equal to the first.
        IF(IP1.LE.0)THEN
          IP1 = ISF
C         Set M1S=0, so the actual value at the first face is used.
          M1S = 0.
        ELSE
          M1S = MSS(IP1)
        END IF
        IF(IP0.LE.0)IP0 = ISF
        P1 = PS(IP1)
        P0 = PS(IP0)
        VSC = VSCS(ISF)
      END IF

      RETURN
      END


      SUBROUTINE CRDFVN(JN,X1V,X2V,X3V,D1DLEV,D2DLEV,D3DLEV)
C     This routine gets the coordinates of the face volume node JN.
C     X1V, X2V, X3V are the coordinate values for this volume node.
C     D1DLEV,D2DLEV,and D3DLEV are the derivative of x1, x2, and x3
C     with respect to the leading edge point for this volume node.
      INTEGER JN
      REAL X1V,X2V,X3V,D1DLEV,D2DLEV,D3DLEV

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER N1,N2

C     N1 and N2 are the nodes where the grid coordinates are stored.
      N1 = JN
      N2 = JN + NI*NJ

      X1V = 0.5*(X1N(N1) + X1N(N2))
      X2V = 0.5*(X2N(N1) + X2N(N2))
      X3V = 0.5*(X3N(N1) + X3N(N2))

      D1DLEV = 0.5*(NX1G(N1) + NX1G(N2))
      D2DLEV = 0.5*(NX2G(N1) + NX2G(N2))
      D3DLEV = 0.5*(NX3G(N1) + NX3G(N2))

      RETURN
      END

      SUBROUTINE IDX1VN(JN)
C     This routine is used to initialize the Dx1 at volume nodes
C     calculation routines.
      INTEGER JN

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'
      INTEGER J,K

      K = (JN-1)/(NI*NJ) + 1
      J = (JN - (K-1)*NI*NJ - 1)/NI + 1

      JVN = J

      IF(NJ.EQ.2)THEN
        IDTH1 = K
        IDTH2 = IDTH1 + 1
        JVN = 0
      ELSE
        IF(J.EQ.NJ)THEN
          IDTH1 = K*(NJ-1)
          IDTH2 = IDTH1 - 1
          JVN = 1
        ELSE IF(J.EQ.1)THEN
          IDTH1 = (K - 1)*(NJ-1) + 1
          IDTH2 = IDTH1 + 1
        ELSE
          IDTH1 = (K - 1)*(NJ-1) + J - 1
          IDTH2 = IDTH1 + 1
        END IF
        IDTH3 = IDTH1 + (NJ-1)
        IDTH4 = IDTH2 + (NJ-1)
      END IF

      RETURN
      END

      SUBROUTINE CRDSI1
C     This routine puts the coordinate values at the psi1 locations in
C     IA(1:12) into the X1A, X2A, and X3A arrays in common.  It also puts
C     the leading edge derivatives into X1LEA, X2LEA, and X3LEA.

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER M,I1E,I,J,K,N1,N2

      DO M=1,12
        I1E = IA(M)
        K = (I1E-1)/(NI*(NJ-1)) + 1
        J = (I1E - (K-1)*NI*(NJ-1) - 1)/NI + 1
        I = I1E - (K-1)*NI*(NJ-1) - (J-1)*NI
        N1 = (K-1)*NI*NJ + (J-1)*NI + I
C       Check node index because some psi1 nodes are not defined.
        IF(N1.LE.0)GOTO 10
        N2 = N1 + NI
        X1A(M) = 0.5*(X1N(N1) + X1N(N2))
        X2A(M) = 0.5*(X2N(N1) + X2N(N2))
        X3A(M) = 0.5*(X3N(N1) + X3N(N2))
        X1LEA(M) = 0.5*(NX1G(N1) + NX1G(N2))
        X2LEA(M) = 0.5*(NX2G(N1) + NX2G(N2))
        X3LEA(M) = 0.5*(NX3G(N1) + NX3G(N2))
10      CONTINUE
      END DO

      RETURN
      END

      SUBROUTINE CRDSI2
C     This routine puts the coordinate values at the psi2 locations in
C     IB(1:12) into the X1B, X2B, and X3B arrays in common.  It also puts
C     the leading edge derivatives into X1LEB, X2LEB, and X3LEB.
```

```
        INCLUDE '3DCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER M,I2E,I,J,K,N1,N2

        DO M=1,12
          I2E = IB(M)
          K = (I2E-1)/((NI-1)*NJ) + 1
          J = (I2E - (K-1)*(NI-1)*NJ - 1)/(NI-1) + 1
          I = I2E - (K-1)*(NI-1)*NJ - (J-1)*(NI-1)
          N1 = (K-1)*NI*NJ + (J-1)*NI + I
C         Check node index because some psi2 nodes are not defined.
          IF(N1.LE.0)GOTO 10
          N2 = N1 + 1
          X1B(M) = 0.5*(X1N(N1) + X1N(N2))
          X2B(M) = 0.5*(X2N(N1) + X2N(N2))
          X3B(M) = 0.5*(X3N(N1) + X3N(N2))
          X1LEB(M) = 0.5*(NX1G(N1) + NX1G(N2))
          X2LEB(M) = 0.5*(NX2G(N1) + NX2G(N2))
          X3LEB(M) = 0.5*(NX3G(N1) + NX3G(N2))
10        CONTINUE
        END DO

        RETURN
        END

        SUBROUTINE FVNSF(I,J,K,LA,LB,LC,LD,
        1        X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA,
        1        X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB,
        1        X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC,
        1        X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED)
C       This routine determines the face volume node variables for an S
C       face. I, J and K are the indices for the s face.  The rest of the
C       arguments are the index, x, y, z, r, theta and delta theta for
C       node a, b, c, and d.
        INTEGER I,J,K,LA,LB,LC,LD
        REAL X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA
        REAL X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB
        REAL X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC
        REAL X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED

        INCLUDE '3DCOM.FOR'

C       4 corner volume nodes at face.  At constant k.  a and b are at i.
C       c and d are at i+1.  a and d are at j.  b and c are at j+1.
        LA = (K-1)*NI*NJ + (J-1)*NI + I
        LB = LA + NI
        LC = LB + 1
        LD = LA + 1

C       Get the coordinate values at the volume nodes of face.
        CALL CRDFVN(LA,X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA)
        CALL CRDFVN(LB,X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB)
        CALL CRDFVN(LC,X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC)
        CALL CRDFVN(LD,X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED)

        RETURN
        END


        SUBROUTINE IAUXP(M)
C       This routine initializes the auxilliary pressure equations for
C       volume M.
        INTEGER M
```

```
      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER I,J,K

C     Get the indices for the volume.
      K = (M-1)/((NI-1)*(NJ-1)) + 2
      J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
      I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C     Get the face indices.
C       a1 : i = I, j = J, k = K
        IPA1 = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C       a2 : a1 + 1,  i = I + 1, j = J, k = K
        IPA2 = IPA1 + 1
C       b1 : i = I, j = J, k = K
        IPB1 = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C       b2 : b1 + (ni-1),  i = I, j = J + 1, k = K
        IPB2 = IPB1 + (NI-1)
C       s1 : i = I, j = J, k = K - 1
        ISF1 = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C       s2 : s1 + (ni-1)(nj-1),  i = I, j = J, k = K
        ISF2 = ISF1 + (NI-1)*(NJ-1)

C     The pressure values.
      PS1 = PS(ISF1)
      PS2 = PS(ISF2)

      PA1 = PA(IPA1)
      PA2 = PA(IPA2)

      PB1 = PB(IPB1)
      PB2 = PB(IPB2)

C     The psi1 indices. psi1_2 is at i,j,k.
      ISI12 = (K-1)*NI*(NJ-1) + (J-1)*NI + I
      ISI11 = ISI12 - NI*(NJ-1)
      ISI13 = ISI12 + NI*(NJ-1)
      ISI14 = ISI11 + 1
      ISI15 = ISI12 + 1
      ISI16 = ISI13 + 1

C     The psi2 indices. psi2_2 is at i,j,k.
      ISI22 = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
      ISI21 = ISI22 - (NI-1)*NJ
      ISI23 = ISI22 + (NI-1)*NJ
      ISI24 = ISI21 + (NI-1)
      ISI25 = ISI22 + (NI-1)
      ISI26 = ISI23 + (NI-1)

C     The psi1 and psi2 values.
      AP1 = PSI1(ISI11)
      AP2 = PSI1(ISI12)
      AP3 = PSI1(ISI13)
      AP4 = PSI1(ISI14)
      AP5 = PSI1(ISI15)
      AP6 = PSI1(ISI16)

      BP1 = PSI2(ISI21)
      BP2 = PSI2(ISI22)
      BP3 = PSI2(ISI23)
      BP4 = PSI2(ISI24)
      BP5 = PSI2(ISI25)
      BP6 = PSI2(ISI26)

      IF(K.EQ.2 .OR. K.EQ.NK-1)THEN
```

```
          KA = 0.
          KB = 0.
        ELSE
          KA = KPA
          KB = KPB
        END IF

        IF(CMPRES)THEN
          IF(IHUP.EQ.1)THEN
C           When rothalpy tracking works, get rothalpy.
            CONTINUE
          ELSE
            H1 = H
            H2 = H
          END IF
        END IF

        RETURN
        END

        SUBROUTINE IRADEQ(I,J)
C       This routine initializes the radial equilibrium equation applied
C       between the s face i, j, nk-1 and i, j+1, nk-1.
        INTEGER I,J

        INCLUDE '3DCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER K

C       Get the face indices.
        K = NK - 1
C         s1 : i = I, j = J, k = K
          ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C         s2 : s1 + (ni-1),  i = I, j = J+1, k = K
          ISF2 = ISF1 + (NI-1)

C       The pressure values.
        PS1 = PS(ISF1)
        PS2 = PS(ISF2)

        IF(CMPRES)THEN
          IF(IHUP.EQ.1)THEN
C           When rothalpy tracking works, get rothalpy.
            CONTINUE
          ELSE
            H1 = H
            H2 = H
          END IF
        END IF

        C1 = CURV((J-1)*(NI-1) + I)
        C2 = CURV(J*(NI-1) + I)

C       Get the volume node info for face 1.
        CALL FVNSF(I,J,K,LA,LB,LC,LD,
        1       X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA,
        1       X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB,
        1       X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC,
        1       X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED)

        Z1 = 0.25*(X3AC + X3BC + X3CC + X3DC)

C       Get the volume node info for face 2.
        CALL FVNSF(I,J+1,K,LA,LB,LC,LD,
        1       X1AC,X2AC,X3AC,D1DLEA,D2DLEA,D3DLEA,
     1        X1BC,X2BC,X3BC,D1DLEB,D2DLEB,D3DLEB,
     1        X1CC,X2CC,X3CC,D1DLEC,D2DLEC,D3DLEC,
     1        X1DC,X2DC,X3DC,D1DLED,D2DLED,D3DLED)

      Z2 = 0.25*(X3AC + X3BC + X3CC + X3DC)

      RETURN
      END
```

## File IOSUBS.FOR

```
C     These routine are used in reading and writing the data domains
C     used by TARA so they work with the CRAY convert routine.

      SUBROUTINE READR(IUNIT,NREC,NDATA,NBLOCK,X)
C     This routine reads the real data X in NREC records of max NBLOCK long.
C     There are NDATA values read from logical unit IUNIT.
      INTEGER IUNIT,NREC,NDATA,NBLOCK
      REAL X(1)

      INTEGER IREC,NBEG,NEND,N

      DO 10 IREC=1,NREC
         NBEG = (IREC-1)*NBLOCK + 1
         NEND = MIN(NDATA,IREC*NBLOCK)
         READ(IUNIT)(X(N),N=NBEG,NEND)
10    CONTINUE

      RETURN
      END

      SUBROUTINE WRTER(IUNIT,NREC,NDATA,NBLOCK,X)
C     This routine writes the real data X in NREC records of max NBLOCK long.
C     There are NDATA values written from logical unit IUNIT.
      INTEGER IUNIT,NREC,NDATA,NBLOCK
      REAL X(1)

      INTEGER IREC,NBEG,NEND,N

      DO 10 IREC=1,NREC
         NBEG = (IREC-1)*NBLOCK + 1
         NEND = MIN(NDATA,IREC*NBLOCK)
         WRITE(IUNIT)(X(N),N=NBEG,NEND)
10    CONTINUE

      RETURN
      END
```

## File LSPFIT.FOR

```
*-LSPFIT-*******************************************************-LSPFIT-*

        SUBROUTINE LSPFIT(X,Y,NPTS, XC,YC,NXC,ND)
        DIMENSION X(10),Y(10), XC(10),YC(10)
*D
*D  Integrate or interpolate using a parabola which passes through
*D  points I and I+1 and misses points I-1 and I+2 (if they both
*D  exist) such that the square of the deviation is a minimum.
*D  Note that  I  is generally selected such that
CD              X(I).LE.XC.LT.X(I+1)
*D  The equation for the parabola is:
CD              Y-Y(I) = B*(X-X(I)) + C*(X-X(I))**2
```

```
CD
CD
CD      input
CD      X, Y    pts. on curve
CD      NPTS    no. of X,Y pairs
CD      XC      list of x coord at which calc to be done
CD      YC(1)   integration constant if  ND=-1
CD      NXC     no. of XC values
CD      ND      0 to interpolate
CD              1 to get 1st derivative,
CD              -1 for integration
CD
CD      COMMON /CLSPF / I,LEND
CD      COMMON /CLSPF2/ NEXTRP
CD
CD      LEND    linear fit in end interval, T or F (logical variable)
CD      NEXTRP  hold the end values constant outside of the curve definition
CD               when interpolating
CD
CD      output
CD
CD      YC      array of coordinate or derivative at XC    or
CD              YC(IC)= integral (Y*DX) from XC(1) to XC(IC) where IC=2,NXC
CD
CD      notes:
CD      X may be in either ascending or descending order.
CD      for integration XC must be in the same order as X.
CD      for interpolation no special order is required.
CD

        LOGICAL          LEND
        COMMON /CLSPF / I,LEND
        DATA            LEND/.FALSE./
        COMMON /CLSPF2/ NEXTRP
        DATA            NEXTRP/0/

        LOGICAL         WITHIN

        N      = NPTS-1
        IF(ND.EQ.(-1)) I=1
        ISAVE = 0
        SGN    = SIGN(1.,X(N+1)-X(1))

C    BEGIN INTERPOLATION LOOP FOR XC(IC)  IC=1,NXC
        IC     = 1

C    LOCATE APPROPRIATE INTERVAL
100     I      = MAX0(1,MIN0(I,N))
        WITHIN= .FALSE.
        NCOUNT= N
102     IF(NCOUNT) 119,103,103
103     NCOUNT= NCOUNT-1

        XI     = X(I)
        XD     = XC(IC)-XI
        IF(N) 104,120,104
104     IF(SGN*XD) 105,107,110

C      F.LT.O. (F IS THE FRACTIONAL POSITION IN THE INTERVAL)
105     IF(I.EQ.1)     GO TO 1202
        IF(ND.EQ.(-1)) GO TO 119
        I      = I-1
        GO TO 102

C      F.EQ.O
107     IF(X(I+1).NE.XI) GO TO 120
```

```
        IF(I.GE.N) GO TO 105
        GO TO 116

C       F.GT.O.
110     IF(SGN*(XC(IC)-X(I+1))) 120,112,114

C       F.EQ.1.0,  CHECK FOR INTEGRATION AND DOUBLE POINT BEFORE INCREMEN
112     IF(ND.EQ.(-1)) GO TO 120
        IF(I.NE.N .AND. X(I+1).EQ.X(I+2)) GO TO 120

C       F.GT.1.0
114     IF(I.EQ.N) GO TO 1204
        IF(ND.EQ.(-1)) GO TO 122
116     I     = I+1
        GO TO 102

119     CALL FXEM3(0,'pts not ordered or integration out of bounds in LSPF
        1IT',14)

C    Return end point values
1202    YC(IC)=Y(1)
        GO TO 1206
1204    YC(IC)=Y(NPTS)
1206    IF(ND.EQ.0 .AND. NEXTRP.EQ.1) GO TO 150
C    PRELIMINARY CALCULATIONS FOR INTERPOLATION OR INTEGRATION
120     WITHIN=.TRUE.
122     IF(I-ISAVE) 124,129,124
124     ISAVE = I
        YI    = Y(I)
        B=0.
        C=0.
        TOP=0.
        BOT=0.
        IF(N.EQ.0) GO TO 129
        X3    = X(I+1)-XI
        Y3    = Y(I+1)-YI
        IF(LEND .AND. (I.EQ.1 .OR. I.EQ.N)) GO TO 128
        IF(I.EQ.1) GO TO 126
        X1    = X(I-1)-XI
        X13   = X(I-1)-X(I+1)
        TOP   = X1*(Y3*X1-(Y(I-1)-YI)*X3)*X13
        BOT   = X1*X1*X13*X13*X3
 126    IF(I.GE.N) GO TO 128
        IF(XD.EQ.0. .AND. BOT.NE.0.) GO TO 127
        X4    = X(I+2)-XI
        X43   = X(I+2)-X(I+1)
        Y4    = Y(I+2)-YI
        TOP   = TOP + X4*(Y3*X4-Y4*X3)*X43
        BOT   = BOT + X4*X4*X43*X43*X3
        GO TO 128
 127    ISAVE=0
C      (X1**2 + X43**2)  MUST BE GREATER THAN  (X3/1000)**2
 128    IF(BOT.NE.0. .AND. ABS(BOT).GE.ABS((X3*X3)*(X3*X3)*X3*1.E-6))
        1 C   = -TOP/BOT
        IF(X3.NE.0.) B = (Y(I+1)-YI)/X3 - C*X3
 129    IF(ND) 130,140,141

C     ND=-1, INTEGRATE
 130    IF(.NOT.WITHIN) XD=X3
        S1    = (YI + (B/2. + C/3.*XD)*XD)*XD
        IF(WITHIN) GO TO 135
C     'I' IS BEING INCREMENTED TO FIND APPROPRIATE INTERVAL.  HENCE,
C       CUMULATE THE INTEGRAL OF THE ITH INTERVAL.
        SA    = SA + S1
        GO TO 116
C     APPROPRIATE INTERVAL FOUND.   X(I)-XC(IC)-X(I+1)
```

```
135     IF(IC.EQ.1) SA=YC(IC)-S1
        IF(IC.NE.1) YC(IC)=SA+S1
        GO TO 150

C    ND=0,  INTERPOLATE FOR COORDINATES
140     YC(IC)= YI + (B + C*XD)*XD
        GO TO 150

C    ND=1,  FIRST DERIVATIVE
141     YC(IC)= B + 2.*C*XD
        GO TO 150

150     IC    = IC+1
        IF(NXC-IC) 900,160,160
160     IF(ND.NE.(-1).AND.XC(IC).EQ.XC(IC-1)) I=I+1
        GO TO 100

900     RETURN
        END

*-FXEM3-*******************************************************-FXEM3-*

        SUBROUTINE FXEM3(N1,TEXT,N2)
*D
*D      dummy routine to simulate Honeywell system routine
*D
        CHARACTER TEXT*(*)
        WRITE(06,10) TEXT
10      FORMAT(1X,A)
        RETURN
        ENTRY FXEM1(NO)
        RETURN
        END
```

## File LSRCH.FOR

```
C
C                             LSRCH
C
C       Algorithm A6.3.1 (LINESEARCH) - Line Search
C       From "Numerical Methods for Unconstrained Optimization and Nonlinear
C       Equations"  by J.E. Dennis and Robert B. Schnabel

C       Given p, calculate x+ = xc + lambda p for lambda in (0,1]
C       such that f(x+) < f(xc) + alpha lambda (g transpose g)
C       using a back-tracking line search.

C       There are several modifications to this routine from that in the book.
C       One is the saving of the constant PMOD which is multiplied by
C       the p vector if MAXSTP is exceeded.  Therefore x+ = xc + lambda*pmod*p.
C       The other is not storing xc and x+, but getting the new function
C       values through the routine FN by passing lambda and P.  The previous
C       lambda is stored, so the solution is updated by (lambda-lambdaprev)*P.
C       To set lambdaprev initially, call FN with lambda < 0.

C       Also, the logical variable USESX is used to see if SX is input
C       as an array for normalizations.

C       And, if the initial slope is calculated to be positive, then
C       the full Newton step will be taken if it reduces the norm at all.
C       If not, a finite difference calculation of this slope will be made.
C       Epsilon will be taken to be 5.e-5.  If this calculated slope is
C       still greater than 1, then an error code will be set.
```

```
      SUBROUTINE LSRCH(N,FC,FN,GC,P,SX,USESX,MAXSTP,STPTOL,
     1        MSGLVL,RETCOD,FPLUS,MAXTAK,LAMBDA,PMOD)

C             Input:
C       N       Number of independant variables.
C       FC      Current value of minimization function.
C       FN      Name of external file which calculates the minimization
C               function.
C       GC      Current function gradient.
C       P       The Newton step.
C       SX      The normalization vector.
C       USESX   The logical variable indicating use of SX.
C                       Normalize: .TRUE.
C                       No normalization: .FALSE.
C       MAXSTP Maximum normalized step size.
C       STPTOL Minimum normalized step size.
C       MSGLVL  The message level indicator.
C                       0: none.
C                       1: warnings printed.
C               Output:
C       RETCOD  0: satisfactory x+ found
C                       1: routine failed to find a satisfactory x+
C       FPLUS   New value of minimization function.
C       MAXTAK Logical variable is true if max step size was taken.
C       LAMBDA  Step size.
C       PMOD    If MAXSTP was exceeded, then Newton step is multiplied
C               by PMOD first.

      INTEGER N,MSGLVL,RETCOD
      DOUBLE PRECISION FC,FPLUS
      EXTERNAL FN
      REAL GC(N),P(N),SX(N),MAXSTP,STPTOL
      REAL LAMBDA,PMOD
      LOGICAL MAXTAK,USESX

      REAL RELLEN,ALPHA,NWTLEN,MINLAM,LAMPRE,LAMTMP
      DOUBLE PRECISION INISLP,FPLUSP,A,B,DISC,FEPS

      DOUBLE PRECISION DOTPSS

      INTEGER I
      REAL EPS
      DATA EPS/5.E-5/

C     Algorithm

C     1.
      MAXTAK = .FALSE.

C     2.
      RETCOD = 2

C     3.
      ALPHA = 1.E-4

C     Initialize the variable updating.
      CALL FN(-1.,P,FPLUS)

C     4.
      IF(USESX)THEN
        CALL SCLNRM(N,SX,P,NWTLEN)
      ELSE
        CALL NORM(N,P,NWTLEN)
      END IF

C     5.
```

```
        PMOD = 1.
        IF(NWTLEN.GT.MAXSTP)THEN
C               5.1
                PMOD = MAXSTP/NWTLEN
                DO I=1,N
                        P(I) = P(I)*PMOD
                END DO
C               5.2
                NWTLEN = MAXSTP
        END IF

C       6.
        INISLP = DOTPSS(N,GC,P)
        IF(INISLP.GE.0.)THEN
          IF(MSGLVL.EQ.1)THEN
            PRINT*,'INISLP = ',INISLP,'.  Greater than 0.'
            PRINT*,'Setting to 0.'
          END IF
          INISLP = 0.
        END IF

C       7.
        RELLEN = 0.
        IF(USESX)THEN
          DO I=1,N
                RELLEN = MAX(  ABS(P(I)) * SX(I)   ,
        1                RELLEN  )
          END DO
        ELSE
          DO I=1,N
                RELLEN = MAX(  ABS(P(I))   ,
        1                RELLEN  )
          END DO
        END IF

C       Check RELLEN. If 0, converged.
        IF(RELLEN.EQ.0.)THEN
                LAMBDA = 1.

                CALL FN(LAMBDA,P,FPLUS)

                RETCOD = 0
                RETURN
        END IF

C       8.
        MINLAM = STPTOL/RELLEN

C       9.
        LAMBDA = 1.

C       10.
10      CONTINUE

C       10.2
        CALL FN(LAMBDA,P,FPLUS)

C       10.3a
        IF(FPLUS.LE.FC + ALPHA*LAMBDA*INISLP)THEN
C               10.3a.1
                RETCOD = 0
C               10.3a.2
                IF(LAMBDA.EQ.1. .AND. NWTLEN.GT.0.99*MAXSTP)MAXTAK=.TRUE.
                RETURN
C       10.3b
        ELSE IF(LAMBDA.LT.MINLAM)THEN
```

```
C             10.3b.1
              RETCOD = 1

              LAMBDA = 0.

              CALL FN(LAMBDA,P,FPLUS)

              RETURN

C       10.3c
        ELSE
C             10.3c.1
              IF(LAMBDA.EQ.1)THEN
C               Check that slope is less than 0.
                IF(INISLP.EQ.0.)THEN
                  IF(MSGLVL.EQ.1)THEN
                    PRINT*,'Calculating INISLP using finite difference.'
                  END IF
                  CALL FN(EPS,P,FEPS)
                  INISLP = (FEPS - FC)/EPS
                  IF(INISLP.GE.0.)THEN
                    IF(MSGLVL.EQ.1)THEN
                      PRINT*,'Finite difference INISLP = ',INISLP
                      PRINT*,'Greater than 0.  Cannot continue.'
                    END IF
                    LAMBDA = 0.
                    RETCOD = 1
                    CALL FN(LAMBDA,P,FPLUS)
                    RETURN
                  END IF
                END IF
C                     10.3c.1T.1
                      LAMTMP = -INISLP/(2.*(FPLUS-FC-INISLP))
              ELSE
C                     10.3c.E.1
                      CALL CUBCOF(LAMBDA,LAMPRE,FPLUS,FPLUSP,FC,
     1                        INISLP,A,B)

C                     10.3c.1E.2
                      DISC = B**2 - 3.*A*INISLP

C                     10.3c.1E.3
                      IF(A.EQ.0.)THEN
                              LAMTMP = -0.5*INISLP/B
                      ELSE
                              LAMTMP = (-B + SQRT(DISC))/(3.*A)
                      END IF

C                     10.3c.1E.4
                      IF(LAMTMP.GT.0.5*LAMBDA)LAMTMP = 0.5*LAMBDA
              END IF

C             10.3c.2
              LAMPRE = LAMBDA

C             10.3c.3
              FPLUSP = FPLUS

C             10.3c.4
              IF(LAMTMP.LE.0.1*LAMBDA)THEN
                      LAMBDA = 0.1*LAMBDA
              ELSE
                      LAMBDA = LAMTMP
              END IF
        END IF
```

```
      IF(RETCOD.LT.2)THEN
              CONTINUE
      ELSE
              GOTO 10
      END IF

      RETURN
      END


      SUBROUTINE CUBCOF(LAMBDA,LAMPRE,FPLUS,FPLUSP,FC,
     1        INISLP,A,B)
C     This routine solve for the cubic coefficients a and b in step
C     10.3c.1E.1 of the LINESEARCHMOD algorithm.
      REAL LAMBDA,LAMPRE
      DOUBLE PRECISION FPLUS,FPLUSP,FC,INISLP,A,B

      DOUBLE PRECISION A11,A12,A21,A22,B1,B2,DEN

      DEN = LAMBDA - LAMPRE

      A11 = 1./LAMBDA**2
      A12 = -1./LAMPRE**2
      A21 = -LAMPRE/LAMBDA**2
      A22 = LAMBDA/LAMPRE**2

      B1 = FPLUS - FC - LAMBDA*INISLP
      B2 = FPLUSP - FC - LAMPRE*INISLP

      A = (A11*B1 + A12*B2)/DEN
      B = (A21*B1 + A22*B2)/DEN

      RETURN
      END


C     Get the L2 norm of the scaled vector.
      SUBROUTINE SCLNRM(N,SX,V,DXVNRM)
      INTEGER N
      REAL SX(N),V(N),DXVNRM

      INTEGER I

      DXVNRM = 0.

      DO I=1,N
              DXVNRM = DXVNRM + (SX(I)*V(I))**2
      END DO

      DXVNRM = SQRT(DXVNRM)

      RETURN
      END

C     Get the L2 norm of the unscaled vector.
      SUBROUTINE NORM(N,V,DXVNRM)
      INTEGER N
      REAL V(N),DXVNRM

      INTEGER I

      DXVNRM = 0.

      DO I=1,N
              DXVNRM = DXVNRM + V(I)**2
      END DO
```

```
        DXVNRM = SQRT(DXVNRM)

        RETURN
        END

C       Dot Product
        DOUBLE PRECISION FUNCTION DOTPSS(N,X,Y)
C       Take the dot product of x and y both vectors of length n.
        INTEGER N
        REAL X(N),Y(N)

        INTEGER I

        DOTPSS = 0.

        DO I=1,N
                DOTPSS = DOTPSS + X(I)*Y(I)
        END DO

        RETURN
        END
```

## File ONEIT.FOR

```
        SUBROUTINE ONEIT
C       This routine performs one major iteration of the algorithm.

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'
        INCLUDE 'MESSAG.FOR'

        INTEGER N,I,RETCOD
        REAL LAMBDA,PMOD,DUM
        DOUBLE PRECISION BSQN
        LOGICAL MAXTAK,USESX
        EXTERNAL RHSSQ

        INTEGER IAF,IBF,MSGLVL,ISF
        INTEGER IOJCN,ISAJ,IOPMSP,ISPM,IOSP,ISISP

        INTEGER J,K,IDTH,I1E,M,ICNTMU,IERR
        INTEGER I2E,IPERMI

        INTEGER IOPT
        REAL TBEG,TEND
        LOGICAL USERCM

        ICNTSM = 0
        DO M=1,NM
          IF(ISSMAP(M))ICNTSM = ICNTSM + 1
        END DO
        IF(MSGLVO.EQ.1)THEN
          PRINT*,ICNTSM,' VOLUMES USE S-MOMENTUM EQ OUT OF ',NM
          ICNTMU = 0
          DO ISF=1,NSF
            IF(MSS(ISF).GT.MCS)ICNTMU = ICNTMU + 1
          END DO
          PRINT*,ICNTMU,' S FACES HAVE M SQUARED GT MCS'
        END IF

C       Get the grid dependence on the leading edge movement.
```

```
      CALL GRDDLE

C     Get the matrix left hand side J.
      IJDROW = NSV
      IAJ = IJDROW + NSV
      IJCN = IAJ + 60*NSV
      IIWSOM = IJCN + 60*NSV + 1
      CALL PTIME(TBEG)
      CALL GETLHS(WS(IAJ+1),IWS(IJCN+1),IWS(IJDROW+1))
      CALL PTIME(TEND)
      PRINT*,' TIME GETTING MATRIX LHS= ',TEND - TBEG
C     Compact the matrix.
      IOJCN = IJCN
C     ISAJ is the size of AJ.
      ISAJ = IWS(IJDROW+NSV)
      IJCN = IAJ + ISAJ
      DO I=1,ISAJ
        IWS(IJCN+I) = IWS(IOJCN+I)
      END DO
      ICWSB = IJCN + ISAJ

C     Get the gradient which is J transpose F.  F is -FVEC
      CALL PTIME(TBEG)
      CALL CALGRD(WS(IAJ+1),IWS(IJCN+1),IWS(IJDROW+1))
      CALL PTIME(TEND)
      PRINT*,' TIME GETTING MATRIX GRAD.= ',TEND - TBEG

      IF(USEENT)THEN
C       Set up the entropy matrix.
        ISD = ICWSB
        IPDROW = ISD + NSF
        ISPMSP = IPDROW + NSF
        IIPMSP = ISPMSP + 35*NSF
        IIWSE = IIPMSP + 35*NSF
        CALL GETENT(WS(ISD+1),WS(ISPMSP+1),WS(IIPMSP+1),WS(IPDROW+1),
     1       WS(IIWSE+1))
C       Compact the entropy matrix.
        IOPMSP = IIPMSP
        ISPM = IWS(IPDROW+NSF)
        IIPMSP = ISPMSP + ISPM
        DO I=1,ISPM
          IWS(IIPMSP+I) = IWS(IOPMSP+I)
        END DO
        ICWSB = IIPMSP+ISPM
      END IF

C     Set up the reduced equation matrix.  Unknows are psi1, psi2 and
C     entropy/pressure.
      IRHS = ICWSB
      INDROW = IRHS + NEQ
      ISSP = INDROW + NEQ
      IISP = ISSP + 80*NEQ
      IWSMSU = IISP + 80*NEQ
      CALL PTIME(TBEG)
      IF(USEENT)THEN
        CALL REDEQ(WS(ISD+1),WS(ISPMSP+1),WS(IIPMSP+1),WS(IPDROW+1))
      ELSE
        CALL REDEQP
      END IF
      CALL PTIME(TEND)
      PRINT*,' TIME REDUCING MATRIX = ',TEND - TBEG

C     At this point, the original matrix and entropy matrix could be
C     written to SSD, and the entire matrix compacted to the top of WS.

C     Compact matrix.
```

```
        ISISP = IWS(INDROW+NEQ)
        IOSP = IISP
        IISP = ISSP + ISISP
        DO I=1,ISISP
          IWS(IISP+I) = IWS(IOSP+I)
        END DO
        ICWSB = IISP + ISISP

C       Solve the matrix.
        CALL PTIME(TBEG)

C       Sparspak.   Not used at GE.
C       call spslv(neq,ws(issp+1),iws(iisp+1),iws(indrow+1),ws(irhs+1),
C       1       WS(ICWSB+1),MWS - ICWSB)

C       Set up the permutation vector which corresponds to normal ordering.
        IPERMI = ICWSB + NEQ
        IR = 0
        DO K=1,NK-1
          DO J=1,NJ-1
            DO I=1,NI-1
C             PSI1
              IR = IR + 1
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
              IWS(IPERMI+IR) = IPSI10 + I1E
C             PSI2
              IR = IR + 1
              I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
              IWS(IPERMI+IR) = IPSI20 + I2E
C             PS
              IR = IR + 1
              ISF = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
              IWS(IPERMI+IR) = IPSO + ISF
            END DO
            I = NI
C           PSI1
            IR = IR + 1
            I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
            IWS(IPERMI+IR) = IPSI10 + I1E
          END DO
          J = NJ
          DO I=1,NI-1
C           PSI2
            IR = IR + 1
            I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
            IWS(IPERMI+IR) = IPSI20 + I2E
          END DO
        END DO
        K = NK
        DO J=1,NJ-1
          DO I=1,NI-1
C           PSI1
            IR = IR + 1
            I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
            IWS(IPERMI+IR) = IPSI10 + I1E
C           PSI2
            IR = IR + 1
            I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
            IWS(IPERMI+IR) = IPSI20 + I2E
          END DO
          I = NI
C         PSI1
          IR = IR + 1
          I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
          IWS(IPERMI+IR) = IPSI10 + I1E
        END DO
```

```
      J = NJ
      DO I=1,NI-1
C       PSI2
        IR = IR + 1
        I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
        IWS(IPERMI+IR) = IPSI20 + I2E
      END DO

C     Check if new factor will be recalculated.
      IF(BSQ/NSV .LE. EPSOF)THEN
        IOPT = 17
      ELSE
        IOPT = -17
      END IF

C     Switch between RCM permutation and natural order set by USERCM.
C     If true, RCM ordering is used.
      USERCM = .FALSE.
      CALL SKYSOL(NEQ,ISSP,IISP,INDROW,IRHS, ICWSB,
      1       WS,IWS,MWS - ICWSB, IOPT,EPSM,KRYDIM,
      1       GMDISK,USERCM,IERR)

C     Set PNS to the psi1 psi2 and entropy/pressure.  UPPS will use this
C     to make changes in entropy changes in Ps if USEENT.
      DO IR=1,NEQ
        PNS(IR) = WS(IRHS+IR)
      END DO

C     Make sure the Delta theta and leading edge arc length
C     changes are put in the correct place.
      DO K=1,NK
        DO J=1,NJ-1
          I = 1
C         Delta theta.  Psi1 is 0.
          IDTH = (K-1)*(NJ-1) + J
C         IR = I1E
          I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
          PNS(IDTHO+IDTH) = PNS(I1E)
          PNS(I1E) = -PSI1(I1E)
        END DO
      END DO

      K = KSBLE
      DO J=1,NJ-1
        I = NI
C       Leading edge arc length.
C       IR = I1E
        I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
        PNS(ISBLEO+J) = PNS(I1E)
        IF(IDWNBC.EQ.2)THEN
C         Make sure (delta psi1)_xi3=0.
          PNS(I1E) = PNS(I1E+NI*(NJ-1)) -
      1       PSI1(I1E) + PSI1(I1E+NI*(NJ-1))
        ELSE
C         Psi1 = psi1mx.
          PNS(I1E) = PSI1MX - PSI1(I1E)
        END IF
      END DO

C     If the original matrix and entropy matrix were written to SSD,
C     read them back in overwriting the current matrix.

      IF(USEENT)THEN
C       Get changes to Ps.
        CALL UPPS(WS(ISD+1),WS(ISPMSP+1),WS(IIPMSP+1),WS(IPDROW+1))
      END IF
```

```
C       Update the changes to PA in PNS.
        IR = NEQ
        DO IAF=1,NAF
          IR = IR + 1
          CALL CALDPA(IAF,PNS(IR),WS(IAJ+1),IWS(IJCN+1),IWS(IJDROW+1))
        END DO

C       Update the changes to PB in PNS.
        DO IBF=1,NBF
          IR = IR + 1
          CALL CALDPB(IBF,PNS(IR),WS(IAJ+1),IWS(IJCN+1),IWS(IJDROW+1))
        END DO

C       Set POLD = PS which is used in update routine RHSSQ.
        DO ISF=1,NSF
          POLD(ISF) = PS(ISF)
        END DO

C       Do not use scale factor.
        USESX = .FALSE.

        STPTOL = 0.0001

        MSGLVL = 1

        CALL PTIME(TEND)
        PRINT*,' TOTAL TIME SOLVING MATRIX = ',TEND - TBEG

        IF(DMPOPT.GE.2)THEN
C         Use damping factor.
          IF(DMPOPT.EQ.3)THEN
            PRINT*,'DAMPING FACTOR?'
            READ*,DAMP
          END IF
          CALL RHSSQ(-1,PNS,BSQN)
          CALL RHSSQ(DAMP,PNS,BSQN)

          RETCOD = 0
          PMOD = 1
          LAMBDA = DAMP

        ELSE
C         Use Line search.
          CALL LSRCH(NSV,BSQ,RHSSQ,GC,PNS,DUM,USESX,MAXSTP,STPTOL,MSGLVL,
     1          RETCOD,BSQN,MAXTAK,LAMBDA,PMOD)

        END IF

        IF(MSGLVO.EQ.1)THEN
          PRINT*,' '
          PRINT*,'ITER, SQRT(BSQN/NSV)=, LAMBDA, PMOD'
          PRINT*,' ',ITER,' ',SQRT(BSQN/NSV),' ',LAMBDA,' ',PMOD
        END IF

        CALL PRSTAT

        IF(RETCOD.EQ.1)THEN
C         Major error.  Action depends on code MAJERR.
          PRINT*,' '
          PRINT*,'Error in linesearch.  Cannot decrease function.'
          IF(MAJERR.EQ.0)THEN
            PRINT*,' '
C           PRINT*,'Will output file and stop.'
            CALL OUT
            STOP
          ELSE IF(MAJERR.EQ.1)THEN
C           Try full Newton as a desparate attempt to make it work.
            PRINT*,' '
            PRINT*,'Will update with full Newton step and continue.'
            CALL RHSSQ(1.,PNS,BSQN)
            PRINT*,' '
            PRINT*,'SQRT(BSQN/NSV) = ',SQRT(BSQN/NSV)
            CALL PRSTAT
            BSQ = BSQN
            RETURN
          END IF
        END IF

        BSQ = BSQN

        RETURN
        END

        SUBROUTINE RHSSQ(LAMBDA,PNSS,FMIN)
C       This routine updates the variables by LAMBDA*PNSS and calculates the
C       right hand side squared. The variables were previously updated
C       by ALPREV stored in common, so this time the multiplier will
C       be (lambda - alprev).  To set alprev, call this routine with a negative
C       value for lambda.
        REAL LAMBDA,PNSS(1)
        DOUBLE PRECISION FMIN

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'MESSAG.FOR'
        INCLUDE 'FACE.FOR'
        INCLUDE 'CLSCOM.FOR'

        INTEGER N,I,ISF,M,ISAVEO,J,K
        REAL ALM,OMEGRS,DTHJK,R,X1
        REAL DSUP
        LOGICAL TABOK

        IF(LAMBDA.LT.O.)THEN
          ALPREV = 0.
          RETURN
        END IF

C       Update the unknowns.
C       ALM is the lambda multiplier.
        ALM = LAMBDA - ALPREV
        ALPREV = LAMBDA

        IF(ALM.EQ.O.)GOTO 10

        IR = 0

        DO N=1,N1E
          IR = IR + 1
C         Psi1.
          PSI1(N) = PSI1(N) + ALM*PNSS(IR)
        END DO

        DO N=1,N2E
          IR = IR + 1
C         Psi2.
          PSI2(N) = PSI2(N) + ALM*PNSS(IR)
        END DO

        DO N=1,NSF
          IR = IR + 1
```

```
C         PS.
          PS(N) = POLD(N) + LAMBDA*PNSS(IR)
        END DO

        DO N=1,NAF
          IR = IR + 1
C         PA.
          PA(N) = PA(N) + ALM*PNSS(IR)
        END DO

        DO N=1,NBF
          IR = IR + 1
C         PB.
          PB(N) = PB(N) + ALM*PNSS(IR)
        END DO

C               Geometry update.
C       Leading edge arc length.
        DO J=1,NJ
          IF(THINLE)THEN
            DSBLE(J) = 0.
          ELSE
            IF(NJ.EQ.2)THEN
              DSBLE(J) = PNSS(ISBLEO+1)
            ELSE IF(J.EQ.1)THEN
              DSBLE(J) = 1.5*PNSS(ISBLEO+1) - 0.5*PNSS(ISBLEO+2)
            ELSE IF(J.EQ.NJ)THEN
              DSBLE(J) = 1.5*PNSS(ISBLEO+NJ-1) - 0.5*PNSS(ISBLEO+NJ-2)
            ELSE
              DSBLE(J) = 0.5*PNSS(ISBLEO+J-1) + 0.5*PNSS(ISBLEO+J)
            END IF
cC          Clamp the leading edge movement to the average leading edge arc
cC          length.
c           DSUP = SBLD(NPBLD(J),J)*(SGL(2,J) + SGU(2,J))/4.
c           IF(ABS(DSBLE(J)).GT.DSUP)THEN
c             DSBLE(J) = ALM*SIGN(DSUP,DSBLE(J))
c           ELSE
              DSBLE(J) = ALM*DSBLE(J)
c           END IF
            SBLE(J) = SBLE(J) + DSBLE(J)
            DSBLEC(J) = DSBLEC(J) + DSBLE(J)
          END IF
        END DO

C       Grid point coordinates.
        DO K=1,NK
          DO J=1,NJ
            IF(NJ.EQ.2)THEN
              DTHJK = PNSS(IDTHO + (K-1)*(NJ-1) + 1)
            ELSE IF(J.EQ.1)THEN
              DTHJK = 1.5*PNSS(IDTHO + (K-1)*(NJ-1) + 1) -
     1          0.5*PNSS(IDTHO + (K-1)*(NJ-1) + 2)
            ELSE IF(J.EQ.NJ)THEN
              DTHJK = 1.5*PNSS(IDTHO + (K-1)*(NJ-1) + NJ-1) -
     1          0.5*PNSS(IDTHO + (K-1)*(NJ-1) + NJ-2)
            ELSE
              DTHJK = 0.5*PNSS(IDTHO + (K-1)*(NJ-1) + J-1) +
     1          0.5*PNSS(IDTHO + (K-1)*(NJ-1) + J)
            END IF
            DTHJK = ALM*DTHJK
            DO I=1,NI
              N = (K-1)*NI*NJ + (J-1)*NI + I
              X1N(N) = X1N(N) + DTHJK + NX1G(N)*DSBLE(J)
C             Update the x2, x3 only if leading edge movement is allowed.
              IF(.NOT.THINLE)THEN
                EMPRM(N) = EMPRM(N) + NMPG(N)*DSBLE(J)
            CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1                 EMPRM(N),X2N(N),1,0)
            CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1                 EMPRM(N),X3N(N),1,0)
          END IF
        END DO
      END DO
      END DO

C     Get the blade grid points if there is a moving leading edge.
      IF(.NOT. THINLE)CALL BLDGP

10    CONTINUE

C     Update PS depending on entropy tracking.
      IF(CMPRES)THEN
C       Compressible flow.  S-momentum is necessary if mu > 0.
        IF(ITER.EQ.0 .OR. SUBSON)CALL ENTRAK
      ELSE
C       Incompressible flow.
C       Rho V is calculated in RMS routine and is saved in
C       either RWXS(isf) or RWXS(1) etc depending on value of MNNS
C       and ISAVE.
        ISAVEO = ISAVE
        ISAVE = 1
        DO ISF=1,NSF
          M = ISF - NB
          IF(M.LE.0)THEN
C           Boundary.  Apply S.
            IF(ISUP.EQ.0)THEN
              S(ISF) = SUP(1)
            ELSE
              S(ISF) = SUP(ISF)
C             RSAB is used to calculate A0 and B0 which are PSI1 and PSI2
C             on the S face.
              CALL RSAB(ISF)
              PSI1UT(ISF) = A0
              PSI2UT(ISF) = B0
              IF(ISF.EQ.NB)THEN
C               Check the upstream psi1 psi2 table.  If TABOK is true,
C               then it is monotonic.
                CALL CHKTAB(PSI1UT,PSI2UT,NI-1,NJ-1,MSGLVO,TABOK)
              END IF
            END IF
          ELSE
C           Check if applied to upstream volume.
            IF(ISSMAP(M) .AND. (ITER.NE.0))GOTO 20
            IF(ISUP.EQ.0)THEN
              S(ISF) = SUP(1)
            ELSE
              IF(TABOK)THEN
C               RSAB is used to calculate A0 and B0 which are PSI1 and PSI2
C               on the S face.
                CALL RSAB(ISF)
                CALL SIINT(PSI1UT,PSI2UT,SUP,NI,NJ,A0,B0,S(ISF))
              ELSE
C               Upstream psi1-psi2 table is not monotonic.  Take value
C               from upstream i-j.
                S(ISF) =  SUP( MOD(ISF-1,NB) + 1 )
              END IF
            END IF
          END IF
C         Get rho V.
          CALL RMS(ISF,1.)
          CALL RRTHFC(R,X1)
          OMEGRS = OMG*OMG*R*R
```

```
            IF(MNNS.EQ.1)THEN
              CALL PFRVSI(RWXS(1),RWYS(1),RWZS(1),S(ISF),ROC,
     1          OMEGRS,PS(ISF))
            ELSE
              CALL PFRVSI(RWXS(ISF),RWYS(ISF),RWZS(ISF),S(ISF),ROC,OMEGRS,
     1          PS(ISF))
            END IF
20          CONTINUE
          END DO
          ISAVE = ISAVEO
        END IF

C       For ITER=0, initialize Pa and Pb since Ps has been changed.
        IF(ITER.EQ.0)CALL PAPBI

        CALL GETRHS

        FMIN = 0.
        DO IR=1,NSV
          FMIN = FMIN + FVEC(IR)*FVEC(IR)
        END DO

        FMIN = 0.5*FMIN

        IF(MSGLVR.EQ.1)THEN
          PRINT*,' '
          PRINT*,'LAMBDA, FMIN in RHSSQ: ',LAMBDA,FMIN
          CALL PRSTAT
        END IF

        RETURN
        END

        SUBROUTINE ENTRAK
C       This routine tracks the entropy, applies the upstream entropy
C       boundary condition, and determines whether the s-momentum equation
C       should be applied for compressible flow.

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'MESSAG.FOR'
        INCLUDE 'FACE.FOR'
        INCLUDE 'CLSCOM.FOR'

        INTEGER ISAVEO,ISF,M,I,J,K,IERR,NERR,IJ
        REAL PPREV,RWX,RWY,RWZ,RWS,RHO,OR2,R,X1,DELS
        REAL SC,RHOC,HSM,MACHSQ,PSQ
        LOGICAL TABOK,TABOKC
        INTEGER NSMOMV

C       Compressible flow.  S-momentum is necessary if mu > 0.

C       Rho V is calculated in RMS routine and is saved in
C       either RWXS(isf) or RWXS(1) etc depending on value of MNNS
C       and ISAVE.

        ISAVEO = ISAVE
        ISAVE = 1

C       Initialize the number of errors in PFRVSC.
        NERR = 0

C       Upstream faces.  Apply entropy condition and set up upstream psi
C       table.
        DO ISF=1,NB
          PPREV = PS(ISF)
          IF(ISUP.EQ.0)THEN
            S(ISF) = SUP(1)
          ELSE
            S(ISF) = SUP(ISF)
          END IF
C         RSAB is used to calculate A0 and B0 which are PSI1 and PSI2
C         on the S face.
          CALL RSAB(ISF)
          PSI1UT(ISF) = A0
          PSI2UT(ISF) = B0
C         Get rho V.  H is also determined in RMSC by IMS.
          CALL RMSC(ISF,1.)
          CALL RRTHFC(R,X1)
          OR2 = OMG*OMG*R*R/2.
          IERR = 1
          IF(MNNS.EQ.1)THEN
            CALL PFRVSC(RWXS(1),RWYS(1),RWZS(1),S(ISF),H,OR2,
     1           PREF,HREF,GAM, PS(ISF),2,IERR)
          ELSE
            CALL PFRVSC(RWXS(ISF),RWYS(ISF),RWZS(ISF),S(ISF),H,OR2,
     1           PREF,HREF,GAM, PS(ISF),2,IERR)
          END IF
          IF(IERR.EQ.1)THEN
            NERR = NERR + 1
          END IF
        END DO

        IF(NERR.NE.0)THEN
          PRINT*,NERR,' errors in calculating pressure upstream.'
        END IF

C       Do nothing if S momentum is always applied.
        IF(SMOMAP .AND. (ITER.NE.0))RETURN

C       Check the upstream psi1 psi2 table.  If TABOK is true,
C       then it is monotonic.  Only do this if ISUP.ne.0.
        IF(ISUP.EQ.0)THEN
          TABOK = .FALSE.
        ELSE
          CALL CHKTAB(PSI1UT,PSI2UT,NI-1,NJ-1,MSGLV0,TABOK)
        END IF

C       Set up the entropy correction table.
        DO ISF=1,NB
          PSI1CT(ISF) = PSI1UT(ISF)
          PSI2CT(ISF) = PSI2UT(ISF)
          CST(ISF) = 0.
        END DO

        TABOKC = TABOK

C       Rest of the s faces at k stations.

C       NSMOMV is the number of volumes which apply the s-momentum eq.
        NSMOMV = 0

        DO K=2,NK-1

          DO J=1,NJ-1
            DO I=1,NI-1
              IJ = (J-1)*(NI-1) + I
              ISF = (K-1)*(NI-1)*(NJ-1) + IJ
              M = ISF - NB
              PPREV = PS(ISF)
C             Get rho V and Mach number.  H is also determined in RMSC by IMS.
              CALL RMSC(ISF,1.)
            CALL RRTHFC(R,X1)
            OR2 = OMG*OMG*R*R/2.
            IF(MNNS.EQ.1)THEN
              RWX = RWXS(1)
              RWY = RWYS(1)
              RWZ = RWZS(1)
              RHO = RHOS(1)
            ELSE
              RWX = RWXS(ISF)
              RWY = RWYS(ISF)
              RWZ = RWZS(ISF)
              RHO = RHOS(ISF)
            END IF
            RWS = RWX**2 + RWY**2 + RWZ**2

            IF(MSS(ISF).GE.MCS .OR. MSS(ISF-NB).GE.MCS .OR.
     1        SMMODE)THEN
C             The upstream volume should have the S momentum applied.
              ISSMAP(M) = .TRUE.
              NSMOMV = NSMOMV + 1
            ELSE
C             The entropy should be convected.
              ISSMAP(M) = .FALSE.
            END IF

C           RSAB is used to calculate AO and BO which are PSI1 and PSI2
C           on the S face.
            CALL RSAB(ISF)
            PSI1S(IJ) = AO
            PSI2S(IJ) = BO
C           Get the entropy from the upstream table.
            IF(ISUP.EQ.0)THEN
              SFUP(IJ) = SUP(1)
            ELSE
              IF(TABOK)THEN
                CALL SIINT(PSI1UT,PSI2UT,SUP,NI,NJ,AO,BO,SFUP(IJ))
              ELSE
C               Upstream psi1-psi2 table is not monotonic.  Take value
C               from upstream i-j.
                SFUP(IJ) =  SUP(IJ)
              END IF
            END IF

C           Check if applied to upstream volume.
            IF(ISSMAP(M) .AND. (ITER.NE.0))THEN
C             Calculate the entropy and correction.
              S(ISF) = SC(RWS,RHO,PS(ISF),H,OR2,PREF,HREF,GAM)
              GOTO 10
            END IF

C           Interpolate the correction term.
            IF(TABOKC)THEN
              CALL SIINT(PSI1CT,PSI2CT,CST,NI,NJ,AO,BO,DELS)
            ELSE
C             Correction psi1-psi2 table is not monotonic or ISUP=0.
C             Take value from just upstream face if any s-momentum eqs
C             applied.
              IF(NSMOMV.EQ.O .AND. ISUP.EQ.O)THEN
                DELS = 0.
              ELSE
                DELS = S(ISF-NB) - SFUP(IJ)
              END IF
            END IF

            S(ISF) = SFUP(IJ) + DELS
```

```
C             Get the pressure based on this entropy.
              IERR = 0
              CALL PFRVSC(RWX,RWY,RWZ,S(ISF),H,OR2,
     1          PREF,HREF,GAM, PS(ISF),2,IERR)
              NERR = NERR + IERR

C             Make sure this pressure does not make (Mach number)^2 > mcs.
              RHO = RHOC(RWS,PS(ISF),H,OR2,GAM)
              MSS(ISF) = RWS/(GAM*RHO*PS(ISF))

              IF(MSS(ISF).GE.MCS .OR. IERR.EQ.1)THEN
                IF(ITER.NE.0)THEN
C                 The upstream volume should have the S momentum applied
C                 because entropy convection equation is not satisfied.
                  ISSMAP(M) = .TRUE.
                  NSMOMV = NSMOMV + 1
C                 The updated pressure should be used.
                  PS(ISF) = PPREV
                  RHO = RHOC(RWS,PS(ISF),H,OR2,GAM)
C                 Calculate the entropy.
                  S(ISF) = SC(RWS,RHO,PS(ISF),H,OR2,PREF,HREF,GAM)
                ELSE
C                 Use the pressure for Mach squared = MCS - 0.001,
C                 RWS, and H(which is rothalpy).
                  MACHSQ = MCS - 0.001
                  HSM = (H + OR2)/(1. + (GAM-1.)/2.*MACHSQ)
                  PSQ = RWS*(GAM-1.)*HSM/(GAM**2*MACHSQ)
                  PS(ISF) = SQRT(PSQ)
                  MSS(ISF) = MACHSQ
                END IF
              END IF

10            CONTINUE
            END DO
          END DO

C         Set up this stations correction table.
          DO IJ=1,NB
            PSI1CT(IJ) = PSI1S(IJ)
            PSI2CT(IJ) = PSI2S(IJ)
            CST(IJ) = S((K-1)*NB + IJ) - SFUP(IJ)
          END DO
C         Check the upstream psi1 psi2 table.  If TABOKC is true,
C         then it is monotonic.
          IF(NSMOMV.EQ.0 .AND. ISUP.EQ.0)THEN
C           Table not needed.
            TABOKC = .FALSE.
          ELSE
            CALL CHKTAB(PSI1CT,PSI2CT,NI-1,NJ-1,MSGLVO,TABOKC)
          END IF

        END DO

        ISAVE = ISAVEO

        IF(NERR.NE.0 .AND. MSGLVO.EQ.1)THEN
          PRINT*,'There were ',NERR,' errors in PFRVSC.'
        END IF

        RETURN
        END

        REAL FUNCTION RHOC(RWS,P,H,OR2,GAM)
C       This function determines the compressible density as a function
C       of (rho W)^2 : RWS, p : P, rothalpy : H,
C       (Omega r)^2/2 : OR2, and GAM.  This calculation was taken
```

```
C       from ENTC.FOR which was generated by SMP.
C
        REAL RWS,P,H,OR2,GAM

     RHOC = 0.5E0 * ((GAM * P + (GAM ** 2 * P ** 2 + 2 * RWS * ((-1) +
    $    GAM) ** 2 * (H + OR2)) ** 0.5E0) / (((-1) + GAM) * (H + OR2))
    $    )

        RETURN
        END

        REAL FUNCTION SC(RWS,RHO,P,H,OR2,PREF,HREF,GAM)
C       This function determines the compressible entropy as a function
C       of (rho W)^2 : RWS, rho : RHO, p : P, rothalpy : H,
C       (Omega r)^2/2 : OR2, PREF, HREF and GAM.  This calculation was taken
C       from ENTC.FOR which was generated by SMP.
C
        REAL RWS,RHO,P,H,OR2,PREF,HREF,GAM

     SC = (GAM * (LOG((H + OR2 + (-0.5E0) * (RWS / RHO ** 2))
    $    / HREF) / 1.)) / ((-1) + GAM) + (-1) * (LOG(P / PREF) /
    $    1.)

        RETURN
        END

        SUBROUTINE PAPBI
C       This routine initializes the Pa and Pb arrays by interpolating the
C       Ps values which are updated if entropy tracking is used.

        INCLUDE '3DCOM.FOR'

        INTEGER I,J,K,IAF,ISF1,ISF2,ISF3,ISF4,IBF

        DO K=2,NK-1
          DO J=1,NJ-1
            I = 1
            IAF = (K-2)*NI*(NJ-1) + (J-1)*NI + I
            ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
            ISF2 = ISF1 - (NI-1)*(NJ-1)
            PA(IAF) = 0.5*(PS(ISF1) + PS(ISF2))

            DO I=2,NI-1
              IAF = (K-2)*NI*(NJ-1) + (J-1)*NI + I
              ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
              ISF2 = ISF1 - (NI-1)*(NJ-1)
              ISF3 = ISF1 - 1
              ISF4 = ISF3 - (NI-1)*(NJ-1)
              PA(IAF) = 0.25*(PS(ISF1) + PS(ISF2) + PS(ISF3) + PS(ISF4))
            END DO

            I = NI
            IAF = (K-2)*NI*(NJ-1) + (J-1)*NI + I
            ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I - 1
            ISF2 = ISF1 - (NI-1)*(NJ-1)
            PA(IAF) = 0.5*(PS(ISF1) + PS(ISF2))

C           Repeating condition.
            IF(K.LE.KLE .OR. K.GE.KTE)THEN
              PA(IAF) = 0.5*(PA(IAF) + PA(IAF-(NI-1)))
              PA(IAF-(NI-1)) = PA(IAF)
            END IF
          END DO


          DO I=1,NI-1
```

```
            J = 1
            IBF = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
            ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
            ISF2 = ISF1 - (NI-1)*(NJ-1)
            PB(IBF) = 0.5*(PS(ISF1) + PS(ISF2))

            DO J=2,NJ-1
              IBF = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
              ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
              ISF2 = ISF1 - (NI-1)*(NJ-1)
              ISF3 = ISF1 - (NI-1)
              ISF4 = ISF3 - (NI-1)*(NJ-1)
              PB(IBF) = 0.25*(PS(ISF1) + PS(ISF2) + PS(ISF3) + PS(ISF4))
            END DO

            J = NJ
            IBF = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
            ISF1 = (K-1)*(NI-1)*(NJ-1) + (J-1-1)*(NI-1) + I
            ISF2 = ISF1 - (NI-1)*(NJ-1)
            PB(IBF) = 0.5*(PS(ISF1) + PS(ISF2))
          END DO
        END DO

        RETURN
        END

        SUBROUTINE CHKTAB(PSI1,PSI2,NI,NJ,MSGLVL,TABOK)
C       This routine checks that the 2-D table of psi1-psi2 is monotonic.
C       If it is, then TABOK=.true.  Otherwise = .false.
C       This table is NI by NJ with I changing fastest.
C       If MSGLVL=0 then only minimal error messages are printed,
C       else if MSGLVL=1, the entire table is printed.
        REAL PSI1(1),PSI2(1)
        INTEGER NI,NJ,MSGLVL
        LOGICAL TABOK

        INTEGER I,J,N

        TABOK = .TRUE.

C       Check psi1.
        N = 0
        DO J=1,NJ
          N = N + 1
          DO I=2,NI
            N = N + 1
            IF(PSI1(N).LT.PSI1(N-1))THEN
              PRINT*,'Error in psi1-psi2 table for PSI1 i=, j=',I,J
              TABOK = .FALSE.
            END IF
          END DO
        END DO

C       Check psi2.
        N = NI
        DO J=2,NJ
          DO I=1,NI
            N = N + 1
            IF(PSI2(N).LT.PSI2(N-NI))THEN
              PRINT*,'Error in psi1-psi2 table for PSI2 i=, j=',I,J
              TABOK = .FALSE.
            END IF
          END DO
        END DO

        IF(.NOT.TABOK .AND. MSGLVL.EQ.1)THEN
```

```
C         Print psi1.
          PRINT*,'PSI1 TABLE:'
          N = 0
          DO J=1,NJ
            DO I=1,NI
              N = N + 1
              PRINT*,'I,J,PSI1',I,J,PSI1(N)
            END DO
          END DO

C         Print psi2.
          PRINT*,'PSI2 TABLE:'
          N = 0
          DO J=1,NJ
            DO I=1,NI
              N = N + 1
              PRINT*,'I,J,PSI2',I,J,PSI2(N)
            END DO
          END DO
        END IF

        RETURN
        END

        SUBROUTINE BLDGP
C       This routine updates the blade grid points using the leading edge
C       movement, blade splines, and grid spacing arrays.

        INCLUDE '3DCOM.FOR'

        INTEGER I,J,K,N,KB
        REAL SS,SP
        REAL SEVAL

        DO J=1,NJ
          DO K=KLE,KTE
            KB = K - KLE + 1

C           Arc length on lower surface of passage.
            SS = SBLE(J)*(1. - SGL(KB,J))
            I = 1
            N = (K-1)*NI*NJ + (J-1)*NI + I
            X1N(N) = SEVAL(SS,X1BLD(1,J),DX1DSB(1,J),SBLD(1,J),NPBLD(J))
            EMPRM(N) = SEVAL(SS,MPBLD(1,J),DMPDSB(1,J),SBLD(1,J),NPBLD(J))
            CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1          EMPRM(N),X2N(N),1,0)
            CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1          EMPRM(N),X3N(N),1,0)

C           Arc length on upper surface of passage.
            SP = SBLE(J) + (SBLD(NPBLD(J),J) - SBLE(J))*SGU(KB,J)
            I = NI
            N = (K-1)*NI*NJ + (J-1)*NI + I
            X1N(N) = SEVAL(SP,X1BLD(1,J),DX1DSB(1,J),SBLD(1,J),NPBLD(J))
     1          + PITCH
            EMPRM(N) = SEVAL(SP,MPBLD(1,J),DMPDSB(1,J),SBLD(1,J),NPBLD(J))
            CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1          EMPRM(N),X2N(N),1,0)
            CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1          EMPRM(N),X3N(N),1,0)
          END DO
        END DO

        RETURN
        END
```

```
      SUBROUTINE REDEQ(SDC,PMSP,IPMSP,NPDROW)
C     This routine sets up the reduced equations.
C     SDC, PMSP, IPMSP, and NPDROW are the entropy matrix.
      REAL SDC(1),PMSP(1)
      INTEGER IPMSP(1),NPDROW(1)

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER NEQCAS
      COMMON /CASRT/NEQCAS

      INTEGER IPP,I,J,NAN,K,L,ICOL,ISF,IPSI1U,IPSI1D,I1E,IDTH
      REAL COF

C     IPSI1U and IPSI1D are the upstream and downstream range of column
C     numbers where the delta theta replaces psi1 or leading edge arc
C     length replaces psi1.
C     Column must be downstream of k=1, j=1, i=1.
      K = 1
      J = 1
      I = 1
      I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
      IPSI1U = IPSI10 +  I1E
C     Column must be upstream of k=nk, j=nj-1, i=1.
      K = NK
      J = NJ - 1
      I = 1
      I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
      IPSI1D = IPSI10 +  I1E

      NEQCAS = NSV

      DO IR=1,NEQ
        CALL EACHEQ
C       Substitute entropy for pressure.
C       Check each column to see if it contains a pressure.
C       These eqs cannot contain pt.  NAN is NA New.
        NAN = NA
        DO L=1,NA
          ICOL = ICOLA(L)
          IF(ICOL.GT.IPSO .AND. ICOL.LE.IPAO)THEN
C           A pressure term.  Eliminate it.
            ISF = ICOL - IPSO
            COF = COFA(L)
            COFA(L) = SDC(ISF)*COF
            IF(ISF.EQ.1)THEN
              IPP = 1
            ELSE
              IPP = NPDROW(ISF-1) + 1
            END IF
            DO K=IPP,NPDROW(ISF)
              CALL ADCOFS(PMSP(K)*COF,IPMSP(K),COFA,ICOLA,NAN)
            END DO
          END IF
        END DO
        NA = NAN

C       Eliminate psi1(i=1,j=1,nj-1,k=1,nk) and psi1(i=ni,j=1,nj-1,k=ksble)
C       in matrix.  For psi1(i=1), this is 0 and should be set.  This
C       eq can be eliminated by setting coef to 0.  For psi1(ni,ksble),
C       the equation is to set psi1=psi1_max unless IDWNBC=2.  Then
C       the (delta psi1)_xi3=0 eq is used.  So if IDWNBC=2, set the column to
C       the one just downstream.  Put delta theta and sblei in
```

```
C         the location for these psi1's by changing the column number.
          DO L=1,NA
            ICOL = ICOLA(L)
            IF(ICOL.GE.IPSI1U .AND. ICOL.LE.IPSI1D)THEN
C             Possibly psi1 on stagnation stream surface with i=1 or
C             leading edge arc length i=ni,k=ksble.
              I1E = ICOL
              K = (I1E - 1)/(NI*(NJ-1)) + 1
              J = (I1E - (K-1)*NI*(NJ-1) - 1)/NI + 1
              I = I1E - (K-1)*NI*(NJ-1) - (J-1)*NI
              IF(I.EQ.1)THEN
                COFA(L) = 0.
              ELSE IF(I.EQ.NI .AND. K.EQ.KSBLE)THEN
                IF(IDWNBC.EQ.2)THEN
                  ICOLA(L) = ICOL + NI*(NJ-1)
                ELSE
                  COFA(L) = 0.
                END IF
              END IF
            ELSE IF(ICOL.GT.IDTHO)THEN
              IF(ICOL.GT.ISBLEO)THEN
C               Leading edge arc length.
                J = ICOL - ISBLEO
                I = NI
                K = KSBLE
                I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
                ICOLA(L) = IPSI1O + I1E
              ELSE
C               Delta theta.
                IDTH = ICOL - IDTHO
                K = (IDTH - 1)/(NJ-1) + 1
                J = IDTH - (K-1)*(NJ-1)
                I = 1
                I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
                ICOLA(L) = IPSI1O + I1E
              END IF
            END IF
          END DO

C         Use IWS as dummy storage.
          CALL CASRTS(COFA,ICOLA,NA,IR,WS(ISSP+1),WS(IISP+1),
        1        WS(INDROW+1),IWS(IWSMSU+1))
        END DO

C       Print different rows of the matrix.
C       CALL PRTMAT(NEQ,WS(ISSP+1),WS(IISP+1),WS(INDROW+1),WS(IRHS+1))

        RETURN
        END

        SUBROUTINE REDEQP
C       This routine sets up the reduced equations which contain pressure
C       and there is now substitution for entropy.

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER NEQCAS
        COMMON /CASRT/NEQCAS

        INTEGER I,J,K,L,ICOL,IPSI1U,IPSI1D,I1E,IDTH

C       IPSI1U and IPSI1D are the upstream and downstream range of column
C       numbers where the delta theta replaces psi1 or leading edge arc
C       length replaces psi1.
```

```
C       Column must be downstream of k=1, j=1, i=1.
        K = 1
        J = 1
        I = 1
        I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
        IPSI1U = IPSI10 +  I1E
C       Column must be upstream of k=nk, j=nj-1, i=1.
        K = NK
        J = NJ - 1
        I = 1
        I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
        IPSI1D = IPSI10 +  I1E

        NEQCAS = NSV

        DO IR=1,NEQ
          CALL EACHEQ

C         Eliminate psi1(i=1,j=1,nj-1,k=1,nk) and psi1(i=ni,j=1,nj-1,k=ksble)
C         in matrix.  For psi1(i=1), this is 0 and should be set.  This
C         eq can be eliminated by setting coef to 0.  For psi1(ni,ksble),
C         the equation is to set psi1=psi1_max unless IDWNBC=2.  Then
C         the (delta psi1)_xi3=0 eq is used.  So if IDWNBC=2, set the column to
C         the one just downstream.  Put delta theta and sblei in
C         the location for these psi1's by changing the column number.
          DO L=1,NA
            ICOL = ICOLA(L)
            IF(ICOL.GE.IPSI1U .AND. ICOL.LE.IPSI1D)THEN
C             Possibly psi1 on stagnation stream surface with i=1 or
C             leading edge arc length i=ni,k=ksble.
              I1E = ICOL
              K = (I1E - 1)/(NI*(NJ-1)) + 1
              J = (I1E - (K-1)*NI*(NJ-1) - 1)/NI + 1
              I = I1E - (K-1)*NI*(NJ-1) - (J-1)*NI
              IF(I.EQ.1)THEN
                COFA(L) = 0.
              ELSE IF(I.EQ.NI .AND. K.EQ.KSBLE)THEN
                IF(IDWNBC.EQ.2)THEN
                  ICOLA(L) = ICOL + NI*(NJ-1)
                ELSE
                  COFA(L) = 0.
                END IF
              END IF
            ELSE IF(ICOL.GT.IDTHO)THEN
              IF(ICOL.GT.ISBLEO)THEN
C               Leading edge arc length.
                J = ICOL - ISBLEO
                I = NI
                K = KSBLE
                I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
                ICOLA(L) = IPSI10 + I1E
              ELSE
C               Delta theta.
                IDTH = ICOL - IDTHO
                K = (IDTH - 1)/(NJ-1) + 1
                J = IDTH - (K-1)*(NJ-1)
                I = 1
                I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
                ICOLA(L) = IPSI10 + I1E
              END IF
            END IF
          END DO

C         Use IWS as dummy storage.
          CALL CASRTS(COFA,ICOLA,NA,IR,WS(ISSP+1),WS(IISP+1),
     1          WS(INDROW+1),IWS(IWSMSU+1))
```

```
      END DO

C     Print different rows of the matrix.
C     CALL PRTMAT(NEQ,WS(ISSP+1),WS(IISP+1),WS(INDROW+1),WS(IRHS+1))

      RETURN
      END


      SUBROUTINE GETRHS
C     This routine gets the rhs of the original matrix.

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER M,ISF,IAF,I,J,K,IJ,IE1DS,IE2DS,I1E,I2E
      REAL ARHS

C     Initialize error stats.
      ESMMAX = 0.
      ESMAVG = 0.
      EAMMAX = 0.
      EAMAVG = 0.
      EBMMAX = 0.
      EBMAVG = 0.
      EPAMAX = 0.
      EPAAVG = 0.
      EPBMAX = 0.
      EPBAVG = 0.
      EBCMAX = 0.
      EBCAVG = 0.

C     Each boundary condition right hand side is stored in FVEC(IR)
C     except for the upstream entropy and radial equilibrium eq.
C     The row number is calculated in the volume routines and these
C     right hand sides are stored in FVEC.  The upstream entropy  and
C     radial equilibrium bc right hand sides are stored in RESX.

C     Psi2=0 bc at n2e - (ni-1)nj + 1.
      IR = 1
      CALL RBPSI2(N2E-(NI-1)*NJ+1,PSI2DN(1))
      ARHS = ABS(FVEC(IR))
      IF(ARHS.GT.EBCMAX)THEN
        EBCMAX = ARHS
        IBCMAX = 4
      END IF
      EBCAVG = EBCAVG + ARHS

C     Volume Equations.
      IF(CMPRES)THEN
C       Compressible flow.
        DO M=1,NM
C         The momentum equations.
          CALL RMC(M)

C         S-momentum errors.  If entropy tracked instead, set to 0.
          IF(.NOT.ISSMAP(M))FVEC(5*M-3) = 0.

          ARHS = ABS(FVEC(5*M-3))
          IF(ARHS.GT.ESMMAX)THEN
            ESMMAX = ARHS
            MSMMAX = M
          END IF
          ESMAVG = ESMAVG + ARHS
```

```
C           A-momentum errors.
            ARHS = ABS(FVEC(5*M-2))
            IF(ARHS.GT.EAMMAX)THEN
              EAMMAX = ARHS
              MAMMAX = M
            END IF
            EAMAVG = EAMAVG + ARHS

C           B-momentum errors.
            ARHS = ABS(FVEC(5*M-1))
            IF(ARHS.GT.EBMMAX)THEN
              EBMMAX = ARHS
              MBMMAX = M
            END IF
            EBMAVG = EBMAVG + ARHS

C           Auxilliary Pressure equations.
            CALL RAUXPC(M)

C           PA errors.
            FVEC(5*M) = RESX
            ARHS = ABS(FVEC(5*M))
            IF(ARHS.GT.EPAMAX)THEN
              EPAMAX = ARHS
              MPAMAX = M
            END IF
            EPAAVG = EPAAVG + ARHS

C           PB errors.
            FVEC(5*M+1) = RESY
            ARHS = ABS(FVEC(5*M+1))
            IF(ARHS.GT.EPBMAX)THEN
              EPBMAX = ARHS
              MPBMAX = M
            END IF
            EPBAVG = EPBAVG + ARHS

          END DO
        ELSE
C         Incompressible flow.
          DO M=1,NM
C           The momentum equations.
            CALL RM(M)

C           S-momentum errors.  If entropy tracked instead, set to 0.
            IF(.NOT.ISSMAP(M))FVEC(5*M-3) = 0.

            ARHS = ABS(FVEC(5*M-3))
            IF(ARHS.GT.ESMMAX)THEN
              ESMMAX = ARHS
              MSMMAX = M
            END IF
            ESMAVG = ESMAVG + ARHS

C           A-momentum errors.
            ARHS = ABS(FVEC(5*M-2))
            IF(ARHS.GT.EAMMAX)THEN
              EAMMAX = ARHS
              MAMMAX = M
            END IF
            EAMAVG = EAMAVG + ARHS

C           B-momentum errors.
            ARHS = ABS(FVEC(5*M-1))
            IF(ARHS.GT.EBMMAX)THEN
              EBMMAX = ARHS
              MBMMAX = M
            END IF
            EBMAVG = EBMAVG + ARHS

C           Auxilliary Pressure equations.
            CALL RAUXP(M)

C           PA errors.
            FVEC(5*M) = RESX
            ARHS = ABS(FVEC(5*M))
            IF(ARHS.GT.EPAMAX)THEN
              EPAMAX = ARHS
              MPAMAX = M
            END IF
            EPAAVG = EPAAVG + ARHS

C           PB errors.
            FVEC(5*M+1) = RESY
            ARHS = ABS(FVEC(5*M+1))
            IF(ARHS.GT.EPBMAX)THEN
              EPBMAX = ARHS
              MPBMAX = M
            END IF
            EPBAVG = EPBAVG + ARHS

          END DO
        END IF

C       Psi1 downstream bc.
        IR = 5*NM + 1
        IE1DS = N1E - NI*(NJ-1)
        DO J=1,NJ-1
          I = 1
          IR = IR + 1
          IJ = (J-1)*NI + I
          IF(KTE.GT.NK)THEN
C           Wall.  Delta theta = 0.
            CALL RDTHO((NK-1)*(NJ-1) + J)
          ELSE
C           Wake.  Delta Pa equation is used in periodic reduced A momentum
C           eq.  which is applied as the delta theta eq.  Therefore aply
C           psi1 bc here.
            CALL RBPSI1(IE1DS+IJ,PSI1DN(IJ))
          END IF
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 4
          END IF
          EBCAVG = EBCAVG + ARHS

          DO I=2,NI-1
            IR = IR + 1
            IF(IDWNBC.EQ.0)THEN
C             Set psi1.
              IJ = (J-1)*NI + I
              CALL RBPSI1(IE1DS+IJ,PSI1DN(IJ))
            ELSE IF(IDWNBC.EQ.1)THEN
              IF(J.EQ.NJ-1)THEN
C               Set pressure.
                IJ = (J-1)*(NI-1) + I - 1
                CALL RBPS(NSF-NB+IJ,PDWN(IJ))
              ELSE
C               Set (delta psi1)_xi2 = 0.
                IJ = (J-1)*NI + I
                CALL RBDP1(IE1DS+IJ,IE1DS+IJ+NI)
              END IF
            ELSE IF(IDWNBC.EQ.2)THEN
C             Set pressure.
              IJ = (J-1)*(NI-1) + I - 1
              CALL RBPS(NSF-NB+IJ,PDWN(IJ))
            ELSE IF(IDWNBC.EQ.3 .OR. IDWNBC.EQ.5)THEN
              IF(J.EQ.NJ-1)THEN
cC              Set delta pressure in xi1 = 0.
c               IJ = (J-1)*(NI-1) + I - 1
c               CALL RBDPS(NSF-NB+IJ,NSF-NB+IJ+1)
                    call rbdps1(i-1,j)
              ELSE
C               Set (delta psi1)_xi2 = 0.
                IJ = (J-1)*NI + I
                CALL RBDP1(IE1DS+IJ,IE1DS+IJ+NI)
              END IF
            ELSE IF(IDWNBC.EQ.4 .OR. IDWNBC.EQ.6)THEN
C             Set delta pressure in xi1 = 0.
              IJ = (J-1)*(NI-1) + I - 1
              CALL RBDPS(NSF-NB+IJ,NSF-NB+IJ+1)
            ELSE
C             IDWNBC < 0.
              IF(J.EQ.-IDWNBC)THEN
C               Set pressure at j=japplied.
                IJ = (J-1)*(NI-1) + I - 1
                CALL RBPS(NSF-NB+IJ,PDWN(1))
              ELSE IF(J.GT.-IDWNBC)THEN
C               Set (delta psi1)_xi2 = 0 from this psi1 to the one below.
                IJ = (J-1)*NI + I
                CALL RBDP1(IE1DS+IJ,IE1DS+IJ-NI)
              ELSE
C               Set (delta psi1)_xi2 = 0 from this psi1 to the one above.
                IJ = (J-1)*NI + I
                CALL RBDP1(IE1DS+IJ,IE1DS+IJ+NI)
              END IF
            END IF
            ARHS = ABS(FVEC(IR))
            IF(ARHS.GT.EBCMAX)THEN
              EBCMAX = ARHS
              IBCMAX = 4
            END IF
            EBCAVG = EBCAVG + ARHS
          END DO

          I = NI
          IR = IR + 1
          IJ = (J-1)*NI + I
          IF(IDWNBC.EQ.2)THEN
            IF(J.EQ.NJ-1)THEN
C             Set pressure.
              IJ = (J-1)*(NI-1) + I - 1
              CALL RBPS(NSF-NB+IJ,PDWN(IJ))
            ELSE
C             Set (delta psi1)_xi2 = 0.
              CALL RBDP1(IE1DS+IJ,IE1DS+IJ+NI)
            END IF
          ELSE
C           PSI1DN(IJ) is psi1_max here.
            CALL RBPSI1(IE1DS+IJ,PSI1DN(IJ))
          END IF
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 4
          END IF
          EBCAVG = EBCAVG + ARHS
```

```
      END DO
C     Psi2 downstream bc.  Except for at n2e - (ni-1)nj - 1.
      IE2DS = N2E - (NI-1)*NJ
      DO J=1,NJ
        DO I=1,NI-1
          IJ = (J-1)*(NI-1) + I
          IF(I.EQ.1 .AND. J.EQ.1)THEN
            GOTO 10
          ELSE IF(J.EQ.1)THEN
            IR = IR + 1
C           PSI2DN(IJ) is 0.
            CALL RBPSI2(IE2DS+IJ,PSI2DN(IJ))
          ELSE IF(J.EQ.NJ)THEN
            IR = IR + 1
            IF(IDWNBC.EQ.1 .OR. IDWNBC.LT.0)THEN
              IF(I.EQ.NI-1)THEN
                IJ = (J-2)*(NI-1) + I
                IF(IDWNBC.EQ.1)THEN
C                 Apply pressure bc.
                  CALL RBPS(NSF-NB+IJ,PDWN(IJ))
                ELSE IF(IDWNBC.EQ.-(NJ-1))THEN
C                 Apply pressure bc for j=japlied.
                  CALL RBPS(NSF-NB+IJ,PDWN(1))
                ELSE
C                 Apply dp/dxi2=0.
                  CALL RBDPS(NSF-NB+IJ,NSF-NB+IJ-(NI-1))
                END IF
              ELSE
C               Apply (delta psi2)_xi1 = 0.
                CALL RBDP2(IE2DS+IJ,IE2DS+IJ+1)
              END IF
            ELSE
C             Apply psi2 = psi2_max
              CALL RBPSI2(IE2DS+IJ,PSI2DN(IJ))
            END IF
          ELSE
            IR = IR + 1
            IF(IDWNBC.EQ.0)THEN
C             psi2 is specified.
              CALL RBPSI2(IE2DS+IJ,PSI2DN(IJ))
            ELSE IF(IDWNBC.EQ.1)THEN
C             Set the pressure.
              IJ = (J-2)*(NI-1) + I
              CALL RBPS(NSF-NB+IJ,PDWN(IJ))
            ELSE IF(IDWNBC.EQ.2)THEN
              IF(I.EQ.NI-1)THEN
C               Set the pressure.
                IJ = (J-2)*(NI-1) + I
                CALL RBPS(NSF-NB+IJ,PDWN(IJ))
              ELSE
C               Apply (delta psi2)_xi1 = 0.
                CALL RBDP2(IE2DS+IJ,IE2DS+IJ+1)
              END IF
            ELSE IF(IDWNBC.EQ.4 .OR. IDWNBC.EQ.6)THEN
              IF(I.EQ.NI-1)THEN
                IF(IDWNBC.EQ.4)THEN
C                 Apply the radial equilibrium eq.
                  CALL RRADEQ(I,J-1)
                  FVEC(IR) = RESX
                ELSE
C                 Apply dp/dxi2=0.
                  IJ = (J-1)*(NI-1) + I
                  CALL RBDPS(NSF-NB+IJ,NSF-NB+IJ-(NI-1))
                END IF
              ELSE
```

```
C                 Apply (delta psi2)_xi1 = 0.
                  CALL RBDP2(IE2DS+IJ,IE2DS+IJ+1)
                END IF
              ELSE
C               IDWNBC = 3,5 or < 0.
                IF(IDWNBC.EQ.5)THEN
C                 Apply dp/dxi2=0.
                  IJ = (J-1)*(NI-1) + I
                  CALL RBDPS(NSF-NB+IJ,NSF-NB+IJ-(NI-1))
                ELSE
C                 Apply the radial equilibrium eq.
                  CALL RBDPS2(I,J-1)
C                 CALL RRADEQ(I,J-1)
C                 FVEC(IR) = RESX
                END IF
              END IF
            END IF
            ARHS = ABS(FVEC(IR))
            IF(ARHS.GT.EBCMAX)THEN
              EBCMAX = ARHS
              IBCMAX = 4
            END IF
            EBCAVG = EBCAVG + ARHS
 10         CONTINUE
          END DO
        END DO

C       Upstream angle.  Include delta theta also.
        DO J=1,NJ-1
          I = 1
C         Apply delta theta angle eq. if there is a leading edge.  Otherwise
C         set delta theta to 0.
          IJ = (J-1)*NI + I
          IR = IR + 1
          IF(KLE.GT.1)THEN
            IF(IUPBC.EQ.0)THEN
C             Angle bc.
              CALL RDTHUP(J,ANG1(J),ANG1(J+1))
            ELSE
C             C_theta bc.
              CALL RCTHUP(J)
            END IF
          ELSE
C           Wall.  Delta theta = 0.
            CALL RDTHO(J)
          END IF
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 2
          END IF
          EBCAVG = EBCAVG + ARHS

          DO I=2,NI
            IJ = (J-1)*NI + I
            IR = IR + 1
            CALL RBDP1(IJ,IJ+NI*(NJ-1))
            ARHS = ABS(FVEC(IR))
            IF(ARHS.GT.EBCMAX)THEN
              EBCMAX = ARHS
              IBCMAX = 2
            END IF
            EBCAVG = EBCAVG + ARHS
          END DO
        END DO
```

```
C       Upstream angle. Delta psi2 = 0.
        DO IJ=1,(NI-1)*NJ
          IR = IR + 1
          CALL RBDP2(IJ,IJ+(NI-1)*NJ)
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 2
          END IF
          EBCAVG = EBCAVG + ARHS
        END DO

C       Entropy upstream.
        DO ISF=1,(NI-1)*(NJ-1)
          IR = IR + 1
          IF(ISUP.EQ.0)THEN
            CALL RBENT(ISF,SUP(1))
          ELSE
            CALL RBENT(ISF,SUP(ISF))
          END IF
          FVEC(IR) = RESX
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 1
          END IF
          EBCAVG = EBCAVG + ARHS
        END DO

C               Wall and Stagnation stream surface conditions.

C       Right faces.
        DO K=2,NK-1
          DO J=1,NJ-1
            IR = IR + 1
            I = 1
            IF(K.LT.KLE .OR. K.GT.KTE)THEN
C             Stagnation stream surface.  Psi1 = 0.
C             Psi1 = 0 at right faces.
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
              CALL RBPSI1(I1E,0.)
            ELSE
C             Blade.  Delta theta = 0.
              CALL RDTHO((K-1)*(NJ-1) + J)
            END IF
            ARHS = ABS(FVEC(IR))
            IF(ARHS.GT.EBCMAX)THEN
              EBCMAX = ARHS
              IBCMAX = 3
            END IF
            EBCAVG = EBCAVG + ARHS
          END DO
        END DO

C       Left faces.
        DO K=2,NK-1
          DO J=1,NJ-1
            IR = IR + 1
            I = NI
            IF(THINLE .AND. K.EQ.KSBLE)THEN
C             No leading edge.  Set dsblei=0.
              FVEC(ISBLEO+J) = 0.
            ELSE
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
              IF(IDWNBC.EQ.2)THEN
C               Set (delta psi1)_xi3 = 0.
```

```
              CALL RBDP1(I1E,I1E+NI*(NJ-1))
            ELSE
C             Set psi1=psi1_max.
              CALL RBPSI1(I1E,PSI1MX)
            END IF
          END IF
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 3
          END IF
          EBCAVG = EBCAVG + ARHS
        END DO
      END DO

C     Bottom faces.
      DO K=2,NK-1
        DO I=1,NI-1
          IR = IR + 1
          J = 1
C         Psi2 = 0 at bottom faces.
          I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
          CALL RBPSI2(I2E,0.)
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 3
          END IF
          EBCAVG = EBCAVG + ARHS
        END DO
      END DO

C     Top faces.
      DO K=2,NK-1
        DO I=1,NI-1
          IR = IR + 1
          J = NJ
          I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
          IF(IDWNBC.EQ.1 .OR. IDWNBC.LT.0)THEN
C           (delta psi2)_xi3 = 0.
            CALL RBDP2(I2E,I2E+NJ*(NI-1))
          ELSE
C           Psi2 = psi2_max at top faces.
            CALL RBPSI2(I2E,PSI2MX)
          END IF
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 3
          END IF
          EBCAVG = EBCAVG + ARHS
        END DO
      END DO

C     Stagnation stream surface equations.
      I = 1
      DO K=1,NK
        DO J=1,NJ-1
          IR = IR + 1
          IF(K.EQ.1 .OR. (K.GE.KLE .AND. K.LE.KTE) .OR.
     1        (KTE.GT.NK .AND. K.EQ.NK))THEN
C           Psi1 = 0 at right faces.
            I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
            CALL RBPSI1(I1E,0.)
          ELSE IF(K.LT.KLE)THEN
C           Upstream stream surface.  Delta Pa = 0.
```

```
            IAF = (K-2)*NI*(NJ-1) + (J-1)*NI + I
            CALL RDPA(IAF,IAF+NI-1,0.)
          ELSE
C           Wake. Delta Pa for k-1 = 0.
            IAF = (K-1-2)*NI*(NJ-1) + (J-1)*NI + I
            CALL RDPA(IAF,IAF+NI-1,DPAA((K-2)*(NJ-1) + J))
          END IF
          ARHS = ABS(FVEC(IR))
          IF(ARHS.GT.EBCMAX)THEN
            EBCMAX = ARHS
            IBCMAX = 3
          END IF
          EBCAVG = EBCAVG + ARHS
        END DO
      END DO

C     Moving leading edge.  Delta Pa = 0.
C     If there is not a leading edge, it is set to 0 at the psi1 location
C     i=ni,k=ksble.  Therefore set this psi1 bc here.
      DO J=1,NSBLEI
        IR = IR + 1
        IF(THINLE)THEN
C         No leading edge.
          I1E = (KSBLE-1)*NI*(NJ-1) + (J-1)*NI + NI
          IF(IDWNBC.EQ.2)THEN
C           Set (delta psi1)_xi3 = 0.
            CALL RBDP1(I1E,I1E+NI*(NJ-1))
          ELSE
C           Set psi1=psi1_max.
            CALL RBPSI1(I1E,PSI1MX)
          END IF
        ELSE
          IAF = (KLE-2)*NI*(NJ-1) + (J-1)*NI + 1
          CALL RDPA(IAF,IAF+NI-1,0.)
        END IF
        ARHS = ABS(FVEC(IR))
        IF(ARHS.GT.EBCMAX)THEN
          EBCMAX = ARHS
          IBCMAX = 3
        END IF
        EBCAVG = EBCAVG + ARHS
      END DO

C     Calculate the averages.
      ESMAVG = ESMAVG/NM
      EAMAVG = EAMAVG/NM
      EBMAVG = EBMAVG/NM
      EPAAVG = EPAAVG/NM
      EPBAVG = EPBAVG/NM
      EBCAVG = EBCAVG/(NSV - 5*NM)

      RETURN
      END

      SUBROUTINE GETLHS(AJ,JCN,JDROW)
C     This routine gets the lhs and stores it in the packed matrix.
C     AJ, JCN and JDROW are the original matrix in compact storage.
      REAL AJ(1)
      INTEGER JCN(1),JDROW(1)

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER NEQCAS
      COMMON /CASRT/NEQCAS
      INTEGER M,ISF,IAF,I,J,K,IJ,IE1DS,IE2DS,I1E,I2E

      NEQCAS = NSV

C     Use IWS(IIWSOM) as dummy storage.

C     Psi2=0 bc at n2e - (ni-1)nj + 1.
      IR = 1
      CALL LBPSI2(N2E-(NI-1)*NJ+1,PSI2DN(1))
      CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))

C     Volume Equations.
      IF(CMPRES)THEN
C       Compressible flow.
        DO M=1,NM
C         The momentum equations.
          CALL LMC(M,AJ,JCN,JDROW,WS(IIWSOM))

C         Auxilliary Pressure equations.
          CALL LAUXPC(M)

C         Auxilliary PA equation is in x-momentum arrays.
          IR = 5*M
          CALL CASRTS(COFX,ICOLX,NCX,IR,AJ,JCN,JDROW,IWS(IIWSOM))

C         Auxilliary PB equation is in y-momentum arrays.
          IR = 5*M + 1
          CALL CASRTS(COFY,ICOLY,NCY,IR,AJ,JCN,JDROW,IWS(IIWSOM))
        END DO
      ELSE
C       Incompressible flow.
        DO M=1,NM
C         The momentum equations.
          CALL LM(M,AJ,JCN,JDROW,WS(IIWSOM))

C         Auxilliary Pressure equations.
          CALL LAUXP(M)

C         Auxilliary PA equation is in x-momentum arrays.
          IR = 5*M
          CALL CASRTS(COFX,ICOLX,NCX,IR,AJ,JCN,JDROW,IWS(IIWSOM))

C         Auxilliary PB equation is in y-momentum arrays.
          IR = 5*M + 1
          CALL CASRTS(COFY,ICOLY,NCY,IR,AJ,JCN,JDROW,IWS(IIWSOM))
        END DO
      END IF

C     Psi1 downstream bc.
      IR = 5*NM + 1
      IE1DS = N1E - NI*(NJ-1)
      DO J=1,NJ-1
        I = 1
        IR = IR + 1
        IJ = (J-1)*NI + I
        IF(KTE.GT.NK)THEN
C         Wall.  Delta theta = 0.
          CALL LDTHO((NK-1)*(NJ-1) + J)
        ELSE
C         Wake.  Delta Pa equation is used in periodic reduced A momentum
C         eq.  which is applied as the delta theta eq.  Therefore aply
C         psi1 bc here.
          CALL LBPSI1(IE1DS+IJ,PSI1DN(IJ))
        END IF
        CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
```

```
          DO I=2,NI-1
            IR = IR + 1
            IF(IDWNBC.EQ.0)THEN
C             Set psi1.
              IJ = (J-1)*NI + I
              CALL LBPSI1(IE1DS+IJ,PSI1DN(IJ))
            ELSE IF(IDWNBC.EQ.1)THEN
              IF(J.EQ.NJ-1)THEN
C               Set pressure.
                IJ = (J-1)*(NI-1) + I - 1
                CALL LBPS(NSF-NB+IJ)
              ELSE
C               Set (delta psi1)_xi2 = 0.
                IJ = (J-1)*NI + I
                CALL LBDP1(IE1DS+IJ,IE1DS+IJ+NI)
              END IF
            ELSE IF(IDWNBC.EQ.2)THEN
C             Set pressure.
              IJ = (J-1)*(NI-1) + I - 1
              CALL LBPS(NSF-NB+IJ)
            ELSE IF(IDWNBC.EQ.3 .OR. IDWNBC.EQ.5)THEN
              IF(J.EQ.NJ-1)THEN
cC              Set delta pressure in xi1 = 0.
c               IJ = (J-1)*(NI-1) + I - 1
c               CALL LBDPS(NSF-NB+IJ,NSF-NB+IJ+1)
                    call lbdps1(i-1,j)
              ELSE
C               Set (delta psi1)_xi2 = 0.
                IJ = (J-1)*NI + I
                CALL LBDP1(IE1DS+IJ,IE1DS+IJ+NI)
              END IF
            ELSE IF(IDWNBC.EQ.4 .OR. IDWNBC.EQ.6)THEN
C             Set delta pressure in xi1 = 0.
              IJ = (J-1)*(NI-1) + I - 1
              CALL LBDPS(NSF-NB+IJ,NSF-NB+IJ+1)
            ELSE
C             IDWNBC < 0.
              IF(J.EQ.-IDWNBC)THEN
C               Set pressure at j=japplied.
                IJ = (J-1)*(NI-1) + I - 1
                CALL LBPS(NSF-NB+IJ)
              ELSE IF(J.GT.-IDWNBC)THEN
C               Set (delta psi1)_xi2 = 0 from this psi1 to the one below.
                IJ = (J-1)*NI + I
                CALL LBDP1(IE1DS+IJ,IE1DS+IJ-NI)
              ELSE
C               Set (delta psi1)_xi2 = 0 from this psi1 to the one above.
                IJ = (J-1)*NI + I
                CALL LBDP1(IE1DS+IJ,IE1DS+IJ+NI)
              END IF
            END IF
            CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
          END DO

          I = NI
          IR = IR + 1
          IJ = (J-1)*NI + I
          IF(IDWNBC.EQ.2)THEN
            IF(J.EQ.NJ-1)THEN
C             Set pressure.
              IJ = (J-1)*(NI-1) + I - 1
              CALL LBPS(NSF-NB+IJ)
            ELSE
C             Set (delta psi1)_xi2 = 0.
              CALL LBDP1(IE1DS+IJ,IE1DS+IJ+NI)
              END IF
            ELSE
C             PSI1DN(IJ) is psi1_max here.
              CALL LBPSI1(IE1DS+IJ,PSI1DN(IJ))
            END IF
            CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
          END DO

C         Psi2 downstream bc.  Except for at n2e - (ni-1)nj.
          IE2DS = N2E - (NI-1)*NJ
          DO J=1,NJ
            DO I=1,NI-1
              IJ = (J-1)*(NI-1) + I
              IF(I.EQ.1 .AND. J.EQ.1)THEN
                GOTO 10
              ELSE IF(J.EQ.1)THEN
                IR = IR + 1
C               PSI2DN(IJ) is 0.
                CALL LBPSI2(IE2DS+IJ,PSI2DN(IJ))
              ELSE IF(J.EQ.NJ)THEN
                IR = IR + 1
                IF(IDWNBC.EQ.1 .OR. IDWNBC.LT.0)THEN
                  IF(I.EQ.NI-1)THEN
                    IJ = (J-2)*(NI-1) + I
                    IF(IDWNBC.EQ.1)THEN
C                     Apply pressure bc.
                      CALL LBPS(NSF-NB+IJ)
                    ELSE IF(IDWNBC.EQ.-(NJ-1))THEN
C                     Apply pressure bc for j=japlied.
                      CALL LBPS(NSF-NB+IJ)
                    ELSE
C                     Apply dp/dxi2=0.
                      CALL LBDPS(NSF-NB+IJ,NSF-NB+IJ-(NI-1))
                    END IF
                  ELSE
C                   Apply (delta psi2)_xi1 = 0.
                    CALL LBDP2(IE2DS+IJ,IE2DS+IJ+1)
                  END IF
                ELSE
C                 Apply psi2 = psi2_max
                  CALL LBPSI2(IE2DS+IJ,PSI2DN(IJ))
                END IF
              ELSE
                IR = IR + 1
                IF(IDWNBC.EQ.0)THEN
C                 psi2 is specified.
                  CALL LBPSI2(IE2DS+IJ,PSI2DN(IJ))
                ELSE IF(IDWNBC.EQ.1)THEN
C                 Set the pressure.
                  IJ = (J-2)*(NI-1) + I
                  CALL LBPS(NSF-NB+IJ)
                ELSE IF(IDWNBC.EQ.2)THEN
                  IF(I.EQ.NI-1)THEN
C                   Set the pressure.
                    IJ = (J-2)*(NI-1) + I
                    CALL LBPS(NSF-NB+IJ)
                  ELSE
C                   Apply (delta psi2)_xi1 = 0.
                    CALL LBDP2(IE2DS+IJ,IE2DS+IJ+1)
                  END IF
                ELSE IF(IDWNBC.EQ.4 .OR. IDWNBC.EQ.6)THEN
                  IF(I.EQ.NI-1)THEN
                    IF(IDWNBC.EQ.4)THEN
C                     Apply the radial equilibrium eq.
                      CALL LRADEQ(I,J-1)
                      CALL CASRTS(COFX,ICOLX,NCX,IR,AJ,JCN,JDROW,
     1                 IWS(IIWSOM))
                     GOTO 10
                   ELSE
C                    Apply dp/dxi2=0.
                     IJ = (J-1)*(NI-1) + I
                     CALL LBDPS(NSF-NB+IJ,NSF-NB+IJ-(NI-1))
                   END IF
                 ELSE
C                  Apply (delta psi2)_xi1 = 0.
                   CALL LBDP2(IE2DS+IJ,IE2DS+IJ+1)
                 END IF
               ELSE
C                IDWNBC = 3,5 or < 0.
                 IF(IDWNBC.EQ.5)THEN
C                  Apply dp/dxi2=0.
                   IJ = (J-1)*(NI-1) + I
                   CALL LBDPS(NSF-NB+IJ,NSF-NB+IJ-(NI-1))
                 ELSE
C                  Apply the radial equilibrium eq.
                   CALL LBDPS2(I,J-1)
C                  CALL LRADEQ(I,J-1)
C                  CALL CASRTS(COFX,ICOLX,NCX,IR,AJ,JCN,JDROW,
C    1                  IWS(IIWSOM))
C                  GOTO 10
                 END IF
               END IF
             END IF
             CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
10           CONTINUE
           END DO
         END DO

C        Upstream angle.  Include delta theta also.
         DO J=1,NJ-1
           I = 1
C          Apply delta theta angle eq. if there is a leading edge.  Otherwise
C          set delta theta to 0.
           IJ = (J-1)*NI + I
           IR = IR + 1
           IF(KLE.GT.1)THEN
             IF(IUPBC.EQ.0)THEN
C              Angle bc.
               CALL LDTHUP(J,ANG1(J),ANG1(J+1))
             ELSE
C              C_theta bc.
               CALL LCTHUP(J)
             END IF
           ELSE
C            Wall.  Delta theta = 0.
             CALL LDTHO(J)
           END IF
           CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))

           DO I=2,NI
             IJ = (J-1)*NI + I
             IR = IR + 1
             CALL LBDP1(IJ,IJ+NI*(NJ-1))
             CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
           END DO
         END DO

C        Upstream angle. Delta psi2 = 0.
         DO IJ=1,(NI-1)*NJ
           IR = IR + 1
           CALL LBDP2(IJ,IJ+(NI-1)*NJ)
           CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
```

```
      END DO

C     Entropy upstream.
      DO ISF=1,(NI-1)*(NJ-1)
        IR = IR + 1
        IF(ISUP.EQ.0)THEN
          CALL LBENT(ISF)
        ELSE
          CALL LBENT(ISF)
        END IF
        CALL CASRTS(COFX,ICOLX,NCX,IR,AJ,JCN,JDROW,IWS(IIWSOM))
      END DO

C             Wall and Stagnation stream surface conditions.

C     Right faces.
      DO K=2,NK-1
        DO J=1,NJ-1
          IR = IR + 1
          I = 1
          IF(K.LT.KLE .OR. K.GT.KTE)THEN
C           Stagnation stream surface.  Psi1 = 0.
C           Psi1 = 0 at right faces.
            I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
            CALL LBPSI1(I1E,0.)
          ELSE
C           Blade.  Delta theta = 0.
            CALL LDTHO((K-1)*(NJ-1) + J)
          END IF
          CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
        END DO
      END DO

C     Left faces.
      DO K=2,NK-1
        DO J=1,NJ-1
          IR = IR + 1
          I = NI
          IF(THINLE .AND. K.EQ.KSBLE)THEN
C           No leading edge.  Set dsblei=0.
            NA = 1
            COFA(1) = 1.
            ICOLA(1) = ISBLEO + J
          ELSE
            I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
            IF(IDWNBC.EQ.2)THEN
C             Set (delta psi1)_xi3 = 0.
              CALL LBDP1(I1E,I1E+NI*(NJ-1))
            ELSE
C             Set psi1=psi1_max.
              CALL LBPSI1(I1E,PSI1MX)
            END IF
          END IF
          CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
        END DO
      END DO

C     Bottom faces.
      DO K=2,NK-1
        DO I=1,NI-1
          IR = IR + 1
          J = 1
C         Psi2 = 0 at bottom faces.
          I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
          CALL LBPSI2(I2E,0.)
          CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
          END DO
        END DO

C       Top faces.
        DO K=2,NK-1
          DO I=1,NI-1
            IR = IR + 1
            J = NJ
            I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
            IF(IDWNBC.EQ.1 .OR. IDWNBC.LT.0)THEN
C             (delta psi2)_xi3 = 0.
              CALL LBDP2(I2E,I2E+NJ*(NI-1))
            ELSE
C             Psi2 = psi2_max at top faces.
              CALL LBPSI2(I2E,PSI2MX)
            END IF
            CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
          END DO
        END DO

C       Stagnation stream surface equations.
        I = 1
        DO K=1,NK
          DO J=1,NJ-1
            IR = IR + 1
            IF(K.EQ.1 .OR. (K.GE.KLE .AND. K.LE.KTE) .OR.
        1        (KTE.GT.NK .AND. K.EQ.NK))THEN
C             Psi1 = 0 at right faces.
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
              CALL LBPSI1(I1E,0.)
            ELSE IF(K.LT.KLE)THEN
C             Upstream stream surface.  Delta Pa = 0.
              IAF = (K-2)*NI*(NJ-1) + (J-1)*NI + I
              CALL LDPA(IAF,IAF+NI-1,0.)
            ELSE
C             Wake. Delta Pa for k-1 = 0.
              IAF = (K-1-2)*NI*(NJ-1) + (J-1)*NI + I
              CALL LDPA(IAF,IAF+NI-1,0.)
            END IF
            CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
          END DO
        END DO

C       Moving leading edge.  Delta Pa = 0.
C       If there is not a leading edge, it is set to 0 at the psi1 location
C       i=ni,k=ksble.  Therefore set this psi1 bc here.
        DO J=1,NSBLEI
          IR = IR + 1
          IF(THINLE)THEN
C           No leading edge.
            I1E = (KSBLE-1)*NI*(NJ-1) + (J-1)*NI + NI
            IF(IDWNBC.EQ.2)THEN
C             Set (delta psi1)_xi3 = 0.
              CALL LBDP1(I1E,I1E+NI*(NJ-1))
            ELSE
C             Set psi1=psi1_max.
              CALL LBPSI1(I1E,PSI1MX)
            END IF
          ELSE
            IAF = (KLE-2)*NI*(NJ-1) + (J-1)*NI + 1
            CALL LDPA(IAF,IAF+NI-1,0.)
          END IF
          CALL CASRTS(COFA,ICOLA,NA,IR,AJ,JCN,JDROW,IWS(IIWSOM))
        END DO

C       Print different rows of the matrix.
```

```
C       CALL PRTMAT(NSV,AJ,JCN,JDROW,FVEC)

        RETURN
        END

        SUBROUTINE PRTMAT(N,SP,ISP,NDROW,RHS)
C       This routine prints the columns and coeffs of the matrix for a given
C       row.
        INTEGER N,ISP(1),NDROW(1)
        REAL SP(1),RHS(1)

        INTEGER IR,IP,L,IFC

10      CONTINUE

        PRINT*,'ROW TO LOOK AT?  0 TO STOP.'
        READ*,IR

        IF(IR.EQ.0)RETURN

        IFC = 6

        WRITE(IFC,1000)IR,RHS(IR)
        IF(IR.EQ.1)THEN
          IP = 1
        ELSE
          IP = NDROW(IR-1) + 1
        END IF
        DO L=IP,NDROW(IR)
          WRITE(IFC,1010)ISP(L),SP(L)
        END DO

        IFC = 9

        WRITE(IFC,1000)IR,RHS(IR)
        IF(IR.EQ.1)THEN
          IP = 1
        ELSE
          IP = NDROW(IR-1) + 1
        END IF
        DO L=IP,NDROW(IR)
          WRITE(IFC,1010)ISP(L),SP(L)
        END DO

        GOTO 10

1000    FORMAT(1X,'Row ',I5,'.  Right hand side is ',E16.5)
1010    FORMAT(1X,I5,3X,E16.5)

        END

        SUBROUTINE EACHEQ
C       This routine sets up each equation in the COEF, ICOL array.
C       IR is set by the calling routine.

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'

        INTEGER IQNO,IQPTR

        IQNO = IEQNO(IR)
        IQPTR = IEQPTR(IR)

          GOTO (1,2,3,4,5,6),IQNO
          PRINT*,'Error in equation ordering. No equation for IR=',IR
          STOP
```

```
C          S-momentum equation on a volume.
1          CALL SMOM(IQPTR,WS(IAJ+1),WS(IJCN+1),WS(IJDROW+1),WS(IRHS+1))
           GOTO 999

C          A-momentum combined.
2          CALL AMOM(IQPTR,WS(IAJ+1),WS(IJCN+1),WS(IJDROW+1),WS(IRHS+1))
           GOTO 999

C          B-momentum combined.
3          CALL BMOM(IQPTR,WS(IAJ+1),WS(IJCN+1),WS(IJDROW+1),WS(IRHS+1))
           GOTO 999

C          Boundary condition.
4          CALL BCRL(IQPTR,WS(IAJ+1),WS(IJCN+1),WS(IJDROW+1),WS(IRHS+1))
           GOTO 999

C          Combined S-momentum equation.
5          CALL SMOMC(IQPTR,WS(IAJ+1),WS(IJCN+1),WS(IJDROW+1),WS(IRHS+1))
           GOTO 999

C          A-momentum combined for periodic boundary.
6          CALL AMOMPB(IQPTR,WS(IAJ+1),WS(IJCN+1),WS(IJDROW+1),WS(IRHS+1))
           GOTO 999

999        CONTINUE

         RETURN
         END


         SUBROUTINE BCRL(IROW,AJ,JCN,JDROW,RHS)
C        This routine takes the bc in the original matrix with row IROW and puts
C        it in the COFA array so it will be added to the reduced matrix.
         INTEGER IROW
         REAL AJ(1)
         INTEGER JCN(1),JDROW(1)
         REAL RHS(1)

         INCLUDE '3DCOM.FOR'
         INCLUDE 'MATCOM.FOR'

         INTEGER JS,I

         RHS(IR) = FVEC(IROW)

         IF(IROW.EQ.1)THEN
           JS = 0
           NA = JDROW(IROW)
         ELSE
           JS = JDROW(IROW-1)
           NA = JDROW(IROW) - JDROW(IROW-1)
         END IF

         DO I=1,NA
           COFA(I) = AJ(JS+I)
           ICOLA(I) = JCN(JS+I)
         END DO

         RETURN
         END


         SUBROUTINE CALGRD(AJ,JCN,JDROW)
C        This routine calculates the gadient which is J transpose F where
C        F is -FVEC.  The Result is put into GC.  IWS(ICWSB+1) is used
```

```
C       to store the row number temporarily.
        REAL AJ(1)
        INTEGER JCN(1),JDROW(1)

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INTEGER IPP,I

C       To calculate the gradient, store the row number of the original
C       matrix.
        IPP = 1
        DO IR=1,NSV
          DO I=IPP,JDROW(IR)
            IWS(ICWSB+I) = IR
          END DO
          IPP = JDROW(IR) + 1
        END DO

C       Get the gradient which is J transpose F.  F is -FVEC
        DO IR=1,NSV
          GC(IR) = 0.
        END DO
        DO I=1,JDROW(NSV)
          GC(JCN(I)) = GC(JCN(I)) - AJ(I)*FVEC(IWS(ICWSB+I))
        END DO

        RETURN
        END

        SUBROUTINE PRSTAT
C       This routine prints the error statistics.
        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'MESSAG.FOR'

        REAL XESMMX,YESMMX,ZESMMX
        REAL XEAMMX,YEAMMX,ZEAMMX
        REAL XEBMMX,YEBMMX,ZEBMMX
        REAL XEPAMX,YEPAMX,ZEPAMX
        REAL XEPBMX,YEPBMX,ZEPBMX

        IF(MSGLVO.EQ.0)RETURN

C       Calculate the x and y values for the errors.
        CALL CELCNT(MSMMAX,XESMMX,YESMMX,ZESMMX)
        CALL CELCNT(MAMMAX,XEAMMX,YEAMMX,ZEAMMX)
        CALL CELCNT(MBMMAX,XEBMMX,YEBMMX,ZEBMMX)
        CALL CELCNT(MPAMAX,XEPAMX,YEPAMX,ZEPAMX)
        CALL CELCNT(MPBMAX,XEPBMX,YEPBMX,ZEPBMX)

        PRINT*,' '
        PRINT*,
        1'  TYPE         MAX                  AVG',
        1'              X            Y            Z'

        WRITE(*,1000)' S-mom ',ESMMAX,ESMAVG,XESMMX,YESMMX,ZESMMX
        WRITE(*,1000)' A-mom ',EAMMAX,EAMAVG,XEAMMX,YEAMMX,ZEAMMX
        WRITE(*,1000)' B-mom ',EBMMAX,EBMAVG,XEBMMX,YEBMMX,ZEBMMX
        WRITE(*,1000)' Aux-Pa',EPAMAX,EPAAVG,XEPAMX,YEPAMX,ZEPAMX
        WRITE(*,1000)' Aux-Pb',EPBMAX,EPBAVG,XEPBMX,YEPBMX,ZEPBMX
        WRITE(*,1010)' BC     ',EBCMAX,EBCAVG
        IF(IBCMAX.EQ.1)THEN
          PRINT*,'Max BC error for upstream Pt.'
        ELSE IF(IBCMAX.EQ.2)THEN
          PRINT*,'Max BC error for upstream flow angle.'
        ELSE IF(IBCMAX.EQ.3)THEN
```

```
          PRINT*,'Max BC error for repeating or wall bc.'
        ELSE IF(IBCMAX.EQ.4)THEN
          PRINT*,'Max BC error for downstream bc.'
        END IF

1000    FORMAT(A7,2(1X,E16.7),3(2X,F10.5))
1010    FORMAT(A7,2(1X,E16.7))

        RETURN
        END

        SUBROUTINE CELCNT(M,XAVG,YAVG,ZAVG)
C       This routine finds the cell center XAVG, YAVG, ZAVG for the cell M.
C       This is not the exact center, only the average of the k face.
        INTEGER M
        REAL XAVG,YAVG,ZAVG

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'

        INTEGER I,J,K,INA(4),L

        IF(M.EQ.0)THEN
          XAVG = 0.
          YAVG = 0.
          ZAVG = 0.
        END IF

C       Get the indices for the volume.
        K = (M-1)/((NI-1)*(NJ-1)) + 2
        J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
        I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

        INA(1) = (K-1)*NI*NJ + (J-1)*NI + I
        INA(2) = INA(1) + 1
        INA(3) = INA(1) + NI
        INA(4) = INA(3) + 1

        XAVG = 0.
        YAVG = 0.
        ZAVG = 0.

        DO L=1,4
          XAVG = XAVG + 0.25*XN(INA(L))
          YAVG = YAVG + 0.25*YN(INA(L))
          ZAVG = ZAVG + 0.25*ZN(INA(L))
        END DO

        RETURN
        END

        SUBROUTINE UPPS(SDC,PMSP,IPMSP,NPDROW)
C       This routine updates Ps in the PNS vector from the solved matrix
C       and the entropy matrix.  dPs = PM (psi1,psi2) + SDC ent.
C       PNS(1:NEQ) has been set to RHS(1:NEQ)
        REAL SDC(1),PMSP(1)
        INTEGER IPMSP(1),NPDROW(1)

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER IPP,ISF,I

        IPP = 1
        DO ISF=1,NSF
```

```
         IR = IPSO + ISF
         PNS(IR) = SDC(ISF)*PNS(IR)
         DO I=IPP,NPDROW(ISF)
           PNS(IR) = PNS(IR) + PMSP(I)*PNS(IPMSP(I))
         END DO
         IPP = NPDROW(ISF) + 1
       END DO

       RETURN
       END

       SUBROUTINE PFRVSI(RWX,RWY,RWZ,S,RHO,OMEGRS, P)
C      This routine finds the Pressure P as a function of density RHO,
C      the entropy variable S = P + 0.5*rho u^2 - 0.5*rho (omega r)^2,
C      (omega r)^2, OMEGARS, and the three components of rho V,
C      RWX, RWY and RWZ for incompressible flow.
       REAL RWX,RWY,RWZ,S,RHO,OMEGRS,P

       P = S - 0.5*(RWX**2 + RWY**2 + RWZ**2)/RHO + 0.5*RHO*OMEGRS

       RETURN
       END

       SUBROUTINE PFRVSC(RWX,RWY,RWZ,S,I,OMEGRS,PREF,HREF,GAM,
       1        P,IOPT,IERR)
C      This routine finds the Pressure P as a function of rothalpy I,
C      the entropy S, (omega r)^2/2, OMEGRS, and the 3 components of rho V,
C      RWX, RWY and RWZ for compressible flow.  On input, P should be the
C      initial guess of the pressure.  IERR = 0 if converged. = 1 if not
C      or if there is no solution.  If close to convergence, the almost
C      converged solution will be returned.  Otherwise the sonic pressure
C      for the given rothalpy and entropy will be returned if on input IERR=0.
C      If on input IERR=1, then the pressure will be set to the initial P.
C      There are two solutions: a subsonic and supersonic.  If IOPT = 0, the
C      subsonic solution will be calculated.  If IOPT=1, the supersonic
C      solution will be calculated, and if IOPT= 2, then the branch will be
C      determined by the initial P and whether it's less than or greater than
C      the sonic pressure.
       REAL RWX,RWY,RWZ,S,I,OMEGRS,PREF,HREF,GAM,P
       INTEGER IOPT,IERR
C      The message common.
       INCLUDE 'MESSAG.FOR'

       REAL RHO,HSTAT,RWS,EPS,R1,R2,R3
       REAL A11,A13,A21,A22,A23,A32,A33,DET
       REAL DRHO,DP,DHSTAT,EMS,RDET
       REAL MACHSQ,RWSM1,PM1,HSM1,RHOM1,HSVO,PVO
       REAL PINIT,HSMS
       INTEGER ITER,ITMAX
       LOGICAL PSONIE
       DATA ITMAX/10/
       DATA EPS/1.E-5/

       IF(IERR.EQ.0)THEN
C        PSONIE is the logical is P SONic If an Error occured flag.
         PSONIE = .TRUE.
       ELSE
         PSONIE = .FALSE.
       END IF

       IERR = 0

       RWS = RWX**2 + RWY**2 + RWZ**2
       EMS = EXP(-S)

C      Get the pressure and RWS at Mach=1.
```

```
      MACHSQ = 1.
      HSM1 = (I + OMEGRS)/(1. + (GAM-1.)/2.*MACHSQ)
      PM1 = PREF*EMS*(HSM1/HREF)**(GAM/(GAM-1.))
      RHOM1 = GAM/(GAM-1.)*PM1/HSM1
      RWSM1 = GAM*RHOM1*PM1*MACHSQ

C     If RWS > RWSM1, then there is no solution.  Return with the
C     sonic pressure if (PSONIE).
      IF(RWS.GE.RWSM1)THEN
        IF(PSONIE)P = PM1
        IF(MSGLVP.EQ.1)THEN
          IF(PSONIE)THEN
            PRINT*,'(rho W)**2 is greater than sonic value. Setting',
     1        ' P to sonic value.'
          ELSE
            PRINT*,'(rho W)**2 is greater than sonic value. Leaving',
     1        ' P at initial value.'
          END IF
        END IF
        IERR = 1
        RETURN
      END IF

      PINIT = P

      IF(IOPT.EQ.0)THEN
C       Make sure pressure is on subsonic branch.
        IF(P.LE.PM1)THEN
C         Set the pressure to that for Mach=0.5
          MACHSQ = 0.5
          HSMS = (I + OMEGRS)/(1. + (GAM-1.)/2.*MACHSQ)
          P = PREF*EMS*(HSMS/HREF)**(GAM/(GAM-1.))
        END IF
      ELSE IF(IOPT.EQ.1)THEN
C       Make sure pressure is on supersonic branch.
        IF(P.GE.PM1)THEN
C         Set the pressure to that for Mach=1.5
          MACHSQ = 1.5
          HSMS = (I + OMEGRS)/(1. + (GAM-1.)/2.*MACHSQ)
          P = PREF*EMS*(HSMS/HREF)**(GAM/(GAM-1.))
        END IF
      END IF

C     If IOPT=2, the initial P decides the correct branch.


      RHO = (  GAM*P + SQRT( (GAM*P)**2 +
     1       2.*(GAM-1.)**2*(I + OMEGRS)*RWS )  )/
     1       ( 2.*(GAM-1.)*(I + OMEGRS) )

      HSTAT = GAM*P/((GAM-1)*RHO)

      ITER = 0

C     Start the iteration.
 10   CONTINUE

      ITER = ITER + 1

      R1 = -( HSTAT - I + RWS/(2.*RHO**2) - OMEGRS )/HREF

      R2 = -( RHO*HSTAT - GAM/(GAM-1)*P )/PREF

      R3 = -( P/PREF - EMS*(HSTAT/HREF)**(GAM/(GAM-1.)) )

C     Check convergence.
```

```
      IF(MAX(ABS(R1),ABS(R2),ABS(R3)).LE.EPS)RETURN

      A11 = -RWS/(HREF*RHO**3)
      A13 = 1./HREF
      A21 = HSTAT/PREF
      A22 = -GAM/(PREF*(GAM-1.))
      A23 = RHO/PREF
      A32 = 1./PREF
      A33 = -GAM/(GAM-1.)*EMS/HREF*(HSTAT/HREF)**(1./(GAM-1.))

      DET = A11*A22*A33 + A21*A32*A13 - A11*A23*A32

      IF(ITER.GT.ITMAX .OR. ABS(DET).LT. 0.01)THEN
        IF(MSGLVP.EQ.1)THEN
          PRINT*,'Pressure not converged in PFRVSC. EQ1,2,3 =',
     1        R1,R2,R3
          PRINT*,'(RHO W)^2,S,ROTH,OMEGRS,RHO,P,H',RWS,S,I,
     1        OMEGRS,RHO,P,HSTAT
          PRINT*,'DET = ',DET
        END IF
        IF(MAX(ABS(R1),ABS(R2),ABS(R3)).LE.EPS*100.)THEN
          IF(MSGLVP.EQ.1)THEN
            PRINT*,'Almost converged.  Using current pressure.'
          END IF
        ELSE
          IF(PSONIE)THEN
            P = PM1
          ELSE
            P = PINIT
          END IF
          IF(MSGLVP.EQ.1)THEN
            PRINT*,'Nonconvergence most likely caused',
     1        ' by ill-conditioned'
            IF(PSONIE)THEN
              PRINT*,'system near sonic.  Setting to sonic pressure.'
            ELSE
              PRINT*,'system near sonic.  Setting to initial pressure.'
            END IF
            PRINT*,'P = ',P
          END IF
        END IF
        IERR = 1
        RETURN
      END IF

      RDET = 1./DET

      DRHO = RDET*(R1*A22*A33 + R2*A32*A13 -
     1        A22*R3*A13 - R1*A23*A32)
      DP = RDET*(A11*R2*A33 + A21*R3*A13 - A11*A23*R3 - R1*A21*A33)
      DHSTAT = RDET*(A11*A22*R3 + A21*A32*R1 - A11*R2*A32)

      RHO = RHO + DRHO
      P = P + DP
      HSTAT = HSTAT + DHSTAT

      IF(RHO.LE.0.)THEN
C       Set to a small positive value.
        RHO = 0.001*PREF/HREF
      END IF

      IF(P.LE.0.)THEN
C       Set to a small positive value.
        P = 0.001*PREF
      END IF
      IF(HSTAT.LE.0.)THEN
C       Set to a small positive value.
        HSTAT = 0.001*HREF
      END IF

      GOTO 10

      END


      SUBROUTINE GRDDLE
C     This routine calculates the grid derivatives with respect to
C     the leading edge movement NX1G, NMPG, NX2G, and NX3G.

      INCLUDE '3DCOM.FOR'

      INTEGER I,J,K,N,KB,KBLD,NL,NU
      REAL SS,SP,DX1DS,DMPDS,XLV,YLV,ZLV,XUV,YUV,ZUV
      REAL DX2DMP,DX3DMP
      REAL XV,YV,ZV,ARCLEN,REX1,REX2
      REAL DEVAL

C     If there is no leading edge movement, then set its dependence to 0.
      IF(THINLE)THEN
        DO N=1,NN
          NX1G(N) = 0.
          NMPG(N) = 0.
          NX2G(N) = 0.
          NX3G(N) = 0.
        END DO
        RETURN
      END IF

C     Linearize the movement along the blade surface first.

      DO J=1,NJ
        DO K=KLE,KTE
          KB = K - KLE + 1

C         Arc length on lower surface of passage.
          SS = SBLE(J)*(1. - SGL(KB,J))
          I = 1
          N = (K-1)*NI*NJ + (J-1)*NI + I
          DX1DS = DEVAL(SS,X1BLD(1,J),DX1DSB(1,J),SBLD(1,J),NPBLD(J))
          DMPDS = DEVAL(SS,MPBLD(1,J),DMPDSB(1,J),SBLD(1,J),NPBLD(J))

          NX1G(N) = (1. - SGL(KB,J))*DX1DS
          NMPG(N) = (1. - SGL(KB,J))*DMPDS


C         Use the chain rule to get dx2/dsble = dm'/dsble  dx2/dm'.
          CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1        EMPRM(N),DX2DMP,1,1)
          NX2G(N) = NMPG(N)*DX2DMP
C         Use the chain rule to get dx3/dsble = dm'/dsble  dx3/dm'.
          CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1        EMPRM(N),DX3DMP,1,1)
          NX3G(N) = NMPG(N)*DX3DMP

C         Arc length on upper surface of passage.
          SP = SBLE(J) + (SBLD(NPBLD(J),J) - SBLE(J))*SGU(KB,J)
          I = NI
          N = (K-1)*NI*NJ + (J-1)*NI + I
          DX1DS = DEVAL(SP,X1BLD(1,J),DX1DSB(1,J),SBLD(1,J),NPBLD(J))
          DMPDS = DEVAL(SP,MPBLD(1,J),DMPDSB(1,J),SBLD(1,J),NPBLD(J))
```

```
            NX1G(N) = (1. - SGU(KB,J))*DX1DS
            NMPG(N) = (1. - SGU(KB,J))*DMPDS

C           Use the chain rule to get dx2/dsble = dm'/dsble  dx2/dm'.
            CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1          EMPRM(N),DX2DMP,1,1)
            NX2G(N) = NMPG(N)*DX2DMP
C           Use the chain rule to get dx3/dsble = dm'/dsble  dx3/dm'.
            CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1          EMPRM(N),DX3DMP,1,1)
            NX3G(N) = NMPG(N)*DX3DMP

          END DO
        END DO

C       Follow the blade movement in the rest of the grid upstream of trailing
C       edge.
        DO K=1,KTE
          IF(K.LE.KLE)THEN
            KBLD = KLE
          ELSE
            KBLD = K
          END IF
          DO J=1,NJ
            NL = (KBLD-1)*NI*NJ + (J-1)*NI + 1
            NU = (KBLD-1)*NI*NJ + (J-1)*NI + NI
            IF(CYLIND)THEN
              XLV = X2N(NL)*SIN(X1N(NL))
              YLV = X2N(NL)*COS(X1N(NL))
              XUV = X2N(NU)*SIN(X1N(NU))
              YUV = X2N(NU)*COS(X1N(NU))
            ELSE
              XLV = X1N(NL)
              YLV = X2N(NL)
              XUV = X1N(NU)
              YUV = X2N(NU)
            END IF
            ZLV = X3N(NL)
            ZUV = X3N(NU)

            DO I=1,NI
              N = (K-1)*NI*NJ + (J-1)*NI + I
              IF(CYLIND)THEN
                XV = X2N(N)*SIN(X1N(N))
                YV = X2N(N)*COS(X1N(N))
              ELSE
                XV = X1N(N)
                YV = X2N(N)
              END IF
              ZV = X3N(N)

C             Normalized arc length and exponential term to lower boundary.
              ARCLEN = 2.*SQRT( (XV-XLV)**2 + (YV-YLV)**2 + (ZV-ZLV)**2 )/
     1                 SBLD(NPBLD(J),J)
              REX1 = EXP(-8.*ARCLEN)

C             Normalized arc length and exponential term to upper boundary.
              ARCLEN = 2.*SQRT( (XV-XUV)**2 + (YV-YUV)**2 + (ZV-ZUV)**2 )/
     1                 SBLD(NPBLD(J),J)
              REX2 = EXP(-8.*ARCLEN)

              NX1G(N) = NX1G(NL)*REX1*(1.-REX2) + NX1G(NU)*REX2*(1.-REX1)
              NMPG(N) = NMPG(NL)*REX1*(1.-REX2) + NMPG(NU)*REX2*(1.-REX1)

C             Use the chain rule to get dx2/dsble = dm'/dsble  dx2/dm'.
              CALL LSPFIT(EMPRMT(1,J),X2MT(1,J),NMPRMT(J),
     1        EMPRM(N),DX2DMP,1,1)
            NX2G(N) = NMPG(N)*DX2DMP
C           Use the chain rule to get dx3/dsble = dm'/dsble  dx3/dm'.
            CALL LSPFIT(EMPRMT(1,J),X3MT(1,J),NMPRMT(J),
     1        EMPRM(N),DX3DMP,1,1)
            NX3G(N) = NMPG(N)*DX3DMP

          END DO
        END DO
      END DO

      DO K=KTE+1,NK
        DO J=1,NJ
          DO I=1,NI
            N = (K-1)*NI*NJ + (NJ-1)*NI + I
            NX1G(N) = 0.
            NX2G(N) = 0.
            NX3G(N) = 0.
            NMPG(N) = 0.
          END DO
        END DO
      END DO

C     Set upstream values to be the same so angle boundary condition applies.
      DO K=1,2
        DO J=1,NJ
          NU = NI*NJ + (J-1)*NI + 1
          DO I=1,NI
            N = (K-1)*NI*NJ + (J-1)*NI + I
            NX1G(N) = NX1G(NU)
            NX2G(N) = NX2G(NU)
            NX3G(N) = NX3G(NU)
            NMPG(N) = NMPG(NU)
          END DO
        END DO
      END DO

C     For the rest of the grid upstream of the leading edge, make
C     sure the periodic boundary has the same result.
      DO K=3,KLE-1
        DO J=1,NJ
          I = 1
          N = (K-1)*NI*NJ + (J-1)*NI + I
          NX1G(N) = (NX1G(N) + NX1G(N + NI-1))/2.
          NX2G(N) = (NX2G(N) + NX2G(N + NI-1))/2.
          NX3G(N) = (NX3G(N) + NX3G(N + NI-1))/2.
          NMPG(N) = (NMPG(N) + NMPG(N + NI-1))/2.

          NX1G(N + NI-1) = NX1G(N)
          NX2G(N + NI-1) = NX2G(N)
          NX3G(N + NI-1) = NX3G(N)
          NMPG(N + NI-1) = NMPG(N)
        END DO
      END DO

      RETURN
      END
```

## File ORDEQ.FOR

```
      SUBROUTINE ORDEQ
C     This routine orders the equations by
```

```
C       using MC21A to permute the row ordering so there
C       are no non-zero diagonals.  The result is put into the arrays
C       IEQNO and IEQPTR in common.
C       It orders the reduced equations.

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'

        INTEGER NUMNZ,I

C       Set up an arbitrary original equation ordering.  Store the
C       dominant non-zero locations in the ICN array.
        CALL OEQORD

C       Permute the equation order to ensure a dominant non-zero is on the
C       diagonal.
        CALL MC21A(NEQ,ICN,NCOL,IP21,LENR,IPERM,NUMNZ,IWSOEQ)

        IF(NUMNZ.LT.NEQ)THEN
C         Error ordering equations.
            PRINT*,'Error in ordering equations. Matrix must be singular.'
            STOP
        END IF

C       Store permuted equations in IEQNO and IEQPTR.
        DO I=1,NEQ
          IEQNO(I) = IOQNO( IPERM(I) )
          IEQPTR(I) = IOQPTR( IPERM(I) )
        END DO

        RETURN
        END

        SUBROUTINE OEQORD
C       This routine sets up an arbitrary original equation order
C       and the dominant non-zeros in each for the reduced system.

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'

        INTEGER IROW,I,J,K,IEQ,M,IAF,IBF,IJ,I1E,I2E

C       Set the arbitrary row counter IR to 0 and the column index ICOL.
C       Both are incremented in SETOEQ.
        IR = 0
        NCOL = 0

C       The reduced system now has the values of delta theta calculated
C       in place of psi1 for i=1, j=1,nj-1 and k=1,nk.  Also the leading
C       edge arc distance is in placew of psi1 at i=ni, j=1,nj-1 and k=kte.
C       The original equation order should reflect this.

C       Original equation order of the reduced system is:
C               Either
C                 (nm-nb) combined s-momentum eqs.
C                 (nb) volume s-momentum eqs.
C               Or
C                 (nm) volume s-momentum eqs.
C
C               (nk-2)(ni-2)(nj-1) reduced a-momentum eqs.
C               (nk-2)(ni-1)(nj-2) reduced b-momentum eqs.
C               1 psi2=0 equation for psi2 value at n2e - (ni-1)nj + 1.
C               ni(nj-1) downstream psi1's.  For i=1 and nk>kte,
C                       this should be the reduced a-momentum eq solving
C                       for delta theta.
C                       On right wall (i=1) and nk>=kte,
c                    psi1 = 0
c                 On left wall (i=ni)
c                    if(idwnbc=2)
c                         if(j=nj-1)
c                            ps = pdwn
c                         else
c                            (delta psi1)_xi2 = 0
c                    else
c                         psi1 = psi1_max
c                 For i=2,ni-1
c                    if(idwnbc=0)
c                         psi1 is specified
c                    else if(idwnbc=1)
c                         if(j=nj-1)
c                            ps = pdwn for face at i-1,j
c                         else
c                            (delta psi1)_xi2 = 0
c                    else if(idwnbc=2)
c                         ps = pdwn for face at i-1,j
c                    else if(idwnbc=3,5)
c                         if(j=nj-1)
c                            (dp)_xi1 = 0
c                         else
c                            (delta psi1)_xi2 = 0
c                    else if(idwnbc=4,6)
c                         (dp)_xi1 = 0
c                    else if(idwnbc<0)
c                         if(j=japplied)
c                            ps = pdwn
c                         else
c                            (delta psi1)_xi2 = 0
c             (ni-1)nj - 1 downstream psi2's.
c                 On lower wall (j=1)
c                    psi2 = 0
c                 On uper wall (j=nj)
c                    if(idwnbc=0,2,3,4,5,6)
c                         psi2 = psi2_max
c                    else if(idwnbc = 1 or -(nj-1))
c                         if(i=ni-1)
c                            Ps = pdwn
c                         else
c                            (delta psi2)_xi1 = 0
c                    else if( 0 < idwnbc < -(nj-1) )
c                         if(i=ni-1)
c                            dp/dx2 specified
c                         else
c                            (delta psi2)_xi3 = 0
c                 For j=2,nj-1
c                    if(idwnbc=0)
c                         psi2 is specified
c                    else if(idwnbc=1)
c                         Ps = pdwn for face at i,j-1
c                    else if(idwnbc=2)
c                         if(i=ni-1)
c                            ps = pdwn for face at i,j-1
c                         else
c                            (delta psi2)_xi1 = 0
c                    else if(idwnbc=4,6)
c                         if(i=ni-1)
c                            dp/dx2 specified
c                         else
c                            (delta psi2)_xi1 = 0
c                    else if(idwnbc=3,5, <0)
c                         dp/dx2 specified
c             ni(nj-1) upstream delta psi1 = 0.  For i=1 and kle>1,
c                     this is the angle bc applied to delta theta.
```

```
C             (ni-1)nj upstream delta psi2 = 0.
C             (ni-1)(nj-1) Pt bc's.
C             (nk-2)(nj-1) right faces
C                     if(k<kle)
C                             Periodic reduced a-momentum eqs.
C                     else if(kle<=k<=kte)
C                             delta theta = 0.
C                     else if(k>kle)
C                             Periodic reduced a-momentum eqs for
C                             immediate upstream station.
C             (nk-2)(nj-1) left faces
C                     if(k=ksble)
C                         if(kle>1 .and. (.not. thinle))
C                             Periodic reduced a-momentum eq for leading
C                             edge.
C                         else
C                             dSBLE = 0.
C                     else
C                         if (idwnbc=2)
C                             (delta psi1)_xi3 = 0
C                         else
C                             psi1 = psi1_max at left faces.
C             (nk-2)(ni-1) psi2 = 0 at bottom faces.
C             (nk-2)(ni-1) psi2  at top faces.
C                     if(idwnbc = 1 or <0)
C                             (delta psi2)_xi3 = 0
C                     else
C                             psi2 = psi2_max at top faces.

      IEQ = 1
      DO M=1,NM-NB
        CALL SETOEQ(IEQ,M)
        CALL POSPS(M + (NI-1)*(NJ-1))
      END DO

C     Allow downstream psi1 and psi2 values for last volume if not USEENT.
      DO M=NM-NB+1,NM
        CALL SETOEQ(IEQ,M)
        CALL POSPS(M + (NI-1)*(NJ-1))
        IF(.NOT.USEENT)THEN
          K = (M-1)/((NI-1)*(NJ-1)) + 2
          J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
          I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)
          IJ = N1E - (NJ-1)*NI + (J-1)*NI + I
C         CALL POSSI1(IJ)
          CALL POSSI1(IJ+1)
          IJ = N2E - (NI-1)*NJ + (J-1)*(NI-1) + I
C         CALL POSSI2(IJ)
          CALL POSSI2(IJ + (NI-1))
        END IF
      END DO

C     Reduced a-momentum eqs.
      IEQ = 2
      DO K=2,NK-1
        DO J=1,NJ-1
          DO I=2,NI-1
            IAF = (K-2)*NI*(NJ-1) + (J-1)*NI + I
            CALL SETOEQ(IEQ,IAF)
            CALL POSSI1(IAF + NI*(NJ-1))
          END DO
        END DO
      END DO

C     Reduced b-momentum eqs.
      IEQ = 3
```

```
          DO K=2,NK-1
            DO J=2,NJ-1
              DO I=1,NI-1
                IBF = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
                CALL SETOEQ(IEQ,IBF)
                CALL POSSI2(IBF + (NI-1)*NJ)
              END DO
            END DO
          END DO

C         Rest are bc's, except for the periodic reduced a-momentum eq.
C         IROW points to the row in the original large equation system.
          IEQ = 4

C         Psi2=0 equation for psi2 value at n2e - (ni-1)*nj + 1.
          IROW = 1
          CALL SETOEQ(IEQ,IROW)
          CALL POSSI2(N2E - (NI-1)*NJ + 1)

          IROW = 5*NM + 1

C         Downstream Psi1's.
          K = NK
          DO J=1,NJ-1
            IF(NK.GT.KTE)THEN
C             Apply periodic reduced a-momentum eq for i=1.
              IEQ = 6
              I = 1
              IROW = IROW + 1
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
C             Apply at a face for k-1.
              IAF = (K-1-2)*NI*(NJ-1) + (J-1)*NI + I
              CALL SETOEQ(IEQ,IAF)
              CALL POSSI1(I1E)
              IEQ = 4
            ELSE
C             Psi1 = 0.
              I = 1
              IROW = IROW + 1
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
              CALL SETOEQ(IEQ,IROW)
              CALL POSSI1(I1E)
            END IF

C           Rest of downstream psi1 boundary.
            DO I=2,NI-1
              IROW = IROW + 1
              CALL SETOEQ(IEQ,IROW)
              IJ = (J-1)*NI + I

              IF(IDWNBC.EQ.0)THEN
C               Psi1 is specified.
                CALL POSSI1(N1E - NB1 + IJ)
              ELSE IF( (IDWNBC.EQ.1 .AND. J.EQ.NJ-1) .OR.
     1                 IDWNBC.EQ.2 )THEN
C               This is the pressure equation at i-1,j.
                IF(USEENT)THEN
                  CALL POSSI1(N1E - NB1 + IJ)
                ELSE
                  CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-1)*(NI-1) + I-1)
                END IF
              ELSE IF( ( (IDWNBC.EQ.3 .OR. IDWNBC.EQ.5) .AND.
     1                   J.EQ.NJ-1 ) .OR.
     1                 ( IDWNBC.EQ.4 .OR. IDWNBC.EQ.6) )THEN
C               This is dp = 0 in xi1.
                IF(USEENT)THEN
              CALL POSSI1(N1E - NB1 + IJ)
            ELSE
              CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-1)*(NI-1) + I-1)
              CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-1)*(NI-1) + I)
            END IF
          ELSE IF(IDWNBC.LT.0 .AND. J.EQ.-IDWNBC)THEN
C           This is the pressure equation at i-1,j.
            IF(USEENT)THEN
              CALL POSSI1(N1E - NB1 + IJ)
            ELSE
              CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-1)*(NI-1) + I-1)
            END IF
          ELSE
C           This is a (delta psi1)_xi2 equation.
            IJ = (J-1)*NI + I
            CALL POSSI1(N1E - NB1 + IJ)
            CALL POSSI1(N1E - NB1 + IJ + NI)
          END IF
        END DO

        I = NI
        IROW = IROW + 1
        CALL SETOEQ(IEQ,IROW)
        IJ = (J-1)*NI + I
        IF(IDWNBC.EQ.2)THEN
          IF(J.EQ.NJ-1)THEN
C           This is the pressure equation at i-1,j.
            IF(USEENT)THEN
              CALL POSSI1(N1E - NB1 + IJ)
            ELSE
              CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-1)*(NI-1) + I-1)
            END IF
          ELSE
C           This is a (delta psi1)_xi2 equation.
            IJ = (J-1)*NI + I
            CALL POSSI1(N1E - NB1 + IJ)
            CALL POSSI1(N1E - NB1 + IJ + NI)
          END IF
        ELSE
C         This is psi1 = psi1_max eq.
          CALL POSSI1(N1E - NB1 + IJ)
        END IF
      END DO

C     Rest of Downstream Psi2's.
      DO J=1,NJ
        DO I=1,NI-1
          IJ = (J-1)*(NI-1) + I
C         IJ = 1 has already been accounted for.
          IF(IJ.EQ.1)GOTO 10

          IROW = IROW + 1
          CALL SETOEQ(IEQ,IROW)

          IF(J.EQ.1)THEN
C           Lower boundary psi2=0.
            CALL POSSI2(N2E - NB2 + IJ)
          ELSE IF(J.EQ.NJ)THEN
C           Upper boundary.
            IF(IDWNBC.EQ.1 .OR. IDWNBC.EQ.-(NJ-1))THEN
              IF(I.EQ.NI-1)THEN
C               Pressure applied at ni-1, nj-1.
                IF(USEENT)THEN
                  CALL POSSI2(N2E - NB2 + IJ)
                ELSE
                  CALL POSPS(NSF)
```

```
                 END IF
               ELSE
C                 (Delta psi2)_xi1 is applied.
                  CALL POSSI2(N2E - NB2 + IJ)
                  CALL POSSI2(N2E - NB2 + IJ + 1)
               END IF
             ELSE IF(IDWNBC.LT.0 .AND. IDWNBC.GT.-(NJ-1))THEN
               IF(I.EQ.NI-1)THEN
C                dp/dx2 applied at ni-1, nj-1.
                 IF(USEENT)THEN
                   CALL POSSI2(N2E - NB2 + IJ)
                 ELSE
                   CALL POSPS(NSF)
                   CALL POSPS(NSF - (ni-1))
                 END IF
               ELSE
C                 (Delta psi2)_xi1 is applied.
                  CALL POSSI2(N2E - NB2 + IJ)
                  CALL POSSI2(N2E - NB2 + IJ + 1)
               END IF
             ELSE
C              Apply psi2=psi2_max.
               CALL POSSI2(N2E - NB2 + IJ)
             END IF
           ELSE
C            J=2 through NJ-1
             IF(IDWNBC.EQ.0)THEN
C              Psi2 is specified.
               CALL POSSI2(N2E - NB2 + IJ)
             ELSE IF(IDWNBC.EQ.1)THEN
C              This is the pressure equation at i,j-1.
               IF(USEENT)THEN
                 CALL POSSI2(N2E - NB2 + IJ)
               ELSE
                 CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-2)*(NI-1) + I)
               END IF
             ELSE IF(IDWNBC.EQ.2)THEN
               IF(I.EQ.NI-1)THEN
C                This is the pressure equation at i,j-1.
                 IF(USEENT)THEN
                   CALL POSSI2(N2E - NB2 + IJ)
                 ELSE
                   CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-2)*(NI-1) + I)
                 END IF
               ELSE
C                This is (delta psi2)_xi1 = 0.
                 CALL POSSI2(N2E - NB2 + IJ)
                 CALL POSSI2(N2E - NB2 + IJ + 1)
               END IF
             ELSE IF(IDWNBC.EQ.4 .OR. IDWNBC.EQ.6)THEN
               IF(I.EQ.NI-1)THEN
C                dp/dx2 eq.
                 IF(USEENT)THEN
                   CALL POSSI2(N2E - NB2 + IJ)
                 ELSE
                   CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-1)*(NI-1) + I)
                   CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-2)*(NI-1) + I)
                 END IF
               ELSE
C                This is (delta psi2)_xi1 = 0.
                 CALL POSSI2(N2E - NB2 + IJ)
                 CALL POSSI2(N2E - NB2 + IJ + 1)
               END IF
             ELSE
C              dp/dx2 eq.
               IF(USEENT)THEN
```

```
                CALL POSSI2(N2E - NB2 + IJ)
              ELSE
                CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-1)*(NI-1) + I)
                CALL POSPS(NSF - (NJ-1)*(NI-1) + (J-2)*(NI-1) + I)
              END IF
            END IF
          END IF

10        CONTINUE
        END DO
      END DO

C     Upstream Delta Psi1 = 0.
C     For i=1, and kle>1, this is the angle bc applied to the delta theta.
C     The pointers work the same as the delta psi1 equation.
      DO IJ=1,NB1
        IROW = IROW + 1
        CALL SETOEQ(IEQ,IROW)
        CALL POSSI1(IJ)
C       CALL POSSI1(IJ+NI*(NJ-1))
      END DO

C     Upstream Delta Psi2 = 0.
      DO IJ=1,NB2
        IROW = IROW + 1
        CALL SETOEQ(IEQ,IROW)
        CALL POSSI2(IJ)
C       CALL POSSI2(IJ+(NI-1)*NJ)
      END DO

C     Entrpy BC upstream.
      DO IJ=1,NB
        IROW = IROW + 1
        CALL SETOEQ(IEQ,IROW)
        CALL POSPS(IJ)
      END DO

C             Wall and periodic boundary conditions.

      DO K=2,NK-1
        IF(K.LT.KLE)THEN
C         Periodic reduced a-momentum eq if on periodic boundary.
          IEQ = 6
          DO J=1,NJ-1
            IROW = IROW + 1
            I = 1
            I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
            IAF = (K-2)*NI*(NJ-1) + (J-1)*NI + I
            CALL SETOEQ(IEQ,IAF)
            CALL POSSI1(I1E)
          END DO
          IEQ = 4
        ELSE IF(K.GE.KLE .AND. K.LE.KTE)THEN
C         Delta theta = 0 at right faces on wall.
          DO J=1,NJ-1
            IROW = IROW + 1
            I = 1
            I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
            CALL SETOEQ(IEQ,IROW)
            CALL POSSI1(I1E)
          END DO
        ELSE
C         Periodic reduced a-momentum eq on wake so k>kte.  Apply to
C         immediate upstream station.
          IEQ = 6
          DO J=1,NJ-1
```

```
              IROW = IROW + 1
              I = 1
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
C             Apply at a face for k-1.
              IAF = (K-1-2)*NI*(NJ-1) + (J-1)*NI + I
              CALL SETOEQ(IEQ,IAF)
              CALL POSSI1(I1E)
            END DO
            IEQ = 4
          END IF
        END DO

C       Psi1 = psi1_max at left faces if a wall.  Or (delta psi1)_xi3 = 0
        DO K=2,NK-1
C         If there is no leading edge, then dsblei=0 is just a boundary
C         condition.
          IF(K.EQ.KSBLE .AND. KLE.GT.1 .AND. (.NOT.THINLE))THEN
C           Periodic reduced a-momentum eq at leading edge.
            IEQ = 6
            DO J=1,NJ-1
              IROW = IROW + 1
              I = NI
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
C             Apply at a face for k=KLE, I=1.
              IAF = (KLE-2)*NI*(NJ-1) + (J-1)*NI + 1
              CALL SETOEQ(IEQ,IAF)
              CALL POSSI1(I1E)
            END DO
            IEQ = 4
          ELSE
            DO J=1,NJ-1
              IROW = IROW + 1
              I = NI
              I1E = (K-1)*NI*(NJ-1) + (J-1)*NI + I
              CALL SETOEQ(IEQ,IROW)
              CALL POSSI1(I1E)
              IF(IDWNBC.EQ.2)CALL POSSI1(I1E+NI*(NJ-1))
            END DO
          END IF
        END DO

C       Psi2 = 0 at bottom faces.
        DO K=2,NK-1
          DO I=1,NI-1
            IROW = IROW + 1
            J = 1
            I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
            CALL SETOEQ(IEQ,IROW)
            CALL POSSI2(I2E)
          END DO
        END DO

C       Psi2 = psi2_max at top faces.
        DO K=2,NK-1
          DO I=1,NI-1
            IROW = IROW + 1
            J = NJ
            I2E = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
            CALL SETOEQ(IEQ,IROW)
            CALL POSSI2(I2E)
            IF(IDWNBC.EQ.1 .OR.IDWNBC.LT.0)THEN
C             Apply the (delta psi2)_xi3=0 eq.
              CALL POSSI2(I2E + (NI-1)*NJ)
            END IF
          END DO
        END DO
```

```
      RETURN
      END

      SUBROUTINE POSSI1(I1E)
C     This routine adds PSI1 at edge I1E as
C     a column in the matrix of dominant non-zeros.

      INTEGER I1E

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      LENR(IR) = LENR(IR) + 1
      NCOL = NCOL + 1
      ICN(NCOL) = IPSI1O + I1E

      RETURN
      END


      SUBROUTINE POSSI2(I2E)
C     This routine adds PSI2 at edge I2E as
C     a column in the matrix of dominant non-zeros.

      INTEGER I2E

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      LENR(IR) = LENR(IR) + 1
      NCOL = NCOL + 1
      ICN(NCOL) = IPSI2O + I2E

      RETURN
      END

      SUBROUTINE POSPS(ISF)
C     This routine adds PS for s face ISF as
C     a column in the matrix of dominant non-zeros.

      INTEGER ISF

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      LENR(IR) = LENR(IR) + 1
      NCOL = NCOL + 1
      ICN(NCOL) = IPSO + ISF

      RETURN
      END

      SUBROUTINE SETOEQ(NO,IPTR)
C     This routine initializes the arrays for the arbitrary original equation
C     order.

      INTEGER NO,IPTR

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'

C     Next Row.  Set number and pointer.
```

```
      IR = IR + 1
      IOQNO(IR) = NO
      IOQPTR(IR) = IPTR

C     IP21 points to the begining of the row.  LENR is the number in the
C     row.
      IP21(IR) = NCOL + 1
      LENR(IR) = 0

      RETURN
      END
```

## File REDUCE.FOR

```
      SUBROUTINE SMOM(M,AJ,JCN,JDROW,RHS)
C     This routine takes the fundamental equations, s-mom, a-mom, b-mom and
C     the auxilliary pressure eqs stored in the AJ, JCN, and JDROW arrays,
C     and manipulates them to create the s-momentum equation with no
C     dependence on PA or PB.  The result is put in the COFA, ICOLA, RHS
C     arrays.  This routine uses the first 5 locations in the COFV,
C     ICOLV, NCV, and RHSV arrays.
      INTEGER M
      REAL AJ(1)
      INTEGER JCN(1),JDROW(1)
      REAL RHS(1)

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER I,J,K,IA1F,IA2F,IB3F,IB4F
C     IQ1 is the pointer for the PA auxilliary pressure equation.
C     IQ2 is the pointer for the PB auxilliary pressure equation.
C     The rest are self explanatory.
      INTEGER IQS,IQA,IQB,IQ1,IQ2
      INTEGER IQSS,IQAS,IQBS,IQ1S,IQ2S
      INTEGER IQSN,IQAN,IQBN,IQ1N,IQ2N
      INTEGER IERR,NEV1,NEV2


C     Get the indices for the volume.
      K = (M-1)/((NI-1)*(NJ-1)) + 2
      J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
      I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C     Get the face indices.
C       a1 : i = I, j = J, k = K
        IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C       a2 : a1 + 1,  i = I + 1, j = J, k = K
        IA2F = IA1F + 1
C       b3 : i = I, j = J, k = K
        IB3F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C       b4 : b3 + (ni-1),  i = I, j = J + 1, k = K
        IB4F = IB3F + (NI-1)

C     Get the equation numbers.
      IQS = 5*M - 3
      IQA = 5*M - 2
      IQB = 5*M - 1
      IQ1 = 5*M
      IQ2 = 5*M + 1

C     Get the subscript at the beginning of the equations.
      IQSS = JDROW(IQS-1) + 1
      IQAS = JDROW(IQA-1) + 1
      IQBS = JDROW(IQB-1) + 1
      IQ1S = JDROW(IQ1-1) + 1
      IQ2S = JDROW(IQ2-1) + 1

C     Get the number of coefficients in each equation.
      IQSN = JDROW(IQS) - IQSS + 1
      IQAN = JDROW(IQA) - IQAS + 1
      IQBN = JDROW(IQB) - IQBS + 1
      IQ1N = JDROW(IQ1) - IQ1S + 1
      IQ2N = JDROW(IQ2) - IQ2S + 1

C             Eq (f).
C     Eliminate Pa1 from the s-momentum eq, by using the pa-eq.
C     Put in temp eq NEV=1.
      NEV = 1
      CALL ELIM(AJ(IQSS),JCN(IQSS),IQSN,FVEC(IQS),
      1       AJ(IQ1S),JCN(IQ1S),IQ1N,FVEC(IQ1),  IPAO+IA1F  ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (g).
C     Eliminate Pb3 from the equation above in NEV=1 and the Pb-eq.
C     Put in temp eq NEV=2.
      NEV = 2
      NEV2 = 1
      CALL ELIM(AJ(IQ2S),JCN(IQ2S),IQ2N,FVEC(IQ2),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1         IPBO+IB3F  ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (h).
C     Eliminate Pa1 from the a-momentum eq, by using the pa-eq.
C     Put in temp eq NEV=1 which is no longer used.
      NEV = 1
      CALL ELIM(AJ(IQAS),JCN(IQAS),IQAN,FVEC(IQA),
      1       AJ(IQ1S),JCN(IQ1S),IQ1N,FVEC(IQ1),  IPAO+IA1F  ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (i).
C     Eliminate Pb3 from the equation above in NEV=1 and the Pb-eq.
C     Put in temp eq NEV=3.
      NEV = 3
      NEV2 = 1
      CALL ELIM(AJ(IQ2S),JCN(IQ2S),IQ2N,FVEC(IQ2),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1         IPBO+IB3F  ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (j).
C     Eliminate Pa1 from the b-momentum eq, by using the pa-eq.
C     Put in temp eq NEV=1 which is no longer used.
      NEV = 1
      CALL ELIM(AJ(IQBS),JCN(IQBS),IQBN,FVEC(IQB),
      1       AJ(IQ1S),JCN(IQ1S),IQ1N,FVEC(IQ1),  IPAO+IA1F  ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (k).
C       Eliminate Pb3 from the equation above in NEV=1 and the Pb-eq.
C       Put in temp eq NEV=4.
      NEV = 4
      NEV2 = 1
      CALL ELIM(AJ(IQ2S),JCN(IQ2S),IQ2N,FVEC(IQ2),
     1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1         IPBO+IB3F  ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (l).
C       Eliminate Pa2 from the a-momentum and s-momentum eqs.
C       These are in NEV1=2 and NEV2=3 (g) and (i).
C       Put in temp eq NEV=1 which is no longer used.
      NEV = 1
      NEV1 = 2
      NEV2 = 3
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
     1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1        IPAO+IA2F ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (m).
C       Eliminate Pa2 from the a-momentum and b-momentum eqs.
C       These are in NEV1=3 and NEV2=4 (i) and (k).
C       Put in temp eq NEV=5.
      NEV = 5
      NEV1 = 3
      NEV2 = 4
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
     1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1        IPAO+IA2F ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in SMOM'
      END IF

C             Eq (n).
C       Eliminate Pb4 from the b-momentum and s-momentum eqs.
C       These are in NEV1=1 and NEV2=5 (l) and (m).
C       Put in cofa which will then be added to A by the calling routine.
      NEV1 = 1
      NEV2 = 5
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
     1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1        IPBO+IB4F ,
     1       COFA,ICOLA,NA,RHS(IR),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in SMOM'
      END IF

C       Set NEV to 5 as the max number of vector locations used.
      NEV = 5

      RETURN
      END


      SUBROUTINE AMOM(IAF,AJ,JCN,JDROW,RHS)
C     This routine takes the fundamental equations, a-mom, b-mom and
C     the auxilliary pressure eqs stored in the JA, JCN, and JDROW arrays,
C     and manipulates them to create the a-momentum equation with no
C     dependence on PA or PB.  The result is put in the COFA, ICOLA, RHS
C     arrays.  This routine uses the first 5 locations in the COFV,
C     ICOLV, NCV, and RHSV arrays.
      INTEGER IAF
      REAL AJ(1)
      INTEGER JCN(1),JDROW(1)
      REAL RHS(1)

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER I,J,K,MI,MII,IA1F,IA2F,IA3F,IB1F,IB2F,IB3F,IB4F
C     IQ1 is the pointer for the PA auxilliary pressure equation.
C     IQ2 is the pointer for the PB auxilliary pressure equation.
C     The rest are self explanatory.
      INTEGER IQAI,IQBI,IQ1I,IQ2I
      INTEGER IQASI,IQBSI,IQ1SI,IQ2SI
      INTEGER IQANI,IQBNI,IQ1NI,IQ2NI
      INTEGER IQAII,IQBII,IQ1II,IQ2II
      INTEGER IQASII,IQBSII,IQ1SII,IQ2SII
      INTEGER IQANII,IQBNII,IQ1NII,IQ2NII
      INTEGER IERR,NEV1,NEV2

C     Get the volume numbers.
      K = (IAF-1)/(NI*(NJ-1)) + 2
      J = (IAF - (K-2)*NI*(NJ-1) - 1)/NI + 1
      I = IAF - (K-2)*NI*(NJ-1) - (J-1)*NI

      MII = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
      MI = MII - 1

C     Get the face indices.
C       Set I to the value for a1.
        I = I - 1

C       a1 : i = I, j = J, k = K
        IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C       a2 : a1 + 1,  i = I + 1, j = J, k = K
        IA2F = IA1F + 1
C       a3 : a2 + 1,  i = I + 2, j = J, k = K
        IA3F = IA2F + 1
C       b1 : i = I, j = J, k = K
        IB1F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C       b2 : b1 + 1, i = I + 1, j = J, k = K
        IB2F = IB1F + 1
C       b3 : b1 + (ni-1), i = I, j = J + 1, k = K
        IB3F = IB1F + (NI-1)
C       b4 : b3 + 1,  i = I + 1, j = J + 1, k = K
        IB4F = IB3F + 1

C      Get the equation numbers.
       IQAI = 5*MI - 2
       IQBI = 5*MI - 1
       IQ1I = 5*MI
       IQ2I = 5*MI + 1
       IQAII = 5*MII - 2
       IQBII = 5*MII - 1
       IQ1II = 5*MII
      IQ2II = 5*MII + 1

C     Get the subscript at the beginning of the equations.
      IQASI = JDROW(IQAI-1) + 1
      IQBSI = JDROW(IQBI-1) + 1
      IQ1SI = JDROW(IQ1I-1) + 1
      IQ2SI = JDROW(IQ2I-1) + 1
      IQASII = JDROW(IQAII-1) + 1
      IQBSII = JDROW(IQBII-1) + 1
      IQ1SII = JDROW(IQ1II-1) + 1
      IQ2SII = JDROW(IQ2II-1) + 1

C     Get the number of coefficients in each equation.
      IQANI = JDROW(IQAI) - IQASI + 1
      IQBNI = JDROW(IQBI) - IQBSI + 1
      IQ1NI = JDROW(IQ1I) - IQ1SI + 1
      IQ2NI = JDROW(IQ2I) - IQ2SI + 1
      IQANII = JDROW(IQAII) - IQASII + 1
      IQBNII = JDROW(IQBII) - IQBSII + 1
      IQ1NII = JDROW(IQ1II) - IQ1SII + 1
      IQ2NII = JDROW(IQ2II) - IQ2SII + 1

C             Eq (i).
C     Eliminate Pa1 from the a-momentum eq at I, by using the pa-eq at I.
C     Put in temp eq NEV=1.
      NEV = 1
      CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
     1        AJ(IQASI),JCN(IQASI),IQANI,FVEC(IQAI),  IPAO+IA1F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in AMOM'
      END IF

C             Eq (j).
C     Eliminate Pa3 from the a-momentum eq at II, by using the pa-eq at II.
C     Put in temp eq NEV=2.
      NEV = 2
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1        AJ(IQASII),JCN(IQASII),IQANII,FVEC(IQAII),  IPAO+IA3F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in AMOM'
      END IF

C             Eq (k).
C     Eliminate Pa1 from the b-momentum eq at I, by using the pa-eq at I.
C     Put in temp eq NEV=3.
      NEV = 3
      CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
     1        AJ(IQBSI),JCN(IQBSI),IQBNI,FVEC(IQBI),  IPAO+IA1F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in AMOM'
      END IF

C             Eq (l).
C     Eliminate Pa3 from the b-momentum eq at II, by using the pa-eq at II.
C     Put in temp eq NEV=4.
      NEV = 4
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1        AJ(IQBSII),JCN(IQBSII),IQBNII,FVEC(IQBII),  IPAO+IA3F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in AMOM'
      END IF
```

```
C           Eq (m).
C       Eliminate Pb1 from equation (i) in NEV=1 and the Pb-eq for I.
C       Put in temp eq NEV=5.
        NEV = 5
        NEV2 = 1
        CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPB0+IB1F  ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOM'
        END IF

C           Eq (n).
C       Eliminate Pb2 from equation (j) in NEV=2 and the Pb-eq for II.
C       Put in temp eq NEV=1.
        NEV = 1
        NEV2 = 2
        CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPB0+IB2F  ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOM'
        END IF

C           Eq (o).
C       Eliminate Pb1 from equation (k) in NEV=3 and the Pb-eq for I.
C       Put in temp eq NEV=2.
        NEV = 2
        NEV2 = 3
        CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPB0+IB1F  ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOM'
        END IF

C           Eq (p).
C       Eliminate Pb2 from equation (l) in NEV=4 and the Pb-eq for II.
C       Put in temp eq NEV=3.
        NEV = 3
        NEV2 = 4
        CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPB0+IB2F  ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOM'
        END IF

C           Eq (q).
C       Eliminate Pb3 from the a-momentum and b-momentum eqs at I.
C       These are in NEV1=5 and NEV2=2 (m) and (o).
C       Put in temp eq NEV=4.
        NEV = 4
        NEV1 = 5
        NEV2 = 2
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1        IPB0+IB3F ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOM'
        END IF
```

```
C             Eq (r).
C       Eliminate Pb4 from the a-momentum and b-momentum eqs at II.
C       These are in NEV1=1 and NEV2=3 (n) and (p).
C       Put in temp eq NEV=5.
        NEV = 5
        NEV1 = 1
        NEV2 = 3
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1        IPBO+IB4F ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOM'
        END IF

C             Eq (s).
C       Eliminate Pa2 from the two a-momentum eqs.
C       These are in NEV1=4 and NEV2=5 (q) and (r).
C       Put in cofa which will then be added to A by the calling routine.
        NEV1 = 4
        NEV2 = 5
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1        IPAO+IA2F ,
        1       COFA,ICOLA,NA,RHS(IR),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOM'
        END IF

C       Add stiffener.
        CALL ADSTIF(ASTIF,COFA,ICOLA,NA,IR,1,NI*(NJ-1))

        RETURN
        END


        SUBROUTINE AMOMPB(IAF,AJ,JCN,JDROW,RHS)
C       This routine is similar to AMOM except it is for a periodic
C       boundary.  IAF should correspond to i=1.
C       This routine takes the fundamental equations, a-mom, b-mom and
C       the auxilliary pressure eqs stored in the JA, JCN, and JDROW arrays,
C       and manipulates them to create the a-momentum equation with no
C       dependence on PA or PB.  The result is put in the COFA, ICOLA, RHS
C       arrays.  This routine uses the first 5 locations in the COFV,
C       ICOLV, NCV, and RHSV arrays.
        INTEGER IAF
        REAL AJ(1)
        INTEGER JCN(1),JDROW(1)
        REAL RHS(1)

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER I,J,K,MI,MII,IA1F,IA2F,IA3F,IA4F,IB1F,IB2F,IB3F,IB4F
C       IQ1 is the pointer for the PA auxilliary pressure equation.
C       IQ2 is the pointer for the PB auxilliary pressure equation.
C       The rest are self explanatory.
        INTEGER IQAI,IQBI,IQ1I,IQ2I
        INTEGER IQASI,IQBSI,IQ1SI,IQ2SI
        INTEGER IQANI,IQBNI,IQ1NI,IQ2NI
        INTEGER IQAII,IQBII,IQ1II,IQ2II
        INTEGER IQASII,IQBSII,IQ1SII,IQ2SII
        INTEGER IQANII,IQBNII,IQ1NII,IQ2NII
C       IQDPA is the pointer for the delta Pa equation across the periodic
```

```
C       boundary.
        INTEGER IQDPA,IQDPAS,IQDPAN
        INTEGER IERR,NEV1,NEV2

C       Get the volume numbers.
        K = (IAF-1)/(NI*(NJ-1)) + 2
        J = (IAF - (K-2)*NI*(NJ-1) - 1)/NI + 1
        I = IAF - (K-2)*NI*(NJ-1) - (J-1)*NI

        IF(I.NE.1)PRINT*,'Error in AMOMPB.  I.NE.1'

        MII = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
        MI = MII + NI - 2

C       Get the face indices.

C         a1 : i = NI-1, j = J, k = K
          I = NI - 1
          IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         a2 : i = 1, j = J, k = K
          I = 1
          IA2F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         a3 : i = 2, j = J, k = K
          I = 2
          IA3F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         a4 : i = NI, j = J, k = K
          I = NI
          IA4F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         b1 : i = NI-1, j = J, k = K
          I = NI - 1
          IB1F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C         b2 : i = 1, j = J, k = K
          I = 1
          IB2F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C         b3 : b1 + (ni-1) ; i = NI-1, j = J + 1, k = K
          IB3F = IB1F + (NI-1)
C         b4 : b2 + (ni-1) ; i = 1, j = J + 1, k = K
          IB4F = IB2F + (NI-1)

C       Get the equation numbers.
        IQAI = 5*MI - 2
        IQBI = 5*MI - 1
        IQ1I = 5*MI
        IQ2I = 5*MI + 1
        IQAII = 5*MII - 2
        IQBII = 5*MII - 1
        IQ1II = 5*MII
        IQ2II = 5*MII + 1

        IF(K.EQ.KLE .AND. (.NOT.THINLE))THEN
C         Movable leading edge.
          IQDPA = ISBLEO + J
        ELSE IF(K.LT.KLE)THEN
C         Upstream stagnation stream surface.
          IQDPA = IDTHO + (K-1)*(NJ-1) + J
        ELSE
C         Wake.  Equation applied to downstream point.
          IQDPA = IDTHO + K*(NJ-1) + J
        END IF

C       Get the subscript at the beginning of the equations.
        IQASI = JDROW(IQAI-1) + 1
        IQBSI = JDROW(IQBI-1) + 1
        IQ1SI = JDROW(IQ1I-1) + 1
        IQ2SI = JDROW(IQ2I-1) + 1
        IQASII = JDROW(IQAII-1) + 1
```

```
      IQBSII = JDROW(IQBII-1) + 1
      IQ1SII = JDROW(IQ1II-1) + 1
      IQ2SII = JDROW(IQ2II-1) + 1

      IQDPAS = JDROW(IQDPA-1) + 1

C     Get the number of coefficients in each equation.
      IQANI = JDROW(IQAI) - IQASI + 1
      IQBNI = JDROW(IQBI) - IQBSI + 1
      IQ1NI = JDROW(IQ1I) - IQ1SI + 1
      IQ2NI = JDROW(IQ2I) - IQ2SI + 1
      IQANII = JDROW(IQAII) - IQASII + 1
      IQBNII = JDROW(IQBII) - IQBSII + 1
      IQ1NII = JDROW(IQ1II) - IQ1SII + 1
      IQ2NII = JDROW(IQ2II) - IQ2SII + 1

      IQDPAN = JDROW(IQDPA) - IQDPAS + 1

C             Eq (i).
C     Eliminate Pa1 from the a-momentum eq at I, by using the pa-eq at I.
C     Put in temp eq NEV=1.
      NEV = 1
      CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
     1        AJ(IQASI),JCN(IQASI),IQANI,FVEC(IQAI),  IPAO+IA1F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (j).
C     Eliminate Pa3 from the a-momentum eq at II, by using the pa-eq at II.
C     Put in temp eq NEV=2.
      NEV = 2
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1        AJ(IQASII),JCN(IQASII),IQANII,FVEC(IQAII),  IPAO+IA3F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (k).
C     Eliminate Pa1 from the b-momentum eq at I, by using the pa-eq at I.
C     Put in temp eq NEV=3.
      NEV = 3
      CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
     1        AJ(IQBSI),JCN(IQBSI),IQBNI,FVEC(IQBI),  IPAO+IA1F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (l).
C     Eliminate Pa3 from the b-momentum eq at II, by using the pa-eq at II.
C     Put in temp eq NEV=4.
      NEV = 4
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1        AJ(IQBSII),JCN(IQBSII),IQBNII,FVEC(IQBII),  IPAO+IA3F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (m).
C     Eliminate Pb1 from equation (i) in NEV=1 and the Pb-eq for I.
C     Put in temp eq NEV=5.
      NEV = 5
      NEV2 = 1
      CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1         IPBO+IB1F   ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (n).
C     Eliminate Pb2 from equation (j) in NEV=2 and the Pb-eq for II.
C     Put in temp eq NEV=1.
      NEV = 1
      NEV2 = 2
      CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1         IPBO+IB2F   ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (o).
C     Eliminate Pb1 from equation (k) in NEV=3 and the Pb-eq for I.
C     Put in temp eq NEV=2.
      NEV = 2
      NEV2 = 3
      CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1         IPBO+IB1F   ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (p).
C     Eliminate Pb2 from equation (l) in NEV=4 and the Pb-eq for II.
C     Put in temp eq NEV=3.
      NEV = 3
      NEV2 = 4
      CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1         IPBO+IB2F   ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (q).
C     Eliminate Pb3 from the a-momentum and b-momentum eqs at I.
C     These are in NEV1=5 and NEV2=2 (m) and (o).
C     Put in temp eq NEV=4.
      NEV = 4
      NEV1 = 5
      NEV2 = 2
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1       IPBO+IB3F ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in AMOMPB'
      END IF

C             Eq (r).
C     Eliminate Pb4 from the a-momentum and b-momentum eqs at II.
C     These are in NEV1=1 and NEV2=3 (n) and (p).
```

```
C       Put in temp eq NEV=5.
        NEV = 5
        NEV1 = 1
        NEV2 = 3
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1        IPBO+IB4F ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOMPB'
        END IF

C               Eq (t).
C       Eliminate Pa4 from equation (q) in NEV=4 and the Delta Pa-eq.
C       Put in temp eq NEV=1.
        NEV = 1
        NEV2 = 4
        CALL ELIM(AJ(IQDPAS),JCN(IQDPAS),IQDPAN,FVEC(IQDPA),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPAO+IA4F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOMPB'
        END IF

C               Eq (u).
C       Eliminate Pa2 from the two a-momentum eqs.
C       These are in NEV1=1 and NEV2=5 (t) and (r).
C       Put in cofa which will then be added to A by the calling routine.
        NEV1 = 1
        NEV2 = 5
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1        IPAO+IA2F ,
        1       COFA,ICOLA,NA,RHS(IR),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in AMOMPB'
        END IF

        RETURN
        END

        SUBROUTINE CALDPA(IAF,DPA,AJ,JCN,JDROW)
C       This routine takes the fundamental equations, a-mom, b-mom and
C       the auxilliary pressure eqs stored in the AJ, JCN, and JDROW arrays,
C       and manipulates them to update the a-face pressure for face IAF.  The
C       changes to the other values are in the PNS array. The resulting
C       change is DPA.
        INTEGER IAF
        REAL DPA
        REAL AJ(1)
        INTEGER JCN(1),JDROW(1)

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER I,J,K,MI,MII,IA1F,IA2F,IA3F,IB1F,IB2F,IB3F,IB4F
C       IQ1 is the pointer for the PA auxilliary pressure equation.
C       IQ2 is the pointer for the PB auxilliary pressure equation.
C       The rest are self explanatory.
        INTEGER IQAI,IQBI,IQ1I,IQ2I
        INTEGER IQASI,IQBSI,IQ1SI,IQ2SI
        INTEGER IQANI,IQBNI,IQ1NI,IQ2NI
        INTEGER IQAII,IQBII,IQ1II,IQ2II
        INTEGER IQASII,IQBSII,IQ1SII,IQ2SII
```

```
      INTEGER IQANII,IQBNII,IQ1NII,IQ2NII
      INTEGER IERR,NEV1,NEV2

      INTEGER IPAC,ICAC
      REAL ACAC

C     Get the face indices.
      K = (IAF-1)/(NI*(NJ-1)) + 2
      J = (IAF - (K-2)*NI*(NJ-1) - 1)/NI + 1
      I = IAF - (K-2)*NI*(NJ-1) - (J-1)*NI


      IF(I.EQ.1)THEN
C       Use eq (r) which is a-mom for volume II.  Pa3, pb2 and pb4
C       must be eliminated using volume II equations and (j), (l),
C       (n), (p), and (r).

C       Get the face indices.
C       I is set to the value for a2.

        MII = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I

C       a2 : i = I, j = J, k = K
        IA2F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C       a3 : a2 + 1,  i = I + 2, j = J, k = K
        IA3F = IA2F + 1
C       b2 : i = I, j = J, k = K
        IB2F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C       b4 : b2 + (ni-1), i = I, j = J + 1, k = K
        IB4F = IB2F + (NI-1)

C       Get the equation numbers.
        IQAII = 5*MII - 2
        IQBII = 5*MII - 1
        IQ1II = 5*MII
        IQ2II = 5*MII + 1

C       Get the subscript at the beginning of the equations.
        IQASII = JDROW(IQAII-1) + 1
        IQBSII = JDROW(IQBII-1) + 1
        IQ1SII = JDROW(IQ1II-1) + 1
        IQ2SII = JDROW(IQ2II-1) + 1

C       Get the number of coefficients in each equation.
        IQANII = JDROW(IQAII) - IQASII + 1
        IQBNII = JDROW(IQBII) - IQBSII + 1
        IQ1NII = JDROW(IQ1II) - IQ1SII + 1
        IQ2NII = JDROW(IQ2II) - IQ2SII + 1

C             Eq (j).
C     Eliminate Pa3 from the a-momentum eq at II, by using the pa-eq at II.
C     Put in temp eq NEV=2.
      NEV = 2
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1       AJ(IQASII),JCN(IQASII),IQANII,FVEC(IQAII),  IPAO+IA3F   ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in CALDPA'
      END IF

C             Eq (l).
C     Eliminate Pa3 from the b-momentum eq at II, by using the pa-eq at II.
C     Put in temp eq NEV=4.
      NEV = 4
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1       AJ(IQBSII),JCN(IQBSII),IQBNII,FVEC(IQBII),  IPAO+IA3F   ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in CALDPA'
      END IF

C            Eq (n).
C     Eliminate Pb2 from equation (j) in NEV=2 and the Pb-eq for II.
C     Put in temp eq NEV=1.
      NEV = 1
      NEV2 = 2
      CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
     1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1          IPBO+IB2F  ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in CALDPA'
      END IF

C            Eq (p).
C     Eliminate Pb2 from equation (l) in NEV=4 and the Pb-eq for II.
C     Put in temp eq NEV=3.
      NEV = 3
      NEV2 = 4
      CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
     1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1          IPBO+IB2F  ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in CALDPA'
      END IF

C            Eq (r).
C     Eliminate Pb4 from the a-momentum and b-momentum eqs at II.
C     These are in NEV1=1 and NEV2=3 (n) and (p).
C     Put in temp eq NEV=5.
      NEV = 5
      NEV1 = 1
      NEV2 = 3
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
     1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1         IPBO+IB4F ,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in CALDPA'
      END IF

C            else    else    else    else    else    else
      ELSE
C            else    else    else    else    else    else

C       Eliminate pa1, and solve for pa2 using eq (q).  Pa1, pb1 and pb3
C       must be eliminated using volume I equations and (i), (k),
C       (m), (o), and (q).

C       Get the face indices.
C       Set I to the value for a1.
        I = I - 1

        MI = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I

C       a1 : i = I, j = J, k = K
        IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C       a2 : a1 + 1,  i = I + 1, j = J, k = K
        IA2F = IA1F + 1
C       b1 : i = I, j = J, k = K
        IB1F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
```

```
C         b3 : b1 + (ni-1), i = I, j = J + 1, k = K
          IB3F = IB1F + (NI-1)

C         Get the equation numbers.
          IQAI = 5*MI - 2
          IQBI = 5*MI - 1
          IQ1I = 5*MI
          IQ2I = 5*MI + 1

C         Get the subscript at the beginning of the equations.
          IQASI = JDROW(IQAI-1) + 1
          IQBSI = JDROW(IQBI-1) + 1
          IQ1SI = JDROW(IQ1I-1) + 1
          IQ2SI = JDROW(IQ2I-1) + 1

C         Get the number of coefficients in each equation.
          IQANI = JDROW(IQAI) - IQASI + 1
          IQBNI = JDROW(IQBI) - IQBSI + 1
          IQ1NI = JDROW(IQ1I) - IQ1SI + 1
          IQ2NI = JDROW(IQ2I) - IQ2SI + 1

C               Eq (i).
C       Eliminate Pa1 from the a-momentum eq at I, by using the pa-eq at I.
C       Put in temp eq NEV=1.
        NEV = 1
        CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
        1       AJ(IQASI),JCN(IQASI),IQANI,FVEC(IQAI),  IPAO+IA1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPA'
        END IF

C               Eq (k).
C       Eliminate Pa1 from the b-momentum eq at I, by using the pa-eq at I.
C       Put in temp eq NEV=3.
        NEV = 3
        CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
        1       AJ(IQBSI),JCN(IQBSI),IQBNI,FVEC(IQBI),  IPAO+IA1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPA'
        END IF

C               Eq (m).
C       Eliminate Pb1 from equation (i) in NEV=1 and the Pb-eq for I.
C       Put in temp eq NEV=5.
        NEV = 5
        NEV2 = 1
        CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1          IPBO+IB1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPA'
        END IF

C               Eq (o).
C       Eliminate Pb1 from equation (k) in NEV=3 and the Pb-eq for I.
C       Put in temp eq NEV=2.
        NEV = 2
        NEV2 = 3
        CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1          IPBO+IB1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
```

```
          PRINT*,'Error in CALDPA'
        END IF

C              Eq (q).
C       Eliminate Pb3 from the a-momentum and b-momentum eqs at I.
C       These are in NEV1=5 and NEV2=2 (m) and (o).
C       Put in temp eq NEV=4.
        NEV = 4
        NEV1 = 5
        NEV2 = 2
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1        COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPBO+IB3F ,
        1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPA'
        END IF

C              endif   endif   endif   endif   endif
        END IF
C              endif   endif   endif   endif   endif

C       The column to solve for is in IPAC.
        IPAC = IPAO + IAF
        DO I=1,NCV(NEV)
          IF(ICOLV(I,NEV).EQ.IPAC)THEN
            ICAC = I
            ACAC = COFV(I,NEV)
            GOTO 10
          END IF
        END DO

        PRINT*,'Error in CALDPA'
        DPA = 0.
        RETURN

10      CONTINUE
        DPA = RHSV(NEV)

        DO I=1,ICAC-1
          IF(ICOLV(I,NEV).GT.IPAO .AND. ICOLV(I,NEV).LE.IDTHO)THEN
            PRINT*,'Error in CALDPA'
            RETURN
          END IF
          DPA = DPA - COFV(I,NEV)*PNS(ICOLV(I,NEV))
        END DO

        DO I=ICAC+1,NCV(NEV)
          IF(ICOLV(I,NEV).GT.IPAO .AND. ICOLV(I,NEV).LE.IDTHO)THEN
            PRINT*,'Error in CALDPA'
            RETURN
          END IF
          DPA = DPA - COFV(I,NEV)*PNS(ICOLV(I,NEV))
        END DO

        DPA = DPA/ACAC

        RETURN
        END

        SUBROUTINE BMOM(IBF,AJ,JCN,JDROW,RHS)
C       This routine takes the fundamental equations, a-mom, b-mom and
C       the auxilliary pressure eqs stored in the JA, JCN, and JDROW arrays,
C       and manipulates them to create the b-momentum equation with no
C       dependence on PA or PB.  The result is put in the COFA, ICOLA, RHS
C       arrays.  This routine uses the first 5 locations in the COFV,
C       ICOLV, NCV, and RHSV arrays.
        INTEGER IBF
        REAL AJ(1)
        INTEGER JCN(1),JDROW(1)
        REAL RHS(1)

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER I,J,K,MI,MII,IA1F,IA2F,IA3F,IA4F,IB1F,IB2F,IB3F
C       IQ1 is the pointer for the PA auxilliary pressure equation.
C       IQ2 is the pointer for the PB auxilliary pressure equation.
C       The rest are self explanatory.
        INTEGER IQAI,IQBI,IQ1I,IQ2I
        INTEGER IQASI,IQBSI,IQ1SI,IQ2SI
        INTEGER IQANI,IQBNI,IQ1NI,IQ2NI
        INTEGER IQAII,IQBII,IQ1II,IQ2II
        INTEGER IQASII,IQBSII,IQ1SII,IQ2SII
        INTEGER IQANII,IQBNII,IQ1NII,IQ2NII
        INTEGER IERR,NEV1,NEV2

C       Get the volume numbers.
        K = (IBF-1)/((NI-1)*NJ) + 2
        J = (IBF - (K-2)*(NI-1)*NJ - 1)/(NI-1) + 1
        I = IBF - (K-2)*(NI-1)*NJ - (J-1)*(NI-1)

        MII = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
        MI = MII - (NI-1)

C       Get the face indices.
C         Set J to the value for b1.
          J = J - 1

C         a1 : i = I, j = J, k = K
          IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         a2 : a1 + 1,  i = I + 1, j = J, k = K
          IA2F = IA1F + 1
C         a3 : a1 + ni,  i = I, j = J + 1, k = K
          IA3F = IA1F + NI
C         a4 : a3 + 1,  i = I + 1, j = J + 1, k = K
          IA4F = IA3F + 1
C         b1 : i = I, j = J, k = K
          IB1F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C         b2 : b1 + (ni-1), i = I, j = J + 1, k = K
          IB2F = IB1F + (NI-1)
C         b3 : b2 + (ni-1), i = I, j = J + 2, k = K
          IB3F = IB2F + (NI-1)

C       Get the equation numbers.
        IQAI = 5*MI - 2
        IQBI = 5*MI - 1
        IQ1I = 5*MI
        IQ2I = 5*MI + 1
        IQAII = 5*MII - 2
        IQBII = 5*MII - 1
        IQ1II = 5*MII
        IQ2II = 5*MII + 1

C       Get the subscript at the beginning of the equations.
        IQASI = JDROW(IQAI-1) + 1
        IQBSI = JDROW(IQBI-1) + 1
        IQ1SI = JDROW(IQ1I-1) + 1
        IQ2SI = JDROW(IQ2I-1) + 1
        IQASII = JDROW(IQAII-1) + 1
        IQBSII = JDROW(IQBII-1) + 1
```

```
      IQ1SII = JDROW(IQ1II-1) + 1
      IQ2SII = JDROW(IQ2II-1) + 1

C     Get the number of coefficients in each equation.
      IQANI = JDROW(IQAI) - IQASI + 1
      IQBNI = JDROW(IQBI) - IQBSI + 1
      IQ1NI = JDROW(IQ1I) - IQ1SI + 1
      IQ2NI = JDROW(IQ2I) - IQ2SI + 1
      IQANII = JDROW(IQAII) - IQASII + 1
      IQBNII = JDROW(IQBII) - IQBSII + 1
      IQ1NII = JDROW(IQ1II) - IQ1SII + 1
      IQ2NII = JDROW(IQ2II) - IQ2SII + 1

C             Eq (i).
C     Eliminate Pb1 from the b-momentum eq at I, by using the pb-eq at I.
C     Put in temp eq NEV=1.
      NEV = 1
      CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
     1        AJ(IQBSI),JCN(IQBSI),IQBNI,FVEC(IQBI),  IPBO+IB1F   ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (j).
C     Eliminate Pb3 from the b-momentum eq at II, by using the pb-eq at II.
C     Put in temp eq NEV=2.
      NEV = 2
      CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
     1        AJ(IQBSII),JCN(IQBSII),IQBNII,FVEC(IQBII),  IPBO+IB3F   ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (k).
C     Eliminate Pb1 from the a-momentum eq at I, by using the pb-eq at I.
C     Put in temp eq NEV=3.
      NEV = 3
      CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
     1        AJ(IQASI),JCN(IQASI),IQANI,FVEC(IQAI),  IPBO+IB1F   ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (l).
C     Eliminate Pb3 from the a-momentum eq at II, by using the pb-eq at II.
C     Put in temp eq NEV=4.
      NEV = 4
      CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
     1        AJ(IQASII),JCN(IQASII),IQANII,FVEC(IQAII),  IPBO+IB3F   ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.O)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (m).
C     Eliminate Pa1 from equation (i) in NEV=1 and the Pa-eq for I.
C     Put in temp eq NEV=5.
      NEV = 5
      NEV2 = 1
      CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
     1        COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1          IPAO+IA1F   ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
```

```
      IF(IERR.NE.0)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (n).
C     Eliminate Pa3 from equation (j) in NEV=2 and the Pa-eq for II.
C     Put in temp eq NEV=1.
      NEV = 1
      NEV2 = 2
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1        COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1          IPAO+IA3F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (o).
C     Eliminate Pa1 from equation (k) in NEV=3 and the Pa-eq for I.
C     Put in temp eq NEV=2.
      NEV = 2
      NEV2 = 3
      CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
     1        COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1          IPAO+IA1F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (p).
C     Eliminate Pa3 from equation (l) in NEV=4 and the Pa-eq for II.
C     Put in temp eq NEV=3.
      NEV = 3
      NEV2 = 4
      CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
     1        COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1          IPAO+IA3F  ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (q).
C     Eliminate Pa2 from the b-momentum and a-momentum eqs at I.
C     These are in NEV1=5 and NEV2=2 (m) and (o).
C     Put in temp eq NEV=4.
      NEV = 4
      NEV1 = 5
      NEV2 = 2
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
     1        COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
     1         IPAO+IA2F ,
     1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (r).
C     Eliminate Pa4 from the b-momentum and a-momentum eqs at II.
C     These are in NEV1=1 and NEV2=3 (n) and (p).
C     Put in temp eq NEV=5.
      NEV = 5
      NEV1 = 1
      NEV2 = 3
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1        IPAO+IA4F ,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in BMOM'
      END IF

C             Eq (s).
C     Eliminate Pb2 from the two b-momentum eqs.
C     These are in NEV1=4 and NEV2=5 (q) and (r).
C     Put in cofa which will then be added to A by the calling routine.
      NEV1 = 4
      NEV2 = 5
      CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
      1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
      1        IPBO+IB2F ,
      1       COFA,ICOLA,NA,RHS(IR),IERR)
      IF(IERR.NE.0)THEN
        PRINT*,'Error in BMOM'
      END IF

C     Add stiffener.
      CALL ADSTIF(BSTIF,COFA,ICOLA,NA,IR,NI-1,(NI-1)*NJ)

      RETURN
      END


      SUBROUTINE CALDPB(IBF,DPB,AJ,JCN,JDROW)
C     This routine takes the fundamental equations, a-mom, b-mom and
C     the auxilliary pressure eqs stored in the JA, JCN, and JDROW arrays,
C     and manipulates them to update the b-face pressure for face IBF.  The
C     changes to the other values are in the PNS array. The resulting
C     change is DPB.
      INTEGER IBF
      REAL DPB
      REAL AJ(1)
      INTEGER JCN(1),JDROW(1)

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER I,J,K,MI,MII,IA1F,IA2F,IA3F,IA4F,IB1F,IB2F,IB3F
C     IQ1 is the pointer for the PA auxilliary pressure equation.
C     IQ2 is the pointer for the PB auxilliary pressure equation.
C     The rest are self explanatory.
      INTEGER IQAI,IQBI,IQ1I,IQ2I
      INTEGER IQASI,IQBSI,IQ1SI,IQ2SI
      INTEGER IQANI,IQBNI,IQ1NI,IQ2NI
      INTEGER IQAII,IQBII,IQ1II,IQ2II
      INTEGER IQASII,IQBSII,IQ1SII,IQ2SII
      INTEGER IQANII,IQBNII,IQ1NII,IQ2NII
      INTEGER IERR,NEV1,NEV2

      INTEGER IPAC,ICAC
      REAL ACAC

C     Get the face indices.
      K = (IBF-1)/((NI-1)*NJ) + 2
      J = (IBF - (K-2)*(NI-1)*NJ - 1)/(NI-1) + 1
      I = IBF - (K-2)*(NI-1)*NJ - (J-1)*(NI-1)

      IF(J.EQ.1)THEN
C       Use eq (r) which is b-mom for volume II.  Pa3, pa4, and pb3
C       must be eliminated using volume II equations and (j), (l),
C         (n), (p), and (r).

C         Get the face indices.
C         J is set to the value for b2.

          MII = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I

C         a3 : i = I, j = J, k = K
          IA3F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         a4 : a3 + 1,  i = I + 1, j = J, k = K
          IA4F = IA3F + 1
C         b2 : i = I, j = J, k = K
          IB2F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C         b3 : b2 + (ni-1), i = I, j = J + 1, k = K
          IB3F = IB2F + (NI-1)

C         Get the equation numbers.
          IQAII = 5*MII - 2
          IQBII = 5*MII - 1
          IQ1II = 5*MII
          IQ2II = 5*MII + 1

C         Get the subscript at the beginning of the equations.
          IQASII = JDROW(IQAII-1) + 1
          IQBSII = JDROW(IQBII-1) + 1
          IQ1SII = JDROW(IQ1II-1) + 1
          IQ2SII = JDROW(IQ2II-1) + 1

C         Get the number of coefficients in each equation.
          IQANII = JDROW(IQAII) - IQASII + 1
          IQBNII = JDROW(IQBII) - IQBSII + 1
          IQ1NII = JDROW(IQ1II) - IQ1SII + 1
          IQ2NII = JDROW(IQ2II) - IQ2SII + 1

C               Eq (j).
C       Eliminate Pb3 from the b-momentum eq at II, by using the pb-eq at II.
C       Put in temp eq NEV=2.
        NEV = 2
        CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
        1       AJ(IQBSII),JCN(IQBSII),IQBNII,FVEC(IQBII),  IPBO+IB3F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (l).
C       Eliminate Pb3 from the a-momentum eq at II, by using the pb-eq at II.
C       Put in temp eq NEV=4.
        NEV = 4
        CALL ELIM(AJ(IQ2SII),JCN(IQ2SII),IQ2NII,FVEC(IQ2II),
        1       AJ(IQASII),JCN(IQASII),IQANII,FVEC(IQAII),  IPBO+IB3F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (n).
C       Eliminate Pa3 from equation (j) in NEV=2 and the Pa-eq for II.
C       Put in temp eq NEV=1.
        NEV = 1
        NEV2 = 2
        CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPAO+IA3F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
```

```
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (p).
C       Eliminate Pa3 from equation (1) in NEV=4 and the Pa-eq for II.
C       Put in temp eq NEV=3.
        NEV = 3
        NEV2 = 4
        CALL ELIM(AJ(IQ1SII),JCN(IQ1SII),IQ1NII,FVEC(IQ1II),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPAO+IA3F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (r).
C       Eliminate Pa4 from the b-momentum and a-momentum eqs at II.
C       These are in NEV1=1 and NEV2=3 (n) and (p).
C       Put in temp eq NEV=5.
        NEV = 5
        NEV1 = 1
        NEV2 = 3
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPAO+IA4F ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               else    else    else    else    else    else
        ELSE
C               else    else    else    else    else    else

C         Eliminate pb1, and solve for pb2 using eq (q).  Pa1, pa2 and pb1
C         must be eliminated using volume I equations and (i), (k),
C         (m), (o), and (q).

C         Get the face indices.
C         Set J to the value for b1.
          J = J - 1

          MI = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I

C         a1 : i = I, j = J, k = K
          IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         a2 : a1 + 1,  i = I + 1, j = J, k = K
          IA2F = IA1F + 1
C         b1 : i = I, j = J, k = K
          IB1F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C         b2 : b1 + (ni-1), i = I, j = J + 1, k = K
          IB2F = IB1F + (NI-1)

C         Get the equation numbers.
          IQAI = 5*MI - 2
          IQBI = 5*MI - 1
          IQ1I = 5*MI
          IQ2I = 5*MI + 1

C         Get the subscript at the beginning of the equations.
          IQASI = JDROW(IQAI-1) + 1
          IQBSI = JDROW(IQBI-1) + 1
          IQ1SI = JDROW(IQ1I-1) + 1
          IQ2SI = JDROW(IQ2I-1) + 1
```

```
C         Get the number of coefficients in each equation.
          IQANI = JDROW(IQAI) - IQASI + 1
          IQBNI = JDROW(IQBI) - IQBSI + 1
          IQ1NI = JDROW(IQ1I) - IQ1SI + 1
          IQ2NI = JDROW(IQ2I) - IQ2SI + 1

C               Eq (i).
C       Eliminate Pb1 from the b-momentum eq at I, by using the pb-eq at I.
C       Put in temp eq NEV=1.
        NEV = 1
        CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
        1       AJ(IQBSI),JCN(IQBSI),IQBNI,FVEC(IQBI),  IPBO+IB1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (k).
C       Eliminate Pb1 from the a-momentum eq at I, by using the pb-eq at I.
C       Put in temp eq NEV=3.
        NEV = 3
        CALL ELIM(AJ(IQ2SI),JCN(IQ2SI),IQ2NI,FVEC(IQ2I),
        1       AJ(IQASI),JCN(IQASI),IQANI,FVEC(IQAI),  IPBO+IB1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (m).
C       Eliminate Pa1 from equation (i) in NEV=1 and the Pa-eq for I.
C       Put in temp eq NEV=5.
        NEV = 5
        NEV2 = 1
        CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPAO+IA1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (o).
C       Eliminate Pa1 from equation (k) in NEV=3 and the Pa-eq for I.
C       Put in temp eq NEV=2.
        NEV = 2
        NEV2 = 3
        CALL ELIM(AJ(IQ1SI),JCN(IQ1SI),IQ1NI,FVEC(IQ1I),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1         IPAO+IA1F   ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
          PRINT*,'Error in CALDPB'
        END IF

C               Eq (q).
C       Eliminate Pa2 from the b-momentum and a-momentum eqs at I.
C       These are in NEV1=5 and NEV2=2 (m) and (o).
C       Put in temp eq NEV=4.
        NEV = 4
        NEV1 = 5
        NEV2 = 2
        CALL ELIM(COFV(1,NEV1),ICOLV(1,NEV1),NCV(NEV1),RHSV(NEV1),
        1       COFV(1,NEV2),ICOLV(1,NEV2),NCV(NEV2),RHSV(NEV2),
        1        IPAO+IA2F ,
        1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),RHSV(NEV),IERR)
        IF(IERR.NE.O)THEN
```

```
          PRINT*,'Error in CALDPB'
        END IF

C                endif   endif   endif   endif   endif
        END IF
C                endif   endif   endif   endif   endif

C       The column to solve for is in IPAC.
        IPAC = IPBO + IBF
        DO I=1,NCV(NEV)
          IF(ICOLV(I,NEV).EQ.IPAC)THEN
            ICAC = I
            ACAC = COFV(I,NEV)
            GOTO 10
          END IF
        END DO

        PRINT*,'Error in CALDPB'
        DPB = 0.
        RETURN

10      CONTINUE
        DPB = RHSV(NEV)

        DO I=1,ICAC-1
          IF(ICOLV(I,NEV).GT.IPAO .AND. ICOLV(I,NEV).LE.IDTHO)THEN
            PRINT*,'Error in CALDPB'
            RETURN
          END IF
          DPB = DPB - COFV(I,NEV)*PNS(ICOLV(I,NEV))
        END DO

        DO I=ICAC+1,NCV(NEV)
          IF(ICOLV(I,NEV).GT.IPAO .AND. ICOLV(I,NEV).LE.IDTHO)THEN
            PRINT*,'Error in CALDPB'
            RETURN
          END IF
          DPB = DPB - COFV(I,NEV)*PNS(ICOLV(I,NEV))
        END DO

        DPB = DPB/ACAC

        RETURN
        END

        SUBROUTINE SMOMC(ISF,AJ,JCN,JDROW,RHS)
C       This routine subtracts the s-momentum eqs for the volumes adjoining
C       the S-face ISF.  These eqs are obtained by calling SMOM for the
C       two volumes.  The result is put  in the COFA, ICOLA, RHS arrays.
        INTEGER ISF
        REAL AJ(1)
        INTEGER JCN(1),JDROW(1)
        REAL RHS(1)

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER I,MI,MII,NEVS

C       Get the two volumes.
        MII = ISF
        MI = MII - (NI-1)*(NJ-1)

        CALL SMOM(MI,AJ,JCN,JDROW,RHS)
```

```
C       Store the result in the NEV+1 V location, so will not be touched
C       by next call to SMOMV.
        NEV = NEV + 1
        NEVS = NEV
        DO I=1,NA
          COFV(I,NEV) = COFA(I)
          ICOLV(I,NEV) = ICOLA(I)
        END DO
        NCV(NEV) = NA
        RHSV(NEV) = RHS(IR)

        CALL SMOM(MII,AJ,JCN,JDROW,RHS)

C       Store the result in the first V location.
        NEV = 1
        DO I=1,NA
          COFV(I,NEV) = COFA(I)
          ICOLV(I,NEV) = ICOLA(I)
        END DO
        NCV(NEV) = NA
        RHSV(NEV) = RHS(IR)

C       Combine equations.
        CALL COM2EQ(COFV(1,NEVS),ICOLV(1,NEVS),NCV(NEVS),RHSV(NEVS),
        1       COFV(1,1),ICOLV(1,1),NCV(1),RHSV(1), -1.,1.,
        1       COFA,ICOLA,NA,RHS(IR))

        RETURN
        END

        SUBROUTINE ADSTIF(STIF,COFA,ICOLA,NA,IR,N1,N2)
C       This routine adds a 2D Laplacian stiffener to the reduced momentum
C       equation.  Basically DIAG_new = (1 + STIF) DIAG_old.
C       OFFDIAG_new = OFFDIAG_old - STIF DIAG_old/4.  For the A momentum
C       eq, the off diags are in the xi1 and xi3 direction.  For the B
C       momentum eq, the off diags are in the xi2 and xi3 direction.
C       COFA, ICOLA, and NA are the packed, sorted equation.  IR is the
C       diagonal.  N1 and N2 are the difference between off diagonal and
C       diagonal column number where N2>N1, so the order is known.
        REAL STIF,COFA(1)
        INTEGER ICOLA(1),NA,IR,N1,N2

        INTEGER IC(4),ICD,I
        REAL ADDIAG,ADOD
        LOGICAL ISRT

C       IC(1)-IC(4) and ICD are the index in the COFA arrays for the off diagonal
C       and diagonal.
        DO I=1,4
          IC(I) = 0
        END DO
        ICD = 0

        DO I=1,NA
          IF(ICOLA(I) .EQ. IR)THEN
C           Diagonal.
            ICD = I
          ELSE IF(ICOLA(I) .EQ. IR-N2)THEN
C           Off diagonal.
            IC(1) = I
          ELSE IF(ICOLA(I) .EQ. IR-N1)THEN
C           Off diagonal.
            IC(2) = I
          ELSE IF(ICOLA(I) .EQ. IR+N1)THEN
C           Off diagonal.
            IC(3) = I
          ELSE IF(ICOLA(I) .EQ. IR+N2)THEN
C           Off diagonal. IR+N2 should be last column.
            IC(4) = I
            GOTO 10
          END IF
        END DO

10      CONTINUE

        IF(ICD.EQ.0)THEN
          PRINT*,'Error adding stiffener.  IR = ',IR
        END IF

        DO I=1,4
          IF(IC(I).EQ.0)THEN
            PRINT*,'Error adding stiffener to off-diag.',
     1        '  IR,I = ',IR,I
            IC(I) = NA + 1
          END IF
        END DO

        ADDIAG = STIF*COFA(ICD)

        COFA(ICD) = COFA(ICD) + ADDIAG

        ADOD = -ADDIAG/4.

        DO I=1,4
          COFA( IC(I) ) = COFA( IC(I) ) + ADOD
        END DO

        RETURN
        END
```

## File SAB.FOR

```
        SUBROUTINE RSAB(ISF)
C       This routine calculates A0 and B0 in the FACE common for
C       the S face ISF.
        INTEGER ISF

        INCLUDE '3DCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER I,J,K

        K = (ISF-1)/((NI-1)*(NJ-1)) + 1
        J = (ISF - (K-1)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
        I = ISF - (K-1)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C       Psi1 indices.
        IA3 = (K-1)*NI*(NJ-1) + (J-1)*NI + I
        IA4 = IA3 + 1
        IA9 = IA3 + NI*(NJ-1)
        IA10 = IA9 + 1

C       These psi1's.
        A3 = PSI1(IA3)
        A4 = PSI1(IA4)
        A9 = PSI1(IA9)
        A10 = PSI1(IA10)

C       Psi2 indices.
        IB2 = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
        IB5 = IB2 + (NI-1)
```

```
      IB8 = IB2 + (NI-1)*NJ
      IB11 = IB8 + (NI-1)

C     These psi2's.
      B2 = PSI2(IB2)
      B5 = PSI2(IB5)
      B8 = PSI2(IB8)
      B11 = PSI2(IB11)

      A0 = (A3 + A4 + A9 + A10)/4.
      B0 = (B2 + B5 + B8 + B11)/4.

      RETURN
      END


      SUBROUTINE LSAB(ISF,DEQDA0,DEQDB0)
C     This routine adds the A0 coefficients*DEQDA0 and
C     the B0 coeffs*DEQDB0 to the COLX arrays.
      INTEGER ISF
      REAL DEQDA0,DEQDB0

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER I,J,K

      K = (ISF-1)/((NI-1)*(NJ-1)) + 1
      J = (ISF - (K-1)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
      I = ISF - (K-1)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C     Psi1 indices.
      IA3 = (K-1)*NI*(NJ-1) + (J-1)*NI + I
      IA4 = IA3 + 1
      IA9 = IA3 + NI*(NJ-1)
      IA10 = IA9 + 1

C     Psi2 indices.
      IB2 = (K-1)*(NI-1)*NJ + (J-1)*(NI-1) + I
      IB5 = IB2 + (NI-1)
      IB8 = IB2 + (NI-1)*NJ
      IB11 = IB8 + (NI-1)

C     Put Jacobians in correct location in matrix.
C     NCX must be set already.
C


C
C     Psi1 at 3
      CALL ADCOFS(0.25*DEQDA0,IA3+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 4
      CALL ADCOFS(0.25*DEQDA0,IA4+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 9
      CALL ADCOFS(0.25*DEQDA0,IA9+IPSI10,COFX,ICOLX,NCX)
C
C     Psi1 at 10
      CALL ADCOFS(0.25*DEQDA0,IA10+IPSI10,COFX,ICOLX,NCX)
C
C     Psi2 at 2
      CALL ADCOFS(0.25*DEQDB0,IB2+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 5
      CALL ADCOFS(0.25*DEQDB0,IB5+IPSI20,COFX,ICOLX,NCX)
C
```

```
C     Psi2 at 8
      CALL ADCOFS(0.25*DEQDBO,IB8+IPSI20,COFX,ICOLX,NCX)
C
C     Psi2 at 11
      CALL ADCOFS(0.25*DEQDBO,IB11+IPSI20,COFX,ICOLX,NCX)
C

        RETURN
        END


        SUBROUTINE IENTEQ(I,J,K)
C       This routine initializes the variables in the FACE common for
C       ENTEQ.
        INTEGER I,J,K

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

C       S face downstream.    i = I, j = J, k = K.
        ISFD = (K-1)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I

C       S face upstream.    i = I, j = J, k = K - 1.
        ISF1 = ISFD - (NI-1)*(NJ-1)

C       Other S faces upstream.
        ISF2 = ISF1 - 1
        ISF3 = ISF1 + 1
        ISF4 = ISF1 - (NI-1)
        ISF5 = ISF1 + (NI-1)

        SD = S(ISFD)
        S1 = S(ISF1)
        CALL RSAB(ISFD)
        ASD = AO
        BSD = BO
        CALL RSAB(ISF1)
        AS1 = AO
        BS1 = BO

        IF(I.NE.1)THEN
          S2 = S(ISF2)
          CALL RSAB(ISF2)
          AS2 = AO
          BS2 = BO
        END IF
        IF(I.NE.NI-1)THEN
          S3 = S(ISF3)
          CALL RSAB(ISF3)
          AS3 = AO
          BS3 = BO
        END IF
        IF(J.NE.1)THEN
          S4 = S(ISF4)
          CALL RSAB(ISF4)
          AS4 = AO
          BS4 = BO
        END IF
        IF(J.NE.NJ-1)THEN
          S5 = S(ISF5)
          CALL RSAB(ISF5)
          AS5 = AO
          BS5 = BO
        END IF
      RETURN
      END
```

## File SAVSUB.FOR

```
      SUBROUTINE SAVEA(IAF,RWX,RWY,RWZ,RHO)
C     This routine saves some calculated flow quantities in common
C     for the a face IAF.
      INTEGER IAF
      REAL RWX,RWY,RWZ,RHO

      INCLUDE '3DCOM.FOR'
      INCLUDE 'CLSCOM.FOR'

      IF(MNNS.EQ.1)THEN
        RWXA(1) = RWX
        RWYA(1) = RWY
        RWZA(1) = RWZ
        RHOA(1) = RHO
      ELSE
        RWXA(IAF) = RWX
        RWYA(IAF) = RWY
        RWZA(IAF) = RWZ
        RHOA(IAF) = RHO
      END IF

      RETURN
      END


      SUBROUTINE SAVEB(IBF,RWX,RWY,RWZ,RHO)
C     This routine saves some calculated flow quantities in common
C     for the B face IBF.
      INTEGER IBF
      REAL RWX,RWY,RWZ,RHO

      INCLUDE '3DCOM.FOR'
      INCLUDE 'CLSCOM.FOR'

      IF(MNNS.EQ.1)THEN
        RWXB(1) = RWX
        RWYB(1) = RWY
        RWZB(1) = RWZ
        RHOB(1) = RHO
      ELSE
        RWXB(IBF) = RWX
        RWYB(IBF) = RWY
        RWZB(IBF) = RWZ
        RHOB(IBF) = RHO
      END IF

      RETURN
      END


      SUBROUTINE SAVES(ISF,RWX,RWY,RWZ,RHO)
C     This routine saves some calculated flow quantities in common
C     for the S face ISF.
      INTEGER ISF
      REAL RWX,RWY,RWZ,RHO

      INCLUDE '3DCOM.FOR'
      INCLUDE 'CLSCOM.FOR'

      IF(MNNS.EQ.1)THEN
        RWXS(1) = RWX
        RWYS(1) = RWY
        RWZS(1) = RWZ
        RHOS(1) = RHO
      ELSE
        RWXS(ISF) = RWX
        RWYS(ISF) = RWY
        RWZS(ISF) = RWZ
        RHOS(ISF) = RHO
      END IF

      RETURN
      END


      SUBROUTINE SAVEM(ISF,MS)
C     This routine saves the Mach number in common for the S face ISF.
      INTEGER ISF
      REAL MS

      INCLUDE '3DCOM.FOR'

      MSS(ISF) = MS

      RETURN
      END
```

## File SETEXP.FOR

```
      SUBROUTINE SETEXP(SNEW,NN,DS1,DSTOT)
C     This routine sets up an exponentially stretched arc length array SNEW
C     of length NN given the first interval length DS1 and the total
C     length DSTOT.
      REAL SNEW(1),DS1,DSTOT
      INTEGER NN

      REAL TMP,RNEX,RATIO,RES,DRESDR,DRATIO,DD,RLX,DS
      INTEGER NEX,ITER,N
C
      TMP = DSTOT/DS1
      NEX = NN-1
      RNEX = FLOAT(NEX)
      RATIO = (TMP/RNEX)**(1.0/(RNEX-1.0)) + 0.1
      DO 1 ITER=1, 100
        RES = RATIO**NEX - 1.0 + (1.0 - RATIO)*TMP
        DRESDR = RNEX*RATIO**(NEX-1) - TMP
        DRATIO = -RES/DRESDR
C
        DD = DRATIO/RATIO
        RLX = 1.0
        IF(RLX*DD .GT. 2.0) RLX = 2.0/DD
        IF(RLX*DD .LT. -.5) RLX = -.5/DD
        RATIO = RATIO + RLX*DRATIO
        IF(ABS(DD) .LT. 1.0E-4) GO TO 11
    1 CONTINUE
      WRITE(6,*) 'SETEXP: Convergence failed.  Continuing anyway ...'
C
   11 SNEW(1) = 0.0
      DS = DS1
      DO 2 N=2, NN
        SNEW(N) = SNEW(N-1) + DS
        DS = DS*RATIO
    2 CONTINUE
C
      RETURN
      END ! SETEXP
```

## File SIINT.FOR

```
        SUBROUTINE SIINT(PSI1,PSI2,V,NI,NJ,PSI1I,PSI2I,VI)
C       This routine is for S (entropy) or I (rothalpy) interpolation
C       based on stream function.  There are (NI-1)*(NJ-1) values of
C       PSI1, PSI2 and V with i increasing fastest.  V is either entropy
C       or rothalpy.  VI is the interpolated value of V for the given PSI1I
C       and PSI2I.
        REAL PSI1(1),PSI2(1),V(1),PSI1I,PSI2I,VI
        INTEGER NI,NJ

        INTEGER IST,JST
        COMMON /SIINTC/IST,JST

        INTEGER I,J,N,IPQ,INQ,ILLC,JLLC
        REAL NUM,DEN
        REAL PSI1N,PSI2N,PSI11,PSI21,PSI12,PSI22,PSI13,PSI23,PSI14,PSI24
        REAL TWOPI
        DATA TWOPI/6.283185/
        REAL TH,TH1,TH2,TH3,TH4,V1,V2,V3,V4,C1,C2,C3,C4,XI(4),YI(4)

C       If NI=2 or NJ=2, use LSPFIT.
        IF(NI.EQ.2)THEN
          CALL LSPFIT(PSI2,V,NJ-1,PSI2I,VI,1,0)
          RETURN
        ELSE IF(NJ.EQ.2)THEN
          CALL LSPFIT(PSI1,V,NI-1,PSI1I,VI,1,0)
          RETURN
        END IF

C       Determine which quadralateral to use.
        IF(IST.GE.1 .AND. IST.LT.NI)THEN
          I = IST
        ELSE
          I = 1
        END IF
        IF(JST.GE.1 .AND. JST.LT.NJ)THEN
          J = JST
        ELSE
          J = 1
        END IF

C       INQ is the new quadrant indicator.
        INQ = 0

10      CONTINUE

C       IPQ is the previous quadrant indicator.
        IPQ = INQ

C       Determine the quadrant the point is in.

        N = (J-1)*(NI-1) + I

        PSI1N = PSI1(N)
        PSI2N = PSI2(N)

        NUM = PSI2I - PSI2N
        DEN = PSI1I - PSI1N
        IF(NUM.EQ.0. .AND. DEN.EQ.0.)THEN
```

```
C         Interpolated point is at I, J.
          VI = V(N)
          IST = I
          JST = J
          RETURN
        ELSE
          TH = ATAN2(NUM,DEN)
        END IF

C       Point 1.
        IF(I.EQ.NI-1)THEN
C         Extrapolate from point 3.
          PSI11 = 2*PSI1N - PSI1((J-1)*(NI-1) + I-1)
          PSI21 = 2*PSI2N - PSI2((J-1)*(NI-1) + I-1)
        ELSE
          PSI11 = PSI1((J-1)*(NI-1) + I+1)
          PSI21 = PSI2((J-1)*(NI-1) + I+1)
        END IF

C       Point 3.
        IF(I.EQ.1)THEN
C         Extrapolate from point 1.
          PSI13 = 2*PSI1N - PSI1((J-1)*(NI-1) + I+1)
          PSI23 = 2*PSI2N - PSI2((J-1)*(NI-1) + I+1)
        ELSE
          PSI13 = PSI1((J-1)*(NI-1) + I-1)
          PSI23 = PSI2((J-1)*(NI-1) + I-1)
        END IF

C       Point 2.
        IF(J.EQ.NJ-1)THEN
C         Extrapolate from point 4.
          PSI12 = 2*PSI1N - PSI1((J-2)*(NI-1) + I)
          PSI22 = 2*PSI2N - PSI2((J-2)*(NI-1) + I)
        ELSE
          PSI12 = PSI1((J)*(NI-1) + I)
          PSI22 = PSI2((J)*(NI-1) + I)
        END IF

C       Point 4.
        IF(J.EQ.1)THEN
C         Extrapolate from point 2.
          PSI14 = 2*PSI1N - PSI1((J)*(NI-1) + I)
          PSI24 = 2*PSI2N - PSI2((J)*(NI-1) + I)
        ELSE
          PSI14 = PSI1((J-2)*(NI-1) + I)
          PSI24 = PSI2((J-2)*(NI-1) + I)
        END IF

        TH1 = ATAN2(PSI21-PSI2N,PSI11-PSI1N)
        TH2 = ATAN2(PSI22-PSI2N,PSI12-PSI1N)
        TH3 = ATAN2(PSI23-PSI2N,PSI13-PSI1N)
        TH4 = ATAN2(PSI24-PSI2N,PSI14-PSI1N)

        IF(TH2.LT.TH1)TH2 = TH2 + TWOPI
        IF(TH3.LT.TH2)TH3 = TH3 + TWOPI
        IF(TH4.LT.TH3)TH4 = TH4 + TWOPI

        IF(TH.LT.TH1)TH = TH + TWOPI

C       INQ is the new quadrant.
        IF(TH.LE.TH2 .AND. TH.GE.TH1)THEN
          INQ = 1
        ELSE IF(TH.LT.TH3 .AND. TH.GT.TH2)THEN
          INQ = 2
        ELSE IF(TH.LE.TH4 .AND. TH.GE.TH3)THEN
```

```
          INQ = 3
        ELSE
          INQ = 4
        END IF

        IF(INQ.EQ.1 .AND. IPQ.EQ.3)THEN
C         Interpolate using quad 1.
          ILLC = I
          JLLC = J
        ELSE IF(INQ.EQ.2 .AND. IPQ.EQ.4)THEN
C         Interpolate using quad 2.
          ILLC = I - 1
          JLLC = J
        ELSE IF(INQ.EQ.3 .AND. IPQ.EQ.1)THEN
C         Interpolate using quad 3.
          ILLC = I - 1
          JLLC = J - 1
        ELSE IF(INQ.EQ.4 .AND. IPQ.EQ.2)THEN
C         Interpolate using quad 4.
          ILLC = I
          JLLC = J - 1
        ELSE
C         Quadralateral not found in interior.

          IF(INQ.EQ.1)THEN

            IF(I.EQ.NI-1)THEN
              IF(IPQ.EQ.4 .OR. J.EQ.NJ-2)THEN
C               Interpolate using quad 2.
                ILLC = I - 1
                JLLC = J
              ELSE IF(J.EQ.NJ-1)THEN
C               Interpolate using quad 3.
                ILLC = I - 1
                JLLC = J - 1
              ELSE
                J = J + 1
                GOTO 10
              END IF
            ELSE IF(J.EQ.NJ-1)THEN
              IF(IPQ.EQ.2)THEN
C               Interpolate using quad 4.
                ILLC = I
                JLLC = J - 1
              ELSE
                I = I + 1
                GOTO 10
              END IF
            ELSE
              I = I + 1
              J = J + 1
              GOTO 10
            END IF

          ELSE IF(INQ.EQ.2)THEN

            IF(I.EQ.1)THEN
              IF(IPQ.EQ.3 .OR. J.EQ.NJ-2)THEN
C               Interpolate using quad 1.
                ILLC = I
                JLLC = J
              ELSE IF(J.EQ.NJ-1)THEN
C               Interpolate using quad 4.
                ILLC = I
                JLLC = J - 1
              ELSE
```

```
                J = J + 1
                GOTO 10
              END IF
            ELSE IF(J.EQ.NJ-1)THEN
              IF(IPQ.EQ.1)THEN
C               Interpolate using quad 3.
                ILLC = I - 1
                JLLC = J - 1
              ELSE
                I = I - 1
                GOTO 10
              END IF
            ELSE
              I = I - 1
              J = J + 1
              GOTO 10
            END IF

          ELSE IF(INQ.EQ.3)THEN

            IF(I.EQ.1)THEN
              IF(IPQ.EQ.2 .OR. J.EQ.2)THEN
C               Interpolate using quad 4.
                ILLC = I
                JLLC = J - 1
              ELSE IF(J.EQ.1)THEN
C               Interpolate using quad 1.
                ILLC = I
                JLLC = J
              ELSE
                J = J - 1
                GOTO 10
              END IF
            ELSE IF(J.EQ.1)THEN
              IF(IPQ.EQ.4)THEN
C               Interpolate using quad 2.
                ILLC = I - 1
                JLLC = J
              ELSE
                I = I - 1
                GOTO 10
              END IF
            ELSE
              I = I - 1
              J = J - 1
              GOTO 10
            END IF

          ELSE IF(INQ.EQ.4)THEN

            IF(I.EQ.NI-1)THEN
              IF(IPQ.EQ.1 .OR. J.EQ.2)THEN
C               Interpolate using quad 3.
                ILLC = I - 1
                JLLC = J - 1
              ELSE IF(J.EQ.1)THEN
C               Interpolate using quad 2.
                ILLC = I - 1
                JLLC = J
              ELSE
                J = J - 1
                GOTO 10
              END IF
            ELSE IF(J.EQ.1)THEN
              IF(IPQ.EQ.3)THEN
C               Interpolate using quad 1.
```

```
                ILLC = I
                JLLC = J
              ELSE
                I = I + 1
                GOTO 10
              END IF
            ELSE
              I = I + 1
              J = J - 1
              GOTO 10
            END IF

          END IF
        END IF

C       Interpolate using bi-linear patch.
        IF(ILLC.EQ.NI-1 .OR. JLLC.EQ.NJ-1)THEN
          PRINT*,'Error in SIINT.'
        END IF

C       Point 1 is Lower left and points go counter clockwise.
        N = (JLLC-1)*(NI-1) + ILLC
        XI(1) = PSI1(N)
        YI(1) = PSI2(N)
        V1 = V(N)

C       Points 2 and 3.
        N = N + 1
        XI(2) = PSI1(N)
        YI(2) = PSI2(N)
        V2 = V(N)

        N = N + (NI-1)
        XI(3) = PSI1(N)
        YI(3) = PSI2(N)
        V3 = V(N)

C       Point 4.
        N = (JLLC)*(NI-1) + ILLC
        XI(4) = PSI1(N)
        YI(4) = PSI2(N)
        V4 = V(N)

        CALL BLIC(XI,YI,PSI1I,PSI2I,C1,C2,C3,C4)

        VI = C1*V1 + C2*V2 + C3*V3 + C4*V4

        RETURN
        END

        SUBROUTINE BLIC(XI,YI,X,Y,C1,C2,C3,C4)
C       This routine calculates the bilinear coefficients c1-c4 given
C       the coodinate of the point (x,y), and the coordinates of the
C       nodes of the element (xi(1),yi(1)) - (xi(4),yi(4)).
        REAL XI(4),YI(4),X,Y,C1,C2,C3,C4

        REAL AR(4),AS(4)
        DATA AR/1.,-1.,-1.,1./
        DATA AS/1.,1.,-1.,-1./

        DOUBLE PRECISION BLC(4,2),AQ,BQ,CQ,R,S,RAT,S1,S2,DEN

        INTEGER I,J

C       a1 is BLC(1,1) , b1 is BLC(2,1), etc.
        DO I=1,4
        DO J=1,2
          BLC(I,J) = 0.
        END DO
      END DO

      DO I=1,4
        BLC(1,1) = BLC(1,1) + AR(I)*XI(I)
        BLC(1,2) = BLC(1,2) + AR(I)*YI(I)
        BLC(2,1) = BLC(2,1) + AS(I)*XI(I)
        BLC(2,2) = BLC(2,2) + AS(I)*YI(I)
        BLC(3,1) = BLC(3,1) + AR(I)*AS(I)*XI(I)
        BLC(3,2) = BLC(3,2) + AR(I)*AS(I)*YI(I)
        BLC(4,1) = BLC(4,1) + XI(I)
        BLC(4,2) = BLC(4,2) + YI(I)
      END DO

C     The point is (X,Y).
      BLC(4,1) = BLC(4,1) - 4.*X
      BLC(4,2) = BLC(4,2) - 4.*Y


      AQ = BLC(3,1)*BLC(2,2) - BLC(3,2)*BLC(2,1)
      BQ = BLC(3,1)*BLC(4,2) + BLC(1,1)*BLC(2,2) - BLC(3,2)*BLC(4,1) -
     1       BLC(1,2)*BLC(2,1)
      CQ = BLC(1,1)*BLC(4,2) - BLC(1,2)*BLC(4,1)

      IF(AQ.EQ.0.)THEN
        IF(BQ.EQ.0.)THEN
          PRINT*,'S IS UNDEFINED IN BLIC.'
          S = 0.
        ELSE
          S = -CQ/BQ
        END IF
      ELSE
        RAT = BQ*BQ - 4.*AQ*CQ
        IF(RAT.LT.0.)THEN
          PRINT*,'S IS UNDEFINED IN BLIC.'
          S = 0.
        ELSE
          S1 = (-BQ + SQRT(RAT))/(2.*AQ)
          S2 = (-BQ - SQRT(RAT))/(2.*AQ)
C         Choose the root closest to 0.
          IF(ABS(S1).GT.ABS(S2))THEN
            S = S2
          ELSE
            S = S1
          END IF
        END IF
      END IF

      DEN = BLC(1,2) + BLC(3,2)*S
      IF(DEN.EQ.0.)THEN
        DEN = BLC(1,1) + BLC(3,1)*S
        IF(DEN.EQ.0.)THEN
          PRINT*,'R IS UNDEFINED IN INTCOF.'
          R = 0.
        ELSE
          R = -(BLC(4,1) + BLC(2,1)*S)/DEN
        END IF
      ELSE
        R = -(BLC(4,2) + BLC(2,2)*S)/DEN
      END IF

      C1 = 0.25*(1.+R)*(1.+S)
      C2 = 0.25*(1.-R)*(1.+S)
      C3 = 0.25*(1.-R)*(1.-S)
      C4 = 0.25*(1.+R)*(1.-S)

      RETURN
      END
```

## File SKYSOL.FOR

```
      SUBROUTINE SKYSOL(NEQ,ISSPI,IISPI,NDROWI,IRHSI, IFREE,
      1       S,IS,IWS, IOPT,EPSM,K,GMDISK,LPERM,IERR)
C     This routine solves the matrix using the skyline factorization
C     scheme with RCM or specified ordering.
C
C    In order to use GMRES, the matrix and other free storage must
C    all be in one large array S which is the same as the integer IS.
C    NEQ is the size of the matrix.
C    ISSPI is the zeroth index of SP in S.
C    IISPI is the zeroth index of ISP in S.
C    NDROWI is the zeroth index of NDROW in S.
C    IRHSI is the zeroth index of RHS in S.
C    IFREE is the zeroth index of free space in S.
C    IWS is the amount of free space after IFREE.
C     SP, ISP and NDROW are the matrix systematically packed.
C     RHS is the right hand side on input and the solution on output.
C     The free part of S must be big enough to hold the factored matrix
C     + 5*neq + k*(k+5) + nscr.
C      nscr = 0 if GMDISK = .true.
C      nscr = (k+1)*neq if GMDISK = .false.
C     IOPT is the option indicator.
C         = 0 factor matrix.
C         > 1 it is the file code which stores the matrix factor.
C             if this is the first time with a different file code,
C             the file is opened, the matrix is factored, and the matrix
C             written to the file.
C             if a factor is already in a file, it is read and used to
C             iteratively solve the matrix using GMRES.  If there is trouble
C             solving the matrix, the matrix is refactored and it is written
C             to the file.
C         < 1 the matrix is factored this iteration, but the factor is
C             stored in the file code = ABS(IOPT) for possible use during
C             the next iteration.
C     EPSM is the convergence criterion.
C     K is the krylov space to search before refactoring.
C     GMDISK is the logical flag which indicates whether to use
C     the disk file to hold the extra GMRES vectors .TRUE. or not .FALSE.
C     LPERM is the logical indicator of whether to use the RCM ordering
C     .TRUE. or to have the permutation vector stored in
C       IS(IFREE + NEQ + 1:IFREE + 2*NEQ) .FALSE.. If the matrix is read
C       from a file, then that ordering is used.
C     IERR = 0 if successful.
      INTEGER NEQ,ISSPI,IISPI,NDROWI,IRHSI,IFREE,IWS,IS(1)
      INTEGER IOPT,K,IERR
      REAL S(1),EPSM
      LOGICAL GMDISK,LPERM

      INTEGER IASIZ,I
      LOGICAL REFACT,NEWFAC
      INTEGER IFCP
      COMMON /SKYINI/IFCP
      DATA IFCP/0/
      EXTERNAL MATSOL
      LOGICAL FXUP
      INTEGER NQ,INCPRD,MAXCYC,IPC,IPU,NCYC,IER,NSCR
      REAL RDCTOL,RELTOL,RDCRES,RELRES
      INTEGER NEQR
      INTEGER NBLOCK,NREC,NDATA

      INTEGER ISSP,IISP,INDROW,IRHS,IX,IPERM,IINVP,
     1        IFX,ITEMP,ID,IKDIAG,IA,IHES,ISCR
      COMMON /SKYWS/ISSP,IISP,INDROW,IRHS,IX,IPERM,IINVP,
     1               IFX,ITEMP,ID,IKDIAG,IA,IHES,ISCR

      DOUBLE PRECISION BNORM,ERR
      DOUBLE PRECISION XNORM
      REAL XMAX
      REAL PT1,PT2
      INTEGER IOPTA

C     S will be carried through to GMRES.

C     The following are the zeroth indices in S or IS.
      ISSP = ISSPI
      IISP = IISPI
      INDROW = NDROWI
      IRHS = IRHSI
      IX = IFREE
      IPERM = IX + NEQ
      IINVP = IPERM + NEQ
      IFX = IINVP + NEQ
      ITEMP = IFX + NEQ
      ID = ITEMP + NEQ
      IKDIAG = ID + NEQ
      IA = IKDIAG + 2*NEQ

      IERR = 0
      PRINT*,'NEQ = ',NEQ
      PRINT*,'Number of nonzeros in matrix is ',IS(INDROW+NEQ)

C     Normalize the matrix.
      CALL MDNRMS(NEQ,S(ISSP+1),IS(IISP+1),IS(INDROW+1),S(IRHS+1))

      NBLOCK = 500
      REFACT = .FALSE.

10    CONTINUE

      IF(IOPT.LE.0 .OR. IOPT.NE.IFCP .OR. REFACT)THEN
C       Get the factor and permutation from calculation.

        CALL PTIME(PT1)

C       Get the permutation based on RCM if not already in IS.
        IF(LPERM)THEN
          CALL RCMPV(NEQ,IS(IISP+1),IS(INDROW+1), IS(IPERM+1),
     1       IS(IFX+1))
        END IF

C       Get the inverse permutation vector.
        CALL INVRTP(NEQ,IS(IPERM+1),IS(IINVP+1))

        CALL PTIME(PT2)
        PRINT*,'Time to order matrix is ',PT2-PT1

C       Set up the matrix factor based on the permutation.
        IASIZ = IWS - (IA - IFREE)
        CALL PFCPSS(NEQ,S(ISSP+1),IS(IISP+1),IS(INDROW+1),
     1   IS(IPERM+1),IS(IINVP+1), IS(IKDIAG+1),S(IA+1),
     1   IASIZ,IERR)

        IF(IERR.GT.0)THEN
          PRINT*,'Error setting up permuted matrix factor.'
          IF(IERR.EQ.2)THEN
            PRINT*,'Size needed is:',IASIZ
            PRINT*,'Size available is:',IWS - (IA - IFREE)
          END IF
          RETURN
        END IF

        PRINT*,'Size of matrix factor is ',IASIZ

C       Factor the matrix.
        CALL PTIME(PT1)
        CALL FACS(S(IA+1),NEQ,S(IKDIAG+1),IERR)
        IF(IERR.NE.0)THEN
          PRINT*,'Error factoring matrix.'
          RETURN
        END IF

        CALL PTIME(PT2)
        PRINT*,'Time to factor matrix is ',PT2-PT1

        IF(IOPT.NE.0)THEN
C         Write the factored matrix.
          IOPTA = IABS(IOPT)
          IF(IOPTA.NE.IFCP)THEN
C           Open the file IOPTA.
            OPEN(UNIT=IOPTA,STATUS='UNKNOWN',FORM='UNFORMATTED')
C            OPEN(UNIT=IOPTA,STATUS='NEW',
C    &         FORM='UNFORMATTED',RECORDTYPE='VARIABLE')
            IFCP = IOPTA
          END IF

          PRINT*,'Writing the matrix factor to file',IOPTA

          REWIND (IOPTA)

C         Write NEQ and IASIZ.
          WRITE(IOPTA)NEQ,IASIZ

C         Write KDIAG.
          NDATA = 2*NEQ
          NREC = (NDATA + NBLOCK - 1)/NBLOCK
          CALL WRTER(IOPTA,NREC,NDATA,NBLOCK,IS(IKDIAG+1))

C         Write PERM.
          NDATA = NEQ
          NREC = (NDATA + NBLOCK - 1)/NBLOCK
          CALL WRTER(IOPTA,NREC,NDATA,NBLOCK,IS(IPERM+1))

C         Write A.
          NDATA = IASIZ
          NREC = (NDATA + NBLOCK - 1)/NBLOCK
          CALL WRTER(IOPTA,NREC,NDATA,NBLOCK,S(IA+1))
        END IF

        NEWFAC = .TRUE.
      ELSE
C       Get the factor and permutation from file IOPT.
        PRINT*,'Reading the matrix factor from file',IOPT

        REWIND (IOPT)

C       Read NEQ and IASIZ.
        READ(IOPT)NEQR,IASIZ

        IF(NEQR.NE.NEQ)THEN
C         This should never happen.
```

```
            PRINT*,'Error in SKYSOL'
            IERR = 1
            RETURN
          END IF

C         Read KDIAG.
          NDATA = 2*NEQ
          NREC = (NDATA + NBLOCK - 1)/NBLOCK
          CALL READR(IOPT,NREC,NDATA,NBLOCK,IS(IKDIAG+1))

C         Read PERM.
          NDATA = NEQ
          NREC = (NDATA + NBLOCK - 1)/NBLOCK
          CALL READR(IOPT,NREC,NDATA,NBLOCK,IS(IPERM+1))

C         Read A.
          NDATA = IASIZ
          NREC = (NDATA + NBLOCK - 1)/NBLOCK
          CALL READR(IOPT,NREC,NDATA,NBLOCK,S(IA+1))

C         Get the inverse permutation vector.
          CALL INVRTP(NEQ,IS(IPERM+1),IS(IINVP+1))

          NEWFAC = .FALSE.
        END IF

C       Permute the right hand side.
        DO I=1,NEQ
          S(IX+I) = S(IRHS + IS(IPERM+I))
        END DO

C       Solve for the permuted right hand side using GMRES.

        BNORM = 0.
        DO I=1,NEQ
          BNORM = BNORM + S(IRHS+I)**2
        END DO
        BNORM = SQRT(BNORM)
        PRINT*,'BNORM in SKYSOL',BNORM

C       Initialize the solution vector.
        CALL MATSOL(2,NEQ,IPU, S(IX+1), IER,S(IFX+1), S)
        IF(IER.NE.0)THEN
          Print*,'Error in MATSOL.'
          IERR = 1
          RETURN
        END IF

C       Permute the solution back.  Store in TEMP.
        DO I=1,NEQ
          S(ITEMP+I) = S(IX + IS(IINVP+I))
        END DO

        CALL MSESSS(NEQ,S(ISSP+1),IS(IISP+1),IS(INDROW+1),
     1  S(IRHS+1), S(ITEMP+1), ERR)
        PRINT*,'After initialization in SKYSOL, ERR = ',ERR

C       Get the L2 norm of x and the max of x.
        XNORM = 0.
        XMAX = 0.
        DO I=1,NEQ
          XNORM = XNORM + S(IX+I)**2
          XMAX = MAX(XMAX,ABS(S(IX+I)))
        END DO
        XNORM = SQRT(XNORM)
        PRINT*,'XNORM in SKYSOL',XNORM
```

```
      PRINT*,'XMAX in SKYSOL',XMAX

C     Initialize parameters.
      IER = 0
      FXUP = .TRUE.
      IF(GMDISK)THEN
        NQ = 15
      ELSE
        NQ = 0
      END IF
      INCPRD = 0
      MAXCYC = 1
      IPC = 2
      IPU = 6
C     The error tolerances.
      RDCTOL = 0.
      RELTOL = 0.

      IHES = IA + IASIZ
      ISCR = IHES + K*(K+5)
      IF(GMDISK)THEN
        NSCR = 0
      ELSE
        NSCR = (K+1)*NEQ
      END IF

      PRINT*,IWS - (ISCR+NSCR - IFREE),' WORDS OF MEMORY LEFT IN S.'

      IF(ISCR+NSCR-IFREE.GT.IWS)THEN
        PRINT*,'Memory exceeded in SKYMAT.'
        IERR = 1
        RETURN
      END IF

      CALL PTIME(PT1)

C     Single Prec GMRES.
      CALL HSGMRS(MATSOL,FXUP, NQ,NEQ,K,INCPRD,MAXCYC, IPC,IPU,
      1       RDCTOL,RELTOL,S(IX+1), S(IFX+1),S(IHES+1),S(ISCR+1),
      1       NSCR,S, RDCRES,RELRES,NCYC,IER)

      CALL PTIME(PT2)
      PRINT*,'Time in GMRES is ',PT2-PT1

C     Permute the solution back.  Store in TEMP.
      DO I=1,NEQ
        S(ITEMP+I) = S(IX + IS(IINVP+I))
      END DO

      CALL MSESSS(NEQ,S(ISSP+1),IS(IISP+1),IS(INDROW+1),
     1 S(IRHS+1), S(ITEMP+1), ERR)
      PRINT*,'Error in matrix solution is ',ERR

C     Get the L2 norm of x and the max of x.
      XNORM = 0.
      XMAX = 0.
      DO I=1,NEQ
        XNORM = XNORM + S(IX+I)**2
        XMAX = MAX(XMAX,ABS(S(IX+I)))
      END DO
      XNORM = SQRT(XNORM)
      PRINT*,'XNORM in SKYSOL',XNORM
      PRINT*,'XMAX in SKYSOL',XMAX

C     If GMRES did not converge, then if this was an old factor,
C     refactor it.
```

```
      IF(IER.NE.0 .AND. IER.NE.1)THEN
        IF( ERR/BNORM.GT.EPSM .AND. .NOT.NEWFAC)THEN
          REFACT = .TRUE.
          GOTO 10
        END IF
      END IF

C     Put the solution into RHS.
      DO I=1,NEQ
        S(IRHS+I) = S(ITEMP+I)
      END DO

      RETURN
      END

      SUBROUTINE MATSOL(IND,NEQ,IPU, X, IER,FX, WFCN)
C     This is the update routine which uses a previous matrix factor to
C     update the solution.
      INTEGER IND,NEQ,IPU,IER
      REAL X(1),FX(1),WFCN(1)

      INTEGER ISSP,IISP,INDROW,IRHS,IX,IPERM,IINVP,
     1        IFX,ITEMP,ID,IKDIAG,IA,IHES,ISCR
      COMMON /SKYWS/ISSP,IISP,INDROW,IRHS,IX,IPERM,IINVP,
     1              IFX,ITEMP,ID,IKDIAG,IA,IHES,ISCR

      INTEGER IR

      INTEGER I
      DOUBLE PRECISION XNORM
      REAL XMAX

      IER = 0

      IF(IND.EQ.2)THEN
C       Find D.  X Must contain the permuted right hand side.
        CALL SOLS(WFCN(IA+1),NEQ,WFCN(IKDIAG+1),X)
        DO IR=1,NEQ
          WFCN(ID+IR) = X(IR)
        END DO
      ELSE IF(IND.EQ.0)THEN
C       fx = J x + d.

C       Multiply X by PAP^(-1).  Put in FX.
        CALL MATMLT(NEQ,WFCN(ISSP+1),WFCN(IISP+1),WFCN(INDROW+1),
     1   WFCN(IPERM+1),WFCN(IINVP+1),X, FX, WFCN(ITEMP+1))

C       Subtract FX from b which is P rhs.
        CALL ERRVEC(NEQ,FX,WFCN(IPERM+1),WFCN(IRHS+1))

        CALL SOLS(WFCN(IA+1),NEQ,WFCN(IKDIAG+1),FX)

CC    Get the L2 norm of Fx and the max of Fx.
C     XNORM = 0.
C      XMAX = 0.
C     DO I=1,NEQ
C       XNORM = XNORM + FX(I)**2
C        XMAX = MAX(XMAX,ABS(FX(I)))
C     END DO
C     XNORM = SQRT(XNORM)
C     PRINT*,'XNORM in SKYSOL',XNORM
C     PRINT*,'XMAX in SKYSOL',XMAX


        DO IR=1,NEQ
C         FX(IR) = X(IR) - FX(IR) + WFCN(ID+IR)
```

```
            FX(IR) = X(IR) + FX(IR)
          END DO

         ELSE IF(IND.EQ.-1)THEN
C         fx = J x.

C         Multiply X by PAP^(-1).  Put in FX.
          CALL MATMLT(NEQ,WFCN(ISSP+1),WFCN(IISP+1),WFCN(INDROW+1),
     1     WFCN(IPERM+1),WFCN(IINVP+1),X, FX, WFCN(ITEMP+1))

C         Subtract FX from b which is P rhs.
          CALL ERRVEC(NEQ,FX,WFCN(IPERM+1),WFCN(IRHS+1))

          CALL SOLS(WFCN(IA+1),NEQ,WFCN(IKDIAG+1),FX)
          DO IR=1,NEQ
C           FX(IR) = X(IR) - FX(IR)
            FX(IR) = X(IR) + FX(IR) - WFCN(ID+IR)
          END DO
        END IF

        IF(IER.NE.0)CALL HHERR(3,'MATSOL',IER,0)

        RETURN
        END



        SUBROUTINE ERRVEC(NEQ,AX,PERM,RHS)
C       This routine subtracts b from AX and stores it back in AX.
C       B is P rhs.
        INTEGER NEQ,PERM(1)
        REAL AX(1),RHS(1)

        INTEGER I

        DO I=1,NEQ
          AX(I) = RHS( PERM(I) ) - AX(I)
        END DO

        RETURN
        END

        SUBROUTINE MATMLT(NEQ,SP,ISP,NDROW,PERM,INVP,X, PAPTX, TEMP)
C       This routine multiplies the vector X by PAP^(-1) or PAP(transpose)
C       and returns the result in PAPTX.  The matrix is in SP,ISP and NDROW.
C       The permutations are in PERM and INVP.  TEMP is temporary storage
C       of length NEQ.
        INTEGER NEQ,ISP(1),NDROW(1),PERM(1),INVP(1)
        REAL SP(1),X(1),PAPTX(1),TEMP(1)

        INTEGER IR

C       First inverse permute X.
C       Store in PAPTX.

        DO IR=1,NEQ
          PAPTX(IR) = X( INVP(IR) )
        END DO

        CALL MVPSSS(NEQ,SP,ISP,NDROW,PAPTX, TEMP)

C       Permute the result back to PAPTX.
        DO IR=1,NEQ
          PAPTX(IR) = TEMP( PERM(IR) )
        END DO
```

```
      RETURN
      END

      SUBROUTINE INVRTP(NEQ,PERM,INVP)
C     This routine gets the inverse permutation vector INVP form the
C     permutation vector PERM.
      INTEGER NEQ,PERM(1),INVP(1)

      INTEGER I

      DO I=1,NEQ
        INVP(PERM(I)) = I
      END DO

      RETURN
      END
```

## File SPLINE.FOR

```
      SUBROUTINE SPLINE(X,XS,S,N)
      PARAMETER (NX=480)
      DIMENSION X(1),XS(1),S(1)
      DIMENSION A(NX),B(NX),C(NX)
C-------------------------------------------------------
C     Calculates spline coefficients for X(S).          |
C     To evaluate the spline at some value of S,        |
C     use SEVAL and/or DEVAL.                           |
C                                                       |
C     S        independent variable array  (input)      |
C     X        dependent variable array    (input)      |
C     XS       dX/dS array                 (calculated) |
C     N        number of points            (input)      |
C                                                       |
C-------------------------------------------------------
      IF(N.GT.NX) STOP 'SPLINE: array overflow'
C
      DO 1 I=2, N-1
        DSM = S(I) - S(I-1)
        DSP = S(I+1) - S(I)
        B(I) = DSP
        A(I) = 2.0*(DSM+DSP)
        C(I) = DSM
        XS(I) = 3.0*((X(I+1)-X(I))*DSM/DSP + (X(I)-X(I-1))*DSP/DSM)
    1 CONTINUE
C
C---- set specified first derivative end conditions
C      XS1 = 0.
C      XS2 = 0.
C      A(1) = 1.0
C      C(1) = 0.
C      XS(1) = XS1
C      A(N) = 1.0
C      B(N) = 0.
C      XS(N) = XS2
C
C---- set zero second derivative end conditions
      A(1) = 2.0
      C(1) = 1.0
      XS(1) = 3.0*(X(2)-X(1)) / (S(2)-S(1))
      B(N) = 1.0
      A(N) = 2.0
      XS(N) = 3.0*(X(N)-X(N-1)) / (S(N)-S(N-1))
C
      CALL TRISOL(A,B,C,XS,N)
      RETURN
      END ! SPLINE


      SUBROUTINE TRISOL(A,B,C,D,KK)
      DIMENSION A(1),B(1),C(1),D(1)
C-----------------------------------------
C     Solves KK long, tri-diagonal system |
C                                         |
C             A C         D               |
C             B A C       D               |
C               B A .     .               |
C                 . . C   .               |
C                   B A   D               |
C                                         |
C     The righthand side D is replaced by |
C     the solution.  A, C are destroyed.  |
C-----------------------------------------
C
      DO 1 K=2, KK
        KM = K-1
        C(KM) = C(KM) / A(KM)
        D(KM) = D(KM) / A(KM)
        A(K) = A(K) - B(K)*C(KM)
        D(K) = D(K) - B(K)*D(KM)
    1 CONTINUE
C
      D(KK) = D(KK)/A(KK)
C
      DO 2 K=KK-1, 1, -1
        D(K) = D(K) - C(K)*D(K+1)
    2 CONTINUE
C
      RETURN
      END ! TRISOL


      FUNCTION SEVAL(SS,X,XS,S,N)
      DIMENSION X(1), XS(1), S(1)
C-------------------------------------------------
C     Calculates X(SS)                            |
C     XS array must have been calculated by SPLINE |
C-------------------------------------------------
      ILOW = 1
      I = N
C
   10 IF(I-ILOW .LE. 1) GO TO 11
C
      IMID = (I+ILOW)/2
      IF(SS .LT. S(IMID)) THEN
       I = IMID
      ELSE
       ILOW = IMID
      ENDIF
      GO TO 10
C
   11 DS = S(I) - S(I-1)
      T = (SS - S(I-1)) / DS
      CX1 = DS*XS(I-1) - X(I) + X(I-1)
      CX2 = DS*XS(I)   - X(I) + X(I-1)
      SEVAL = T*X(I) + (1.0-T)*X(I-1) + (T-T*T)*((1.0-T)*CX1 - T*CX2)
      RETURN
      END ! SEVAL

      FUNCTION DEVAL(SS,X,XS,S,N)
      DIMENSION X(1), XS(1), S(1)
C-------------------------------------------------
```

```
C     Calculates dX/dS(SS)                         |
C     XS array must have been calculated by SPLINE |
C-------------------------------------------------
      ILOW = 1
      I = N
C
   10 IF(I-ILOW .LE. 1) GO TO 11
C
      IMID = (I+ILOW)/2
      IF(SS .LT. S(IMID)) THEN
       I = IMID
      ELSE
       ILOW = IMID
      ENDIF
      GO TO 10
C
   11 DS = S(I) - S(I-1)
      T = (SS - S(I-1)) / DS
      CX1 = DS*XS(I-1) - X(I) + X(I-1)
      CX2 = DS*XS(I)   - X(I) + X(I-1)
      DEVAL = X(I) - X(I-1) + (1.-4.0*T+3.0*T*T)*CX1 + T*(3.0*T-2.)*CX2
      DEVAL = DEVAL/DS
      RETURN
      END ! DEVAL

      FUNCTION D2VAL(SS,X,XS,S,N)
      DIMENSION X(1), XS(1), S(1)
C-------------------------------------------------
C     Calculates d2X/dS2(SS)                       |
C     XS array must have been calculated by SPLINE |
C-------------------------------------------------
      ILOW = 1
      I = N
C
   10 IF(I-ILOW .LE. 1) GO TO 11
C
      IMID = (I+ILOW)/2
      IF(SS .LT. S(IMID)) THEN
       I = IMID
      ELSE
       ILOW = IMID
      ENDIF
      GO TO 10
C
   11 DS = S(I) - S(I-1)
      T = (SS - S(I-1)) / DS
      CX1 = DS*XS(I-1) - X(I) + X(I-1)
      CX2 = DS*XS(I)   - X(I) + X(I-1)
      D2VAL = (6.*T-4.)*CX1 + (6.*T-2.0)*CX2
      D2VAL = D2VAL/DS**2
      RETURN
      END ! D2VAL


      FUNCTION CURV(SS,X,XS,Y,YS,S,N)
      DIMENSION X(1), XS(1), Y(1), YS(1), S(1)
C-----------------------------------------------
C     Calculates curvature of splined 2-D curve |
C     at S = SS                                 |
C                                               |
C     S         arc length array of curve       |
C     X, Y      coordinate arrays of curve      |
C     XS,YS     derivative arrays               |
C               (calculated earlier by SPLINE)  |
C-----------------------------------------------
C
```

```
      ILOW = 1
      I = N
C
   10 IF(I-ILOW .LE. 1) GO TO 11
C
      IMID = (I+ILOW)/2
      IF(SS .LT. S(IMID)) THEN
       I = IMID
      ELSE
       ILOW = IMID
      ENDIF
      GO TO 10
C
   11 DS = S(I) - S(I-1)
      T = (SS - S(I-1)) / DS
C
      CX1 = DS*XS(I-1) - X(I) + X(I-1)
      CX2 = DS*XS(I)   - X(I) + X(I-1)
      XD = X(I) - X(I-1) + (1.0-4.0*T+3.0*T*T)*CX1 + T*(3.0*T-2.0)*CX2
      XDD = (6.0*T-4.0)*CX1 + (6.0*T-2.0)*CX2
C
      CY1 = DS*YS(I-1) - Y(I) + Y(I-1)
      CY2 = DS*YS(I)   - Y(I) + Y(I-1)
      YD = Y(I) - Y(I-1) + (1.0-4.0*T+3.0*T*T)*CY1 + T*(3.0*T-2.0)*CY2
      YDD = (6.0*T-4.0)*CY1 + (6.0*T-2.0)*CY2
C
      CURV = (XD*YDD - YD*XDD) / (DS*(XD*XD + YD*YD))
C
      RETURN
      END ! CURV

      FUNCTION CURVS(SS,X,XS,Y,YS,S,N)
      DIMENSION X(1), XS(1), Y(1), YS(1), S(1)
C-------------------------------------------------
C     Calculates curvature derivative of         |
C     splined 2-D curve at S = SS                |
C                                                |
C     S        arc length array of curve         |
C     X, Y     coordinate arrays of curve        |
C     XS,YS    derivative arrays                 |
C              (calculated earlier by SPLINE)    |
C-------------------------------------------------
C
      ILOW = 1
      I = N
C
   10 IF(I-ILOW .LE. 1) GO TO 11
C
      IMID = (I+ILOW)/2
      IF(SS .LT. S(IMID)) THEN
       I = IMID
      ELSE
       ILOW = IMID
      ENDIF
      GO TO 10
C
   11 DS = S(I) - S(I-1)
      T = (SS - S(I-1)) / DS
C
      CX1 = DS*XS(I-1) - X(I) + X(I-1)
      CX2 = DS*XS(I)   - X(I) + X(I-1)
      XD = X(I) - X(I-1) + (1.0-4.0*T+3.0*T*T)*CX1 + T*(3.0*T-2.0)*CX2
      XDD = (6.0*T-4.0)*CX1 + (6.0*T-2.0)*CX2
      XDDD = 6.0*CX1 + 6.0*CX2
C
      CY1 = DS*YS(I-1) - Y(I) + Y(I-1)
      CY2 = DS*YS(I)    - Y(I) + Y(I-1)
      YD = Y(I) - Y(I-1) + (1.0-4.0*T+3.0*T*T)*CY1 + T*(3.0*T-2.0)*CY2
      YDD = (6.0*T-4.0)*CY1 + (6.0*T-2.0)*CY2
      YDDD = 6.0*CY1 + 6.0*CY2
C
      BOT = XD*XD + YD*YD
      DBOTDT = 2.0*XD*XDD + 2.0*YD*YDD
C
      TOP = XD*YDD - YD*XDD
      DTOPDT = XD*YDDD - YD*XDDD
      CURVS = (DTOPDT*BOT - DBOTDT*TOP) / (DS*BOT)**2
C
      RETURN
      END ! CURVS

      SUBROUTINE SINVRT(SI,XI,X,XS,S,N)
C-------------------------------------------------
C     Calculates the "inverse" spline function S(X). |
C     Since S(X) can be multi-valued or not defined, |
C      this is not a "black-box" routine.  The call- |
C      ing program must pass via SI a sufficiently   |
C      good initial guess for S(XI).                 |
C                                                    |
C     XI      specified X value       (input)        |
C     SI      calculated S(XI) value  (input,output) |
C     X,XS,S  usual spline arrays     (input)        |
C                                                    |
C-------------------------------------------------
C
      DO 10 ITER=1, 10
        RES  = SEVAL(SI,X,XS,S,N) - XI
        RESP = DEVAL(SI,X,XS,S,N)
        DS = -RES/RESP
        SI = SI + DS
        IF(ABS(DS).LT.1.0E-5) RETURN
   10 CONTINUE
      STOP 'SINVRT: spline inversion failed'
      END ! SINVRT

      SUBROUTINE SCALC(X,Y,S,N)
      DIMENSION X(1), Y(1), S(1)
C-----------------------------------
C     Calculates the arc length array S  |
C     for a 2-D array of points (X,Y).   |
C-----------------------------------
C
      S(1) = 0.
      DO 10 I=2, N
        S(I) = S(I-1) + SQRT((X(I)-X(I-1))**2 + (Y(I)-Y(I-1))**2)
   10 CONTINUE
C
      RETURN
      END ! SCALC
```

## File SPSLV.FOR

```
        SUBROUTINE SPSLV(NEQ,SP,ISP,NDROW,RHS,S,IWS)
C       This routine solves the matrix using SPARSPAK.
        INTEGER NEQ,ISP(1),NDROW(1),IWS
        REAL SP(1),RHS(1),S(1)

        INTEGER NNZA,ISBEG,IR,IPP,NINR,METHOD
        REAL RELERR
        INTEGER MSGLVA,IERRA,MAXSA,NEQNS
        COMMON /SPAUSR/ MSGLVA,IERRA,MAXSA,NEQNS
```

```
C       PRINT*,'What sparspak mthod do you want? '
C        PRINT*,'0-RCM, 1-min deg, 2-nd, 3-one-way disect, 4-refined quot.'
C       READ*,METHOD
          METHOD = 0

          CALL SPRSPK

C         Print some extra statistics.
          MSGLVA = 3
c         MSGLVA = 2

          NNZA = NDROW(NEQ)

          MAXSA = 3*(NNZA + 5*NEQ + 1)
C         MAXSA = 100000
          ISBEG = 1

          CALL IJBEGN

C         Input the structure of each row.
          IPP = 1
          DO IR=1,NEQ
            NINR = NDROW(IR) - IPP + 1
            CALL INROW(IR,NINR,ISP(IPP), S(ISBEG) )
            IPP = NDROW(IR) + 1
          END DO

          CALL IJEND( S(ISBEG) )

          MAXSA = IWS

C         Order the matrix.
          IF(METHOD.EQ.0)THEN
            CALL ORDRA2( S(ISBEG) )
          ELSE IF(METHOD.EQ.1)THEN
            CALL ORDRB6( S(ISBEG) )
          ELSE IF(METHOD.EQ.2)THEN
            CALL ORDRA6( S(ISBEG) )
          ELSE IF(METHOD.EQ.3)THEN
            CALL ORDRA4( S(ISBEG) )
          ELSE IF(METHOD.EQ.4)THEN
            CALL ORDRB4( S(ISBEG) )
          END IF
          IF(IERRA.GT.0)THEN
C           Print Statistics.
            CALL STATSA
            PRINT*,'Error in sparspak.  Cannot continue. ierra=',IERRA
            STOP
          END IF

C         Input the values of each row.
          IPP = 1
          DO IR=1,NEQ
            NINR = NDROW(IR) - IPP + 1
            IF(METHOD.EQ.0)THEN
              CALL INROW2(IR,NINR,ISP(IPP),SP(IPP), S(ISBEG) )
            ELSE IF(METHOD.EQ.1 .OR. METHOD.EQ.2)THEN
              CALL INROW6(IR,NINR,ISP(IPP),SP(IPP), S(ISBEG) )
            ELSE IF(METHOD.EQ.3 .OR. METHOD.EQ.4)THEN
              CALL INROW4(IR,NINR,ISP(IPP),SP(IPP), S(ISBEG) )
            END IF
            IPP = NDROW(IR) + 1
          END DO

C         Input the right hand side.
```

```
          CALL INRHS(RHS, S(ISBEG) )

C         Solve the matrix.
          IF(METHOD.EQ.0)THEN
            CALL SOLVE2( S(ISBEG) )
          ELSE IF(METHOD.EQ.1 .OR. METHOD.EQ.2)THEN
            CALL SOLVE6( S(ISBEG) )
          ELSE IF(METHOD.EQ.3 .OR. METHOD.EQ.4)THEN
            CALL SOLVE4( S(ISBEG) )
          END IF
          IF(IERRA.GT.0)THEN
            PRINT*,'Error in sparspak.  Cannot continue. ierra=',IERRA
            STOP
          END IF

C         Put the solution in RHS.
          DO IR=1,NEQ
            RHS(IR) = S(ISBEG-1+IR)
          END DO

C         Get an error estimate.
          IF(METHOD.EQ.0)THEN
            CALL EREST2(RELERR, S(ISBEG) )
          ELSE IF(METHOD.EQ.1 .OR. METHOD.EQ.2)THEN
            CALL EREST6(RELERR, S(ISBEG) )
          ELSE IF(METHOD.EQ.3 .OR. METHOD.EQ.4)THEN
            CALL EREST4(RELERR, S(ISBEG) )
          END IF

C         Reset error flag if error indicator = -1.
          IF(IERRA.EQ.164)IERRA = 0

C         Print Statistics.
          CALL STATSA

          PRINT*,'Relative error.',RELERR

        RETURN
        END
```

## File TARAMAT1.FOR

```
C***********************************************************************
C
C       From DOTPR.FOR
C
C***********************************************************************
C                       DOT PRoduct routines.

C       This file contains several dot product functions for
C       various options of type of vector and precision.
C       It also contains 2 vector pivot routines VMODS and VMODD.
C               Each Dot product function is of the form
C                       D23456(vector1,vector2).
C       Location 2 is:  S for a single precision function,
C                       D for a double precision function.
C       Location 3 is the type of the first vector:
C                       C is a complete contiguous vector.  This
C                               would be V1.
C                       P is a packed vector for which only the
C                               nonzero components are stored.  The
C                               coefficients are stored in SP1 and the
C                               corresponding columns are in ISP1.  N1
C                               is the number of coefficients in the first
```

```
C                                  vector.
C       Location 4 is the type of the second vector:
C                        C is a complete contiguous vector.  This
C                                  would be V2.  If the first vector is also
C                                  complete, N is the number of elements in
C                                  both vectors.  Otherwise V2 spans the range
C                                  of vector1.
C                        P is a packed vector for which only the
C                                  nonzero components are stored.  The
C                                  coefficients are stored in SP2 and the
C                                  corresponding columns are in ISP2.  N2
C                                  is the number of coefficients in the second
C                                  vector.
C       Location 5 is the precision of the first vector:
C                        S is single,
C                        D is Double.
C       Location 6 is the precision of the second vector:
C                        S is single,
C                        D is Double.
C

        REAL FUNCTION DSCCSS(V1,V2,N)
C       Dot Single precision function, Complete Complete Single Single.
        REAL V1(1)
        REAL V2(1)
        INTEGER N

        INTEGER I

        DSCCSS = 0.

        DO I=1,N
          DSCCSS = DSCCSS + V1(I)*V2(I)
        END DO

        RETURN
        END


        DOUBLE PRECISION FUNCTION DDCCSS(V1,V2,N)
C       Dot Double precision function, Complete Complete Single Single.
        REAL V1(1)
        REAL V2(1)
        INTEGER N

        INTEGER I

        DDCCSS = 0.

        DO I=1,N
          DDCCSS = DDCCSS + V1(I)*V2(I)
        END DO

        RETURN
        END


        SUBROUTINE VPIVS(A,B,X,N)
C       Vector PIVot, Single precision.
C
C       This routine performs the pivot operation A = A + xB.
C       A an B are vectors.
C       X is the scalar.
C       N is the length of A and B.
C       A, B and X are Single precision.
C
```

```
      REAL A(1),B(1),X
      INTEGER N

      INTEGER I

      DO 10 I=1,N
10    A(I) = X*B(I) + A(I)

      RETURN
      END

C*********************************************************************
C
C     From MATOP.FOR
C
C*********************************************************************
C                       MATrix OPeration routines.

C     The routines in this file do matrix operations such as
C     Matrix-vector multiplies, error calculations, and matrix
C     normalizations for a matrix stored in compact form.


      SUBROUTINE MVPSSS(N,SP,ISP,NDROW,V, PROD)
C     Matrix-Vector Product, Single matrix, Single v, Single product.
C
C     This routine calculates Mv and stores it in PROD.
C
C     N is the size of the matrix.
C     SP, ISP, and NDROW represent the matrix in compact form.
C     V is the vector to be multiplied.
C     PROD is the vector result.
C     SP is in Single precision.
C     V is in Single precision.
C     PROD is in Single precision.
C
      INTEGER N
      REAL SP(1)
      INTEGER ISP(1),NDROW(1)
      REAL V(1),PROD(1)

      INTEGER IR,IP,IRC

C     Calculate M v.
      IP = 1
      DO IR=1,N
        PROD(IR) = 0.
        DO IRC=IP,NDROW(IR)
          PROD(IR) = PROD(IR) + SP(IRC)*V(ISP(IRC))
        END DO
        IP = NDROW(IR) + 1
      END DO

      RETURN
      END

      SUBROUTINE MSESSS(NEQ,SP,ISP,NDROW,RHS,X, ERR)
C     Matrix Solution Error Single matrix, Single rhs, Single soln.
C
C     This routine calculates the l2 norm of the matrix error (rhs - Mx).
C     This is similar to the root mean square (rms) except the
C     result is not divided by NEQ.
C
C     NEQ is the size of the matrix.
C     SP, ISP, and NDROW represent the matrix in compact form.
C     RHS is the right hand side vector.
```

```
C       X is the current solution vector.
C       ERR is the 12 norm of the matrix error.
C       SP is in Single precision.
C       RHS is in Single precision.
C       X is in Single precision.
C       ERR is in Double precision.
C
        INTEGER NEQ
        REAL SP(1)
        INTEGER ISP(1),NDROW(1)
        REAL RHS(1)
        REAL X(1)
        DOUBLE PRECISION ERR

        INTEGER IP,IR,L
        DOUBLE PRECISION ROWERR

        ERR = 0.
        IP = 1
        DO IR=1,NEQ
          ROWERR = RHS(IR)
          DO L=IP,NDROW(IR)
            ROWERR = ROWERR - SP(L)*X(ISP(L))
          END DO
          IP = NDROW(IR) + 1
          ERR = ERR + ROWERR**2
        END DO
        ERR = SQRT(ERR)

        RETURN
        END

        SUBROUTINE MDNRMS(NEQ,SP,ISP,NDROW,B)
C       Matrix Diagonal NoRMalization, Single precision.
C
C       This routine will normalize the matrix by the diagonal.
C       If no diagonal, it will be normalized by the max coef.
C
C       NEQ is the size of the matrix.
C       SP, ISP, and NDROW represent the matrix in compact form.
C       B is the right hand side vector.
C       SP and B are in Single precision.
C
        INTEGER NEQ
        REAL SP(1)
        INTEGER ISP(1),NDROW(1)
        REAL B(1)

        INTEGER IR,IP,I
        REAL AMAXR,AC,RAC

        IP = 1
        DO IR=1,NEQ
          AMAXR = 0.
          DO I=IP,NDROW(IR)
            IF(ISP(I).EQ.IR)THEN
              IF(SP(I).NE.0.)THEN
                RAC = 1./SP(I)
                GOTO 10
              END IF
            END IF
            AC = ABS(SP(I))
            IF(AC.GT.AMAXR)AMAXR = AC
          END DO
          IF(AMAXR.NE.0.)THEN
            RAC = 1./AMAXR
          ELSE
            RAC = 1
          END IF

10        CONTINUE

          DO I=IP,NDROW(IR)
            SP(I) = SP(I)*RAC
          END DO
          B(IR) = B(IR)*RAC
          IP = NDROW(IR) + 1
        END DO

        RETURN
        END

C*********************************************************************
C
C       From SKYLIN.FOR
C
C*********************************************************************
C                  SKYLINe matrix solution routines.

C       These routines consist of a preprocessor, factorization and
C       forward-backward solution subroutines for solving a matrix
C       stored in compact form using a skyline or profile storage scheme.
C       The matrices are assumed nonsymmetric both in values and structure.

C       There is only one requirement for the column ordering in the
C       SP and ISP arrays for the factorization preprocessors.  That is
C       that the first column for a given row is in the first location
C       in the SP and ISP arrays for that row (namely NDROW(ir-1) + 1).
C       The rest of the ordering does not matter for creating the KDIAG
C       array and initializing the matrix A in skyline mode.

        SUBROUTINE PFCPSS(NN,SP,ISP,NDROW,PERM,INVP, KDIAG,A,IASIZ,IERR)
C       Preprocessor for FaC with Permutation, Single matrix, Single factor.
C
C            This routine is similar to PFACSS, except the matrix which is
C       set up is permuted based on the permutation vector PERM,
C       and its transpose INVP.  FAC is called in the same manner.  However
C       the right hand side input to SOL must be permuted, and the resulting
C       solution vector must be inverse permuted.
C       input:
C           NN  the absolute value of NN is N, the number of equations.
C               If NN is positive, KDIAG is set up.  If NN is negative,
C               KDIAG has already been set up and is available on input.
C          SP, ISP and NDROW represent the matrix in compact storage.
C         PERM is the permutation vector.
C         INVP is the inverse permutation vector.
C       input or output:
C         KDIAG is the coefficient pointer used in FACS, FACD, SOLS and SOLD -
C               must be dimensioned to (2,N).  input if NN < 0;
C               output if NN > 0.
C         IASIZ is the size of A available on input.  On output,
C               it is set to KDIAG(2,N) which is size of A used.
C       output:
C           A   coeficient matrix in nonsymmetric skyline storage
C               used in FACS and SOLS- must be at least dimensioned
C               to 2N so KDIAG can be calculated.
C          IERR is the error code:
C               = 0 successful.
C               = 1 size of A not large enough to calculate KDIAG.
C               = 2 size of A not large enough for factor.  Size
C                       needed is in IASIZ.
C               = 3 coefficient calculation is not correct.
```

```
C               = -ir, then the diagonal of row ir is 0.
C
C       SP is Single precision.
C       A is Single precision.
C
        INTEGER NN
        REAL SP(1)
        INTEGER ISP(1),NDROW(1),KDIAG(2,1),PERM(1),INVP(1)
        REAL A(1)
        INTEGER IASIZ,IERR

        INTEGER N,I,IP,IR,IC,ICOF,IACOF,IRO

        N = ABS(NN)
        IERR = 0

        IF(IASIZ.LT.2*N)THEN
C         A not big enough to be used in SETCP.
          IERR = 1
          RETURN
        END IF

C       Set up the coefficient pointer if NN > 0.  A has not been set
C       and is being use in SETCP for two integer working arrays.
        IF(NN.GT.0)
     1  CALL SETCPP(N,ISP,NDROW,PERM,INVP,KDIAG,A,A(N+1), IERR)

        IF(IERR.NE.0)RETURN

        IF(IASIZ.LT.KDIAG(2,N))THEN
C         A not big enough for factor.
          IERR = 2
          IASIZ = KDIAG(2,N)
          RETURN
        END IF

        IASIZ = KDIAG(2,N)

C       Initialize the A matrix.
        DO 10 I=1,IASIZ
          A(I) = 0.
 10     CONTINUE

C       Put the coefficients in SP in the proper address in A.  The
C       function ICOF finds the proper address given the row and column
C       in addition to the pointer array KDIAG.
        IP = 1
        DO 30 IRO=1,N
          IR = INVP(IRO)
          DO 20 I=IP,NDROW(IRO)
            IC = INVP(ISP(I))
            IACOF = ICOF(IR,IC,KDIAG)
            IF(IACOF.LE.0 .OR. IACOF.GT.IASIZ)THEN
              IERR = 3
              RETURN
            END IF
            A(IACOF) = SP(I)
 20       CONTINUE
          IP = NDROW(IRO) + 1
 30     CONTINUE

        RETURN
        END

        SUBROUTINE SETCPP(N,ISP,NDROW,PERM,INVP,KDIAG,ICOL,ICMIN, IERR)
C       SET Coefficient Pointer with Permutation vector.
```

```
C
C       Used internally to PFCPSS, PFCPSD, and PFCPDD.
C
C       ICOL is a working storage column array of dimension N.
C       ICMIN is a working storage column array of dimension N.
C       IERR is the error code.
C               = 0 successful
C               = -ir no diagonal for column ir.
C
        INTEGER N,ISP(1),NDROW(1),PERM(1),INVP(1),KDIAG(2,1),ICOL(1)
        INTEGER ICMIN(1),IERR

        INTEGER I,IR,IRO,IP,IC,IRM1,IDIAG

C       Initialize ICOL's to 0 and ICMIN to N+1.
        DO 10 IR=1,N
          ICOL(IR) = 0
          ICMIN(IR) = N + 1
10      CONTINUE

C       Search for the first permuted row in which coefficients will be in a
C       column.  Also make sure the diagonals have coefficients.

C       For the permuted row, determine the mininimum column number.

        DO 30 IR=1,N
          IRO = PERM(IR)
          IF(IRO.EQ.1)THEN
            IP = 1
          ELSE
            IP = NDROW(IRO-1) + 1
          END IF
          IDIAG = 0
          DO 20 I=IP,NDROW(IRO)
            IC = INVP(ISP(I))
            ICMIN(IR) = MIN(ICMIN(IR),IC)
            IF(IC.EQ.IR)IDIAG = 1
            IF(ICOL(IC).EQ.0)ICOL(IC) = IR
20        CONTINUE
          IF(IDIAG.EQ.0)THEN
C           No diagonal for this row.
            IERR = -IR
            IF(ICOL(IR).EQ.0)ICOL(IR) = IR
          END IF
30      CONTINUE

C       Set up KDIAG's.
        KDIAG(1,1) = 0
        KDIAG(2,1) = 1

        DO 40 IR=2,N
          IRM1 = IR - 1
          KDIAG(1,IR) = KDIAG(2,IRM1) + IR - ICOL(IR)
          KDIAG(2,IR) = KDIAG(1,IR) + IR - ICMIN(IR) + 1
40      CONTINUE

        RETURN
        END

        INTEGER FUNCTION ICOF(IR,IC,KDIAG)
C       Index COeFficient calculation.
C
C       Used internally to PFACSS, PFACSD, PFACDD, PFCPSS, PFCPSD, and PFCPDD.
C       The index for the matrix factor stored in skyline mode is calculated
C       from the row IR, column IC and KDIAG.
C
```

```
        INTEGER IR,IC,KDIAG(2,1)

        IF(IC.LE.IR)THEN
          ICOF=KDIAG(2,IR) + IC - IR
        ELSE
          ICOF=KDIAG(1,IC) + IR - IC +1
        END IF

        RETURN
        END

C****************************************************
C            From SRT.FOR
C****************************************************
C                         SoRT routines.

C            These are the matrix sort routines.

        SUBROUTINE ISRTI(ID,IX,N)
C       Integer quick SoRT Index routine.
C
C       IX is the index array to the N elements of ID in increasing order.
C
        INTEGER ID(1),IX(1),N

        INTEGER I(25),L(25),J,M,IT,K,IXIM

C       INITIALIZE INDEX
        DO 5 J=1,N
5         IX(J)=J

C       ACTUAL MAXIMUM DIMENSION OF I,L IS LOG BASE 2 OF N
        M=1
        I(1)=1
        L(1)=N

C       BEGIN LOOP 1 - RETURNS FROM AFTER 50 AND AFTER 70
10      J=I(M)
        K=L(M)+1

C       SEGMENT M MAY HAVE 0 ELEMENTS  - I(M) OR L(M) MAY BE OUT OF BOUNDS
        IT=K-J-2
        IF(IT) 50,15,15
15      IXIM=IX(J)
        IF(IT) 110,20,110

C       SEGMENT M HAS 2 MEMBERS - SORT THEM
20      IF(ID(IXIM) .LE.ID(IX(J +1))) GO TO 50
        IX(J )=IX(J +1)
        IX(J +1)=IXIM
50      M=M-1
        IF(M) 10,90,10

90      RETURN

C       SEGMENT M HAS OVER 2 MEMBERS - PARTITION INTO 3 SEGMENTS

C       BEGIN LOOP 2 - FIND FIRST ENTRY TO BE MOVED TO END
110     J=J+1
        IF(K.EQ.J) GO TO 180
        IF(ID(IXIM) .GE.ID(IX(J))) GO TO 110

C       FIND LAST ENTRY TO BE MOVED TO FRONT (LOOP 3)
120     K=K-1

C       IF NONE, TAIL SEGMENT BEGINS AT K=J
```

```
        IF(K.EQ.J) GO TO 180
        IF(ID(IXIM) .LE.ID(IX(K))) GO TO 120

C       SWAP IX(J) AND IX(K)
        IT=IX(J)
        IX(J)=IX(K)
        IX(K)=IT
        GO TO 110

C       PUT FIRST ELEMENT BETWEEN 2 NEW SEGMENTS AT K-1
180     IX(I(M))=IX(K-1)
        IX(K-1)=IXIM

C       LARGER SEGMENT IS NEW SEGMENT M - SIZES ARE K-I(M)-1 , L(M)-K+1
        IF(K*2-L(M)-I(M)-2) 55,60,60
55      I(M+1)=I(M)
        I(M)=K
        L(M+1)=K-2
        GO TO 70
60      I(M+1)=K
        L(M+1)=L(M)
        L(M)=K-2

C       WORK ON SHORT SEGMENT NEXT
70      M=M+1

        IF(M.LT.14) GO TO 10

        WRITE(6,1000)
 1000   FORMAT('NUMBER OF SEGMENTS IN QUICK SORT EXCEEDS DIMENSION')
        RETURN

        END

        SUBROUTINE ISRT(ID,N)
C       Integer SoRT routine.
C
C       Integer sort routine which sorts N elements of ID array.
C
C       ID is the array to be sorted.
C       N is the number of elements in the array.
C
        INTEGER ID(1),N

        INTEGER M,I,J,L,T

        M=N/2

 20     IF(.NOT.(M.GT.0))GOTO 22
        DO 23 J=1,N-M
          I=J
 25       IF(.NOT.(I.GT.0))GOTO 23
          L=I+M
          IF(.NOT.(ID(I).LE.ID(L)))GOTO 28
          GOTO 23
 28       CONTINUE
          T=ID(I)
          ID(I)=ID(L)
          ID(L)=T
          I=I-M
          GOTO 25
 23     CONTINUE

        M=M/2
        GOTO 20
```

```
22      CONTINUE

        RETURN
        END
C****************************************************
C            From PERM.FOR
C****************************************************
C                            matrix PERMutation routions.

C       These routines create a permutation vector which reorders a matrix.
C       The Reverse Cuthill-McKee (RCM) algorithm is currently the only one
C       available.

        SUBROUTINE RCMPV(NEQ,ISP,NDROW, PERM, WS)
C       Reverse Cuthill-McKee Permutation Vector routine.
C
C       This routine finds the permutation vector PERM for an RCM ordering
C       given the matrix stored in ISP and NDROW.
C       NEQ is the size of the matrix.
C       ISP, and NDROW represent the matrix structure in compact form.
C       PERM is the permutation vector.
C       WS is the working storage and should be 4*NDROW(NEQ) + 2*NEQ long.
C
        INTEGER NEQ,ISP(1),NDROW(1),PERM(1),WS(1)

        INTEGER IXADJ,IAJNCY,IISRTC,IIROW,IIRC,IMASK,IXLS

        IXADJ = 0
        IAJNCY = IXADJ + NEQ + 1
        IISRTC = IAJNCY + 2*NDROW(NEQ)
        IIROW = IISRTC + NDROW(NEQ)
        IIRC = IIROW + NDROW(NEQ)

C       Get the adjacency structure.
        CALL GETADJ(NEQ,ISP,NDROW,WS(IXADJ+1),WS(IAJNCY+1),
        1       WS(IISRTC+1),WS(IIROW+1),WS(IIRC+1))

        IMASK = IAJNCY + 2*NDROW(NEQ)
        IXLS = IMASK + NEQ

C       For the given adjacency structure, find PERM.
        CALL GENRCM(NEQ,WS(IXADJ+1),WS(IAJNCY+1),PERM,
        1       WS(IMASK+1),WS(IXLS+1))

        RETURN
        END


        SUBROUTINE GETADJ(N,ISP,NDROW,XADJ,ADJNCY,ISRTC,IROW,IRC)
C       GET ADJacency.
C
C       Used internally by RCMPV.
C
C       This routine finds the adjacency structure of the matrix
C       stored in row format using ISP and NDROW.  N is the number of equations.
C       ISP, ISRTC, and IROW must be dimensioned to NDROW(N).  ISRTC is
C       used as the index which takes rows to columns.  IROW is the row
C       storage.  (XADJ,ADJNCY) are the array pair which defines the graph.
C       XADJ must be dimensioned to N, and the size of ADJNCY is the number
C       of nonzero elements in the matrix plus its transpose minus the diag.
C       IRC is used to stored the row plus column temporarily and should
C       therefore be dimensioned accordingly.
        INTEGER N,ISP(1),NDROW(1)
        INTEGER XADJ(1),ADJNCY(1)
        INTEGER ISRTC(1),IROW(1),IRC(1)
```

```
      INTEGER IP,IR,ICOL,ISPC,NINCOL

C     Store rows in IROW array.
      IP = 1
      DO IR=1,N
        DO ISPC=IP,NDROW(IR)
          IROW(ISPC) = IR
        END DO
        IP = NDROW(IR) + 1
      END DO

C     Sort the columns.
      CALL ISRTI(ISP,ISRTC,NDROW(N))

C     Add the column to IRC.  When it is found, append the row.  Then
C     sort and eliminate the diagonal.
      ICOL = 1
      ISPC = 1
      NINCOL = 0
      XADJ(1) = 1

10    CONTINUE

      IF(ISP(ISRTC(ISPC)).EQ.ICOL)THEN
        IF(ISPC.EQ.NDROW(N))GOTO 20
        NINCOL = NINCOL + 1
        IRC(NINCOL) = IROW(ISRTC(ISPC))
        ISPC = ISPC + 1
        GOTO 10
      ELSE
C       End of column.
        CALL ARSED(IRC,NINCOL,ISP,NDROW,ICOL,XADJ,ADJNCY)
C       Initialize Next column.
        ICOL = ICOL + 1
        NINCOL = 0
        GOTO 10
      END IF

20    CONTINUE
C     Last column.
      CALL ARSED(IRC,NINCOL,ISP,NDROW,ICOL,XADJ,ADJNCY)

      RETURN
      END


      SUBROUTINE ARSED(IRC,NINCOL,ISP,NDROW,ICOL,XADJ,ADJNCY)
C     Append Row, Sort and Eliminate Diagonal.
C
C     Used internally by RCMPV.
C
C     This routine Appends the Row to IRC, Sorts IRC and Eliminates
C     the Diagonal.  The result is the neighbors of a node for the
C     symmetric matrix and these are put into the (XADJ,ADJNCY) pair.
      INTEGER IRC(1),NINCOL,ISP(1),NDROW(1),ICOL,XADJ(1),ADJNCY(1)

      INTEGER IP,I,IPRE,ADJBEG

C     Append the row to IRC.
      IF(ICOL.EQ.1)THEN
        IP = 1
      ELSE
        IP = NDROW(ICOL-1) + 1
      END IF

      DO I=IP,NDROW(ICOL)
```

```
           NINCOL = NINCOL + 1
           IRC(NINCOL) = ISP(I)
         END DO

C        Sort IRC.
         CALL ISRT(IRC,NINCOL)

C        Eliminate the diagonal and duplicates.
         IPRE = 0
         IP = 0
         ADJBEG = XADJ(ICOL) - 1
         DO I=1,NINCOL
           IF(IRC(I).EQ.IPRE)GOTO 10
           IF(IRC(I).EQ.ICOL)GOTO 10
           IP = IP + 1
           ADJNCY(ADJBEG+IP) = IRC(I)
           IPRE = IRC(I)
10         CONTINUE
         END DO

         XADJ(ICOL+1) = ADJBEG + IP + 1

         RETURN
         END
```

## File VOL.FOR

```
         SUBROUTINE RM(M)
C        This routine finds the residuals for the momentum equations
C        and stores them in FVEC(5*M-3), FVEC(5*M-2) and FVEC(5*M-1)
C        for the S, A and B momentum equations respectively.
         INTEGER M

         INCLUDE '3DCOM.FOR'
         INCLUDE 'MATCOM.FOR'
         INCLUDE 'FACE.FOR'

         INTEGER I,J,K,IA1F,IA2F,IB3F,IB4F,IS5F,IS6F
         REAL EQXM,EQYM,EQZM
         INTEGER IROW

         RESX = 0.
         RESY = 0.
         RESZ = 0.

C        Initialize the volume calculation variables used by the S face
C        if there are any source terms.  These will also be used to determine
C        the vectors to apply the A, B and S momentum equations.
         CALL SETVOL(M)

C        Find the contibutions from the six faces of the volume.

C        Get the indices for the volume.
         K = (M-1)/((NI-1)*(NJ-1)) + 2
         J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
         I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C        Get the face indices.
C          a1 : i = I, j = J, k = K
           IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C          a2 : a1 + 1,  i = I + 1, j = J, k = K
           IA2F = IA1F + 1
C          b3 : i = I, j = J, k = K
           IB3F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C          b4 : b3 + (ni-1),  i = I, j = J + 1, k = K
```

```
           IB4F = IB3F + (NI-1)
C          s5 : i = I, j = J, k = K - 1
           IS5F = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C          s6 : s5 + (ni-1)(nj-1),  i = I, j = J, k = K
           IS6F = IS5F + (NI-1)*(NJ-1)

C       A1 face.
        CALL RMA(IA1F,-1.)

C       A2 face.
        CALL RMA(IA2F,1.)

C       B3 face.
        CALL RMB(IB3F,-1.)

C       B4 face.
        CALL RMB(IB4F,1.)

C       S5 face.
        CALL RMS(IS5F,-1.)

C       S6 face.
        CALL RMS(IS6F,1.)

C       Get the S-momentum equation.
        EQXM = SMXC(M)
        EQYM = SMYC(M)
        EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
        IROW = 5*M - 3

        FVEC(IROW) = EQXM*RESX + EQYM*RESY + EQZM*RESZ

C       Get the A-momentum equation.
        EQXM = AMXC(M)
        EQYM = AMYC(M)
        EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
        IROW = 5*M - 2

        FVEC(IROW) = EQXM*RESX + EQYM*RESY + EQZM*RESZ

C       Get the B-momentum equation.
        EQXM = BMXC(M)
        EQYM = BMYC(M)
        EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
        IROW = 5*M - 1

        FVEC(IROW) = EQXM*RESX + EQYM*RESY + EQZM*RESZ

        RETURN
        END

        SUBROUTINE LM(M,AJ,JCN,JDROW,IWSOM)
C       This routine finds the left hand side for the momentum equations
C       and stores them in rows 5*M-3, 5*M-2 and 5*M-1 of the original matrix
C       for the S, A and B momentum equations respectively.
        INTEGER M
        REAL AJ(1)
        INTEGER JCN(1),JDROW(1),IWSOM(1)

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER NEQCAS
        COMMON /CASRT/NEQCAS
```

```
      INTEGER I,J,K,IA1F,IA2F,IB3F,IB4F,IS5F,IS6F
      REAL EQXM,EQYM,EQZM
      INTEGER IROW
      REAL DUMR

      NEQCAS = NSV

      NCX = 0
      NCY = 0
      NCZ = 0

C     Initialize the volume calculation variables used by the S face
C     if there are any source terms.  These will also be used to determine
C     the vectors to apply the A, B and S momentum equations.
      CALL SETVOL(M)

C     Find the contibutions from the six faces of the volume.

C     Get the indices for the volume.
      K = (M-1)/((NI-1)*(NJ-1)) + 2
      J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
      I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C     Get the face indices.
C       a1 : i = I, j = J, k = K
        IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C       a2 : a1 + 1,  i = I + 1, j = J, k = K
        IA2F = IA1F + 1
C       b3 : i = I, j = J, k = K
        IB3F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C       b4 : b3 + (ni-1),  i = I, j = J + 1, k = K
        IB4F = IB3F + (NI-1)
C       s5 : i = I, j = J, k = K - 1
        IS5F = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C       s6 : s5 + (ni-1)(nj-1),  i = I, j = J, k = K
        IS6F = IS5F + (NI-1)*(NJ-1)

C     A1 face.
      CALL LMA(IA1F,-1.)

C     A2 face.
      CALL LMA(IA2F,1.)

C     B3 face.
      CALL LMB(IB3F,-1.)

C     B4 face.
      CALL LMB(IB4F,1.)

C     Make sure the upstream S face is called first so the Mach number
C     upstream gets stored for the upwinded pressure calculation.

C     S5 face.
      CALL LMS(IS5F,-1.)

C     S6 face.
      CALL LMS(IS6F,1.)

C     The equations must be sorted.
C     The x-eq will be put in NEV=1.
C     The y-eq will be put in NEV=2.
C     The z-eq will be put in NEV=3.
      NEV = 1
      CALL CASRTS(COFX,ICOLX,NCX,1,
     1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),IWSOM)
      NEV = 2
      CALL CASRTS(COFY,ICOLY,NCY,1,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),IWSOM)
      NEV = 3
      CALL CASRTS(COFZ,ICOLZ,NCZ,1,
      1       COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),IWSOM)

C     Get the S-momentum equation.
      EQXM = SMXC(M)
      EQYM = SMYC(M)
      EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
      IROW = 5*M - 3

      IF(ISSMAP(M))THEN
C       S-momentum equation applied.
        CALL COM3EQ(COFV(1,1),ICOLV(1,1),NCV(1),RESX,
      1       COFV(1,2),ICOLV(1,2),NCV(2),RESY,
      1       COFV(1,3),ICOLV(1,3),NCV(3),RESZ,EQXM,EQYM,EQZM,
      1       COFA,ICOLA,NA,DUMR)
C       Put COFA into J matrix.
        CALL CASRTS(COFA,ICOLA,NA,IROW,AJ,JCN,JDROW,IWSOM)
      ELSE
        IF(ISUP.EQ.0 .AND. ICNTSM.EQ.0)THEN
C         There is constant entropy everywhere.  Set it.
          CALL LBENT(M+(NI-1)*(NJ-1))
        ELSE
C         Entropy transport equation applied.
          CALL ENTEQ(I,J,K,NI,NJ)
        END IF
C       Put COFX into J matrix.
        CALL CASRTS(COFX,ICOLX,NCX,IROW,AJ,JCN,JDROW,IWSOM)
      END IF


C     Get the A-momentum equation.
      EQXM = AMXC(M)
      EQYM = AMYC(M)
      EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
      IROW = 5*M - 2

      CALL COM3EQ(COFV(1,1),ICOLV(1,1),NCV(1),RESX,
      1       COFV(1,2),ICOLV(1,2),NCV(2),RESY,
      1       COFV(1,3),ICOLV(1,3),NCV(3),RESZ,EQXM,EQYM,EQZM,
      1       COFA,ICOLA,NA,DUMR)
C     Put COFA into J matrix.
      CALL CASRTS(COFA,ICOLA,NA,IROW,AJ,JCN,JDROW,IWSOM)

C     Get the B-momentum equation.
      EQXM = BMXC(M)
      EQYM = BMYC(M)
      EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
      IROW = 5*M - 1

      CALL COM3EQ(COFV(1,1),ICOLV(1,1),NCV(1),RESX,
      1       COFV(1,2),ICOLV(1,2),NCV(2),RESY,
      1       COFV(1,3),ICOLV(1,3),NCV(3),RESZ,EQXM,EQYM,EQZM,
      1       COFA,ICOLA,NA,DUMR)
C     Put COFA into J matrix.
      CALL CASRTS(COFA,ICOLA,NA,IROW,AJ,JCN,JDROW,IWSOM)

      RETURN
      END

      SUBROUTINE RMC(M)
C     This routine finds the residuals for the momentum equations
C     and stores them in FVEC(5*M-3), FVEC(5*M-2) and FVEC(5*M-1)
C     for the S, A and B momentum equations respectively for compressible
```

```
C       flow.
        INTEGER M

        INCLUDE '3DCOM.FOR'
        INCLUDE 'MATCOM.FOR'
        INCLUDE 'FACE.FOR'

        INTEGER I,J,K,IA1F,IA2F,IB3F,IB4F,IS5F,IS6F
        REAL EQXM,EQYM,EQZM
        INTEGER IROW

        RESX = 0.
        RESY = 0.
        RESZ = 0.

C       Initialize the volume calculation variables used by the S face
C       if there are any source terms.  These will also be used to determine
C       the vectors to apply the A, B and S momentum equations.
        CALL SETVOL(M)

C       Find the contibutions from the six faces of the volume.

C       Get the indices for the volume.
        K = (M-1)/((NI-1)*(NJ-1)) + 2
        J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
        I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C       Get the face indices.
C         a1 : i = I, j = J, k = K
          IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C         a2 : a1 + 1,  i = I + 1, j = J, k = K
          IA2F = IA1F + 1
C         b3 : i = I, j = J, k = K
          IB3F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C         b4 : b3 + (ni-1),  i = I, j = J + 1, k = K
          IB4F = IB3F + (NI-1)
C         s5 : i = I, j = J, k = K - 1
          IS5F = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C         s6 : s5 + (ni-1)(nj-1),  i = I, j = J, k = K
          IS6F = IS5F + (NI-1)*(NJ-1)

C       A1 face.
        CALL RMAC(IA1F,-1.)

C       A2 face.
        CALL RMAC(IA2F,1.)

C       B3 face.
        CALL RMBC(IB3F,-1.)

C       B4 face.
        CALL RMBC(IB4F,1.)

C       Make sure the upstream S face is called first so the Mach number
C       upstream gets stored for the upwinded pressure calculation.

C       S5 face.
        CALL RMSC(IS5F,-1.)

C       S6 face.
        CALL RMSC(IS6F,1.)

C       Get the S-momentum equation.
        EQXM = SMXC(M)
        EQYM = SMYC(M)
        EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
```

```
      IROW = 5*M - 3

      FVEC(IROW) = EQXM*RESX + EQYM*RESY + EQZM*RESZ

C     Get the A-momentum equation.
      EQXM = AMXC(M)
      EQYM = AMYC(M)
      EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
      IROW = 5*M - 2

      FVEC(IROW) = EQXM*RESX + EQYM*RESY + EQZM*RESZ

C     Get the B-momentum equation.
      EQXM = BMXC(M)
      EQYM = BMYC(M)
      EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
      IROW = 5*M - 1

      FVEC(IROW) = EQXM*RESX + EQYM*RESY + EQZM*RESZ

      RETURN
      END

      SUBROUTINE LMC(M,AJ,JCN,JDROW,IWSOM)
C     This routine finds the left hand side for the momentum equations
C     and stores them in rows 5*M-3, 5*M-2 and 5*M-1 of the original matrix
C     for the S, A and B momentum equations respectively for compressible
C     flow.
      INTEGER M
      REAL AJ(1)
      INTEGER JCN(1),JDROW(1),IWSOM(1)

      INCLUDE '3DCOM.FOR'
      INCLUDE 'MATCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER NEQCAS
      COMMON /CASRT/NEQCAS

      INTEGER I,J,K,IA1F,IA2F,IB3F,IB4F,IS5F,IS6F
      REAL EQXM,EQYM,EQZM
      INTEGER IROW
      REAL DUMR

      NEQCAS = NSV

      NCX = 0
      NCY = 0
      NCZ = 0

C     Initialize the volume calculation variables used by the S face
C     if there are any source terms.  These will also be used to determine
C     the vectors to apply the A, B and S momentum equations.
      CALL SETVOL(M)

C     Find the contibutions from the six faces of the volume.

C     Get the indices for the volume.
      K = (M-1)/((NI-1)*(NJ-1)) + 2
      J = (M - (K-2)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
      I = M - (K-2)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

C     Get the face indices.
C       a1 : i = I, j = J, k = K
        IA1F = (K-2)*NI*(NJ-1) + (J-1)*NI + I
C       a2 : a1 + 1,  i = I + 1, j = J, k = K
```

```
          IA2F = IA1F + 1
C         b3 : i = I, j = J, k = K
          IB3F = (K-2)*(NI-1)*NJ + (J-1)*(NI-1) + I
C         b4 : b3 + (ni-1),  i = I, j = J + 1, k = K
          IB4F = IB3F + (NI-1)
C         s5 : i = I, j = J, k = K - 1
          IS5F = (K-2)*(NI-1)*(NJ-1) + (J-1)*(NI-1) + I
C         s6 : s5 + (ni-1)(nj-1),  i = I, j = J, k = K
          IS6F = IS5F + (NI-1)*(NJ-1)

C       A1 face.
        CALL LMAC(IA1F,-1.)

C       A2 face.
        CALL LMAC(IA2F,1.)

C       B3 face.
        CALL LMBC(IB3F,-1.)

C       B4 face.
        CALL LMBC(IB4F,1.)

C       Make sure the upstream S face is called first so the Mach number
C       upstream gets stored for the upwinded pressure calculation.

C       S5 face.
        CALL LMSC(IS5F,-1.)

C       S6 face.
        CALL LMSC(IS6F,1.)

C       The equations must be sorted.
C       The x-eq will be put in NEV=1.
C       The y-eq will be put in NEV=2.
C       The z-eq will be put in NEV=3.
        NEV = 1
        CALL CASRTS(COFX,ICOLX,NCX,1,
        1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),IWSOM)
        NEV = 2
        CALL CASRTS(COFY,ICOLY,NCY,1,
        1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),IWSOM)
        NEV = 3
        CALL CASRTS(COFZ,ICOLZ,NCZ,1,
        1        COFV(1,NEV),ICOLV(1,NEV),NCV(NEV),IWSOM)

C       Get the S-momentum equation.
        EQXM = SMXC(M)
        EQYM = SMYC(M)
        EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
        IROW = 5*M - 3

        IF(ISSMAP(M))THEN
C         S-momentum equation applied.
          CALL COM3EQ(COFV(1,1),ICOLV(1,1),NCV(1),RESX,
        1        COFV(1,2),ICOLV(1,2),NCV(2),RESY,
        1        COFV(1,3),ICOLV(1,3),NCV(3),RESZ,EQXM,EQYM,EQZM,
        1        COFA,ICOLA,NA,DUMR)
C         Put COFA into J matrix.
          CALL CASRTS(COFA,ICOLA,NA,IROW,AJ,JCN,JDROW,IWSOM)
        ELSE
          IF(ISUP.EQ.0 .AND. ICNTSM.EQ.0)THEN
C           There is constant entropy everywhere.  Set it.
            CALL LBENT(M+(NI-1)*(NJ-1))
          ELSE
C           Entropy transport equation applied.
            CALL ENTEQ(I,J,K,NI,NJ)
            END IF
C           Put COFX into J matrix.
            CALL CASRTS(COFX,ICOLX,NCX,IROW,AJ,JCN,JDROW,IWSOM)
          END IF


C         Get the A-momentum equation.
          EQXM = AMXC(M)
          EQYM = AMYC(M)
          EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
          IROW = 5*M - 2

          CALL COM3EQ(COFV(1,1),ICOLV(1,1),NCV(1),RESX,
          1      COFV(1,2),ICOLV(1,2),NCV(2),RESY,
          1      COFV(1,3),ICOLV(1,3),NCV(3),RESZ,EQXM,EQYM,EQZM,
          1      COFA,ICOLA,NA,DUMR)
C         Put COFA into J matrix.
          CALL CASRTS(COFA,ICOLA,NA,IROW,AJ,JCN,JDROW,IWSOM)

C         Get the B-momentum equation.
          EQXM = BMXC(M)
          EQYM = BMYC(M)
          EQZM = SQRT( MAX(1. - EQXM**2 - EQYM**2,0.) )
          IROW = 5*M - 1

          CALL COM3EQ(COFV(1,1),ICOLV(1,1),NCV(1),RESX,
          1      COFV(1,2),ICOLV(1,2),NCV(2),RESY,
          1      COFV(1,3),ICOLV(1,3),NCV(3),RESZ,EQXM,EQYM,EQZM,
          1      COFA,ICOLA,NA,DUMR)
C         Put COFA into J matrix.
          CALL CASRTS(COFA,ICOLA,NA,IROW,AJ,JCN,JDROW,IWSOM)

          RETURN
          END

          SUBROUTINE SETVOL(M)
C         This routine sets up the variables in common which are used to
C         calculate the volume.
          INTEGER M

          INCLUDE '3DCOM.FOR'
          INCLUDE 'FACE.FOR'

          INTEGER I,J,K,ISFUP,ISFDN

C         Divide this into 2 ISF faces.
          ISFUP = M
          ISFDN = M + (NI-1)*(NJ-1)

C         Upstream face.
          K = (ISFUP-1)/((NI-1)*(NJ-1)) + 1
          J = (ISFUP - (K-1)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
          I = ISFUP - (K-1)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

          CALL FVNSF(I,J,K,LE,LF,LG,LH,
          1      X1EC,X2EC,X3EC,D1DLEE,D2DLEE,D3DLEE,
          1      X1FC,X2FC,X3FC,D1DLEF,D2DLEF,D3DLEF,
          1      X1GC,X2GC,X3GC,D1DLEG,D2DLEG,D3DLEG,
          1      X1HC,X2HC,X3HC,D1DLEH,D2DLEH,D3DLEH)

C         Downstream face.
          K = (ISFDN-1)/((NI-1)*(NJ-1)) + 1
          J = (ISFDN - (K-1)*(NI-1)*(NJ-1) - 1)/(NI-1) + 1
          I = ISFDN - (K-1)*(NI-1)*(NJ-1) - (J-1)*(NI-1)

          CALL FVNSF(I,J,K,LI,LJ,LK,LL,
     1         X1IC,X2IC,X3IC,D1DLEI,D2DLEI,D3DLEI,
     1         X1JC,X2JC,X3JC,D1DLEJ,D2DLEJ,D3DLEJ,
     1         X1KC,X2KC,X3KC,D1DLEK,D2DLEK,D3DLEK,
     1         X1LC,X2LC,X3LC,D1DLEL,D2DLEL,D3DLEL)

      JSLEL = J
      JSLEU = J + 1

      RETURN
      END


      SUBROUTINE GETVUV
C     This routine calculates and stores the volume unit vectors which are the
C     multiplies of the x, y, and z momentum equations.

      INCLUDE '3DCOM.FOR'
      INCLUDE 'FACE.FOR'

      INTEGER M
      REAL DX1,DY1,DZ1,DX2,DY2,DZ2,DX3,DY3,DZ3,SIZ
      REAL XE,XF,XG,XH,XI,XJ,XK,XL
      REAL YE,YF,YG,YH,YI,YJ,YK,YL
      REAL ZE,ZF,ZG,ZH,ZI,ZJ,ZK,ZL
      REAL DX1H,DY1H,DZ1H,DX2H,DY2H,DZ2H

      DO M=1,NM
C       Get the coordinates of the volume corners.
        CALL SETVOL(M)

C       Get the unit vectors of the volume A, B, and S directions.
        IF(CYLIND)THEN
          XE = X2EC*SIN(X1EC)
          XF = X2FC*SIN(X1FC)
          XG = X2GC*SIN(X1GC)
          XH = X2HC*SIN(X1HC)
          XI = X2IC*SIN(X1IC)
          XJ = X2JC*SIN(X1JC)
          XK = X2KC*SIN(X1KC)
          XL = X2LC*SIN(X1LC)

          YE = X2EC*COS(X1EC)
          YF = X2FC*COS(X1FC)
          YG = X2GC*COS(X1GC)
          YH = X2HC*COS(X1HC)
          YI = X2IC*COS(X1IC)
          YJ = X2JC*COS(X1JC)
          YK = X2KC*COS(X1KC)
          YL = X2LC*COS(X1LC)
        ELSE
          XE = X1EC
          XF = X1FC
          XG = X1GC
          XH = X1HC
          XI = X1IC
          XJ = X1JC
          XK = X1KC
          XL = X1LC

          YE = X2EC
          YF = X2FC
          YG = X2GC
          YH = X2HC
          YI = X2IC
          YJ = X2JC
          YK = X2KC
```

```
            YL = X2LC
          END IF

          ZE = X3EC
          ZF = X3FC
          ZG = X3GC
          ZH = X3HC
          ZI = X3IC
          ZJ = X3JC
          ZK = X3KC
          ZL = X3LC

C         The unit vector for the S momentum equation.
C         This is in the direction which connects the centers of the
C         two s faces.
          DX3 = 0.25*((XI + XJ + XK + XL) - (XE + XF + XG + XH))
          DY3 = 0.25*((YI + YJ + YK + YL) - (YE + YF + YG + YH))
          DZ3 = 0.25*((ZI + ZJ + ZK + ZL) - (ZE + ZF + ZG + ZH))
C         Normalize by its length, and make the sign such that DZ3 > 0.
          SIZ = SIGN( SQRT(DX3**2 + DY3**2 + DZ3**2) , DZ3 )
          DX3 = DX3/SIZ
          DY3 = DY3/SIZ
          DZ3 = DZ3/SIZ
          SMXC(M) = DX3
          SMYC(M) = DY3

          DX1 = 0.25*((XG + XH + XK + XL) - (XE + XF + XI + XJ))
          DY1 = 0.25*((YG + YH + YK + YL) - (YE + YF + YI + YJ))
          DZ1 = 0.25*((ZG + ZH + ZK + ZL) - (ZE + ZF + ZI + ZJ))

          DX2 = 0.25*((XF + XG + XJ + XK) - (XE + XH + XI + XL))
          DY2 = 0.25*((YF + YG + YJ + YK) - (YE + YH + YI + YL))
          DZ2 = 0.25*((ZF + ZG + ZJ + ZK) - (ZE + ZH + ZI + ZL))

C         The unit vector for the A momentum equation.
C         This is orthogonal to the s momentum vector and the vector
C         which connects the centers of the two b faces.
C         s1(hat) = s2 X s3
          DX1H = DY2*DZ3 - DZ2*DY3
          DY1H = DZ2*DX3 - DX2*DZ3
          DZ1H = DX2*DY3 - DY2*DX3
C         Normalize by its length, and make the sign such that DZ1H > 0.
          SIZ = SIGN( SQRT(DX1H**2 + DY1H**2 + DZ1H**2) , DZ1H )
          DX1H = DX1H/SIZ
          DY1H = DY1H/SIZ
          AMXC(M) = DX1H
          AMYC(M) = DY1H

C         The unit vector for the B momentum equation.
C         This is orthogonal to the s momentum vector and the vector
C         which connects the centers of the two a faces.
C         s2(hat) = s3 X s1
          DX2H = DY3*DZ1 - DZ3*DY1
          DY2H = DZ3*DX1 - DX3*DZ1
          DZ2H = DX3*DY1 - DY3*DX1
C         Normalize by its length, and make the sign such that DZ2H > 0.
          SIZ = SIGN( SQRT(DX2H**2 + DY2H**2 + DZ2H**2) , DZ2H )
          DX2H = DX2H/SIZ
          DY2H = DY2H/SIZ
          BMXC(M) = DX2H
          BMYC(M) = DY2H

        END DO

        RETURN
        END
```