NEWTON SOLUTION OF STEADY TWO-DIMENSIONAL TRANSONIC FLOW

by

MICHAEL BRYCE GILES

S.M., Massachusetts Institute of Technology
(1983)

B.A., Cambridge University
(1981)

Submitted to the Department of Aeronautics
and Astronautics in Partial Fulfillment of
the Requirements of the Degree of

DOCTOR OF PHILOSOPHY IN
AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1985

Signature of Author _____
Dept. of Aeronautics and Astronautics, 30 June 1985

Certified by _____
Professor W.T. Thompkins, Jr. Thesis Supervisor

Certified by _____
Professor Jack L. Kerrebrock Thesis Committee

Certified by _____
Professor Earll M. Murman Thesis Committee

Accepted by _____
Professor Harold Y. Wachman Chairman, Departmental
Graduate Committee

NEWTON SOLUTION OF STEADY TWO-DIMENSIONAL TRANSONIC FLOW

by

Michael B. Giles

Submitted to the Department of Aeronautics and Astronautics on 30 June 1985 in partial fulfillment of the requirements of the Degree of Doctor of Philosophy in Aeronautics and Astronautics

Abstract

A new method is developed for the solution of the steady, two-dimensional Euler equations for transonic flows. The discrete steady-state equations are derived in conservative finite-volume form on an intrinsic streamline grid, and are solved using Newton's method. Direct solution of the linear system of Newton equations is shown to be more efficient than iterative solution. Test cases include duct, cascade, and isolated airfoil flows, and demonstrate the speed and robustness of the method. The accuracy of the solutions is verified by comparison against values obtained analytically, experimentally and by other numerical methods.

Thesis Supervisor: W.T. Thompkins, Jr.
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgements

Many friends and colleagues have helped in this research effort. Firstly, I wish to thank Mark Drela for sharing two years of hard work and fun, with all of the long hours, frustrations and successes, brainstorming and tedious programming. Many thanks also to my thesis supervisor Professor Tilt Thompkins for sharing our enthusiasm and letting us have fun; he knew when to let us charge ahead boldly (or blindly?), and when to be a critical observer, and I greatly appreciate his help during the writing of this thesis. I also wish to thank the other members of my committee, Professors Jack Kerrebrock and Earll Murman, for their helpful discussions, and the other students in the group for their technical comments and for their company during the late nights and the long weekends. Finally, I wish to thank my family who has supported me throughout my studies, and John Dannenhoffer who has helped me so much as a friend and a colleague, with his encouragement, advice and thoughtful criticisms.

This work was supported by the Air Force Office of Scientific Research grant F49620-78C-0084, with technical monitor Dr. James D. Wilson.

Table of Contents

List of Figures

List of Tables

List of Symbols

$\vec{A}_1, \vec{A}_2$	face vectors
$\vec{B}^-, \vec{B}^+$	face vectors
c	speed of sound
h_t	stagnation enthalpy
i,j	discrete variable indices
m_j	mass flow in j^{th} streamtube
m_{tot}	total mass flow
M	Mach number
$\vec{N}$	normal vector
$\hat{n}$	unit vector in direction normal to streamlines
δn	normal movement of grid node
p	pressure (on normal faces)
$\vec{q}$	velocity
$q=\lvert\vec{q}\rvert$	speed
$\hat{s}=\vec{q}/q$	unit vector in streamwise direction
s	entropy
$\vec{S}$	streamwise vector
x,y	coordinates
ρ	density
ρ_t	stagnation density
μ	artificial compressibility factor
Π	pressure (on streamline faces)

1. INTRODUCTION

This thesis presents a new algorithm for solving the steady-state two-dimensional transonic flow through ducts, and over cascades and isolated airfoils. The full flow field is governed by the Navier-Stokes equations, but the assumption is made that the viscous and heat conduction effects are small and can be neglected. Under this approximation the Navier-Stokes equations reduce to the Euler equations which describe inviscid, rotational flow, and in integral form also give the correct Rankine-Hugoniot shock relations. A further approximation which was made by researchers in the past, is the approximation that the flow is isentropic and irrotational, in which case the flow is governed by the potential equation, a scalar equation, which can be solved much more easily and economically. In subsonic flow, the flow is indeed isentropic and irrotational, but in transonic flow shocks produce both entropy and vorticity, and Salas et al [24] have shown that neglecting their effect can lead to serious errors. Hence in this study the potential approximation is not made, and instead the Euler equations are used.

This study is concerned solely with steady state solutions. At present most methods for the numerical calculation of steady state, transonic solutions to the Euler equations are time-marching methods, a finite difference approximation to the unsteady Euler equations. The advantage of this approach is that it is conceptually straightforward and avoids the principal difficulty with the steady state transonic equations. In supersonic regions the steady state equations are hyperbolic with four different characteristics. In subsonic flow two of these characteristics become imaginary, or in other words become a coupled elliptic system. Hyperbolic and elliptic equations in general require different numerical solution methods. Space marching methods are used for hyperbolic equations but cannot be used for elliptic equations, and relaxation methods are used for elliptic equations but cannot be used for hyperbolic equations. The unsteady Euler equations, how-

ever, are hyperbolic in time, both in supersonic and subsonic regions, and so the time marching methods avoid this difficulty. The principal disadvantage of these methods is that the convergence rate to the steady state solution is limited by the relatively slow propagation of pressure waves throughout the flow domain and their reflection at the boundaries of the computational domain. Without the use of acceleration methods several hundred iterations are required. Current methods overcome this problem through a variety of acceleration methods such as using variable time steps, in which the local time step is the maximum possible for numerical stablity, implicit operators, which increase the stability bound and so allow larger time steps, and multigrid, in which several levels of grid coarseness are employed and larger time steps can be used on the coarser grids. Despite all these advances over one hundred iterations are still required in a typical transonic calculation.

The alternative approach is to solve the steady state equations directly using some iterative method which bears no relation to the physical time marching process. The problem, as mentioned before, is the mixed hyperbolic/elliptic nature of the transonic equations. This problem was first overcome for the transonic potential equation by Murman and Cole [21] who introduced a form of numerical viscosity in the supersonic region which allows a relaxation method to be used in the supersonic region. When formulated conservatively as an upwinded density, this allows the capture of shocks without any loss of mass flux across the shock, although momentum is not conserved due to the limitations of the potential approximation that the flow is isentropic and irrotational. One way of interpreting the relaxation procedure is that it is an application of Newton's method (also called the Newton-Raphson method) to a system of nonlinear equations. The unknowns are the values of the potential at a set of points. The nonlinear equations are the discrete mass equations, which state that the total mass flux into and out of each computational cell is zero. Newton's method is to linearize the nonlinear equations about the current approximate solution, and then

solve the linearized system to obtain a better approximate solution. The linear Newton equations are usually solved using SLOR (Successive Line Over-Relaxation) accelerated by multigrid.

This same approach has been applied to the Euler and Navier-Stokes equations by Childs and Pulliam [7], and Jespersen [18]. Childs and Pulliam solve the linear Newton equations using the factored implicit algorithm of Beam and Warming [4] accelerated by multigrid, and Jespersen uses Gauss-Seidel with multigrid. Their conclusions were mixed; the Newton procedure worked, but did not have any advantages, either in accuracy, capabilities or speed, over traditional time-marching methods, which are much simpler to program. An intermediate approach, lying between timemarching and full Newton methods, is the method of Mulder and Van Leer [20], in which a Backward Euler time integration scheme is used to solve the unsteady Euler equations. For small values of Δt, the time step, this behaves like a time-marching method. In the limit $\Delta t \to \infty$, it becomes Newton's method. Again the implicit system of equations is solved using SOR and SLOR with a multigrid accelerator. In applications they use a small Δt initially while there are large changes in the flow field and the shock position is not established, and then increase Δt to obtain a rapid final convergence to the steady state solution. Their results demonstrate that good computational efficiency can be achieved. The approach in this thesis, developed independently from Childs and Pulliam, and Jespersen, also uses Newton's method. Both direct and iterative methods for solving the Newton equations are developed and it is found that the direct method is more efficient than the iterative method for grids of reasonable size. This is in contrast to the work of Childs and Pulliam, Jespersen and Van Leer and Mulder for whom a direct solution would be much more expensive, because their formulation of the discrete equations has four variables per computational cell compared to the two variables per cell required by the present method due to its unique formulation of the discrete steady-state Euler equations.

Most finite difference discretizations of the steady state Euler equations use conservative fluxes on a fixed grid of computational cells. The grid points and computational cells have fixed positions set initially by the user, and the discrete mass, momentum and energy equations are a discrete approximation to the integral form of the Euler equations applied to each computational cell. The reason this approach is called conservative is that the flux of mass, for example, out of one cell is exactly equal to the flux of mass into the neighboring cell, and so on a global view all of the internal flux cancel, and hence the mass flux across the inlet boundary is exactly equal to the mass flux across the outlet boundary. Thus mass is "conserved"; there is no "production" of mass inside the domain. The advantage of the conservative approach is that it guarantees the correct treatment of shocks. In exactly the same way that the Rankine-Hugoniot shock jump relations can be obtained from the Euler equations written in integral form [19], the conservative form guarantees that the flux of mass, momentum and stagnation enthalpy on either side of the shock are the same and so the flow on the two sides of the shock must satisfy the Rankine-Hugoniot relations. An alternative discretization of the steady state Euler equations is that used by streamline curvature methods [22]. In this case one set of grid lines corresponds to streamlines and so the grid is not fixed but determined as part of the solution. Instead of using a conservative flux formulation the finite difference equations are a discrete approximation to the normal and tangential momentum equations in differential form, together with the conditions that the mass flux amd stagnation enthalpy are constant along each streamtube. These equations are solved by a relaxation procedure. The streamline curvature method remans popular in industry for calculating subsonic flows in turbomachinery, but when applied to transonic flow cases two problems arise. The first is that the relaxation method may not be stable in the supersonic region because of the change in the type of the steady state equations. The second is that even if the steady state solution is obtained, it may have large errors because the non-conservative formulation means that shocks are

not correctly calculated.

The discretization used in this thesis combines the conservative formulation of finite volume schemes with the intrinsic streamline grid of streamline curvature methods. The discrete steady-state equations are an approximation to the integral form of the Euler equations, applied to quadrilateral cells which are defined such that there is no mass flux across two of the four sides. The only contribution to the steady-state equations from the streamline faces comes from the pressure contribution to the momentum equations. The mass and energy equations for each cell are particularly simple since mass flux and stagnation enthalpy are conserved along each streamtube. Since one set of grid lines is defined to be streamlines, the grid is not known a priori, but must be determined as part of the solution. Although it might appear that this increases the number of unknown variables in the problem, in fact the linear Newton equations can be manipulated to reduce to just two equations and two unknowns per computational cell, which is fewer than for normal finite volume formulations and so is much more efficiently solved by the direct solution method used for the Newton equations. An additional, very important, advantage of this formulation is that it is as simple to specify the pressure on the surface of an airfoil and determine the shape of the airfoil, as it is to specify the position of the shape of the airfoil and determine the pressure distribution. The former problem is called the inverse problem and its solution for transonic flow is extremely difficult, but this method with the streamline grid determined as part of the solution and the robust Newton procedure is ideally suited for it. The application of this method to the inverse problem is not discussed in this thesis, but is presented in the Ph.D. thesis of a colleague, Mark Drela [11]. His thesis also presents a method for incorporating a coupled boundary layer analysis, using an integral boundary layer method. The power and simplicity of the Newton solution procedure is demonstrated by the fact that the boundary layer equations and the coupling relations between the boundary layer and the

outer inviscid flow can simply be treated as additional equations and included in the Newton iterative procedure.

The historical roots of the steady-state discrete equations lie in previous work by Wornom [28-30] and Giles [13]. In reference [28] Wornom solved the steady Euler equation for quasi-one-dimensional flow using a box formulation. One particularly interesting feature of this work is that for subsonic flow no artificial dissipation of any sort is needed, and only physical boundary conditions are required, as opposed to most time-marching schemes in which some form of extrapolation is required at boundaries in addition to the physical boundary conditions. In the case of transonic flow Wornom introduces artificial compressibility in order to maintain a well-posed problem. This involves replacing the density in the mass equation by a weighted average of the densities at a given node and the its upstream neighbor, and was first used by Eberle [12], and Hafez et al. [16] for the full potential equation. Wornom then extended his method to two-dimensional supersonic flow on a fixed grid [29], using special shock and sonic point operators instead of artificial compressibility. Independently, Giles [13] used a box method to solve the unsteady Euler equations for quasi-one-dimensional transonic flow. Special conservative treatment of the sonic and shock cells, incorporating shock fitting, was used to obtain a method which had no artificial dissipation, and no non-physical boundary conditions. For subsonic steady flow the discrete equations were identical to those used by Wornom [28]. One conclusion of this work was that the use of artificial compressiblity, instead of the special treatment of shocks and sonic cells, was preferable for reasons of robustness and simplicity.

The work which is presented in this thesis represents the natural extension of the box method to two-dimensional flow. In particular the discrete equations for a streamtube with a straight centerline in the two-dimensional case reduce exactly to the equations for the quasi-one-

dimensional case. Some intermediate work and preliminary results of the current research were presented in three research papers. The first paper [9] contains a solution method for supersonic flow, with applications to both a duct flow for verification purposes, and a free supersonic jet problem to demonstrate the inverse capability of specifying the pressure on the surface streamline instead of the position of the surface streamline. The solution is obtained using Newton's method applied in a space-marching way, very similar to the Keller Box method for solving finite difference boundary layer equations [6]. The paper also presents preliminary results for subsonic and transonic flow, solved by a line-relaxation method instead of the Newton method. The second paper [10] introduces the Newton solution procedure for transonic flow and the incorporation of the coupled integral boundary layer analysis. The final paper [14] presents the solution of the inverse problem for transonic flow, again using the Newton solution procedure.

Chapter 2 of this thesis derives the discrete steady-state Euler equations and the corresponding boundary conditions. Chapter 3 discusses the introduction of artificial compressibility into the mass equation in supersonic regions. An analysis of first order artificial compressibility shows that there is a minimum amount required for the problem to be well-posed, and that twice this level produces sharp shocks. Also a second order accurate correction is defined, and numerical test cases demonstrate it produces smaller stagnation density errors. Chapter 4 introduces the Newton procedure and shows the manner in which the discrete nonlinear equations are linearized, including both the Euler equations and the boundary conditions. Chapters 5, 6 and 7 discuss different ways of solving the linear set of Newton equations. Chapter 5 presents a direct solution method which uses a modified block-tridiagonal algorithm. Chapter 6 shows the modifications to both the boundary conditions and the solution procedure required for cases in which the flow is choked. Chapter 7 discusses the relative advantages of direct and iterative solution methods, and then presents an iterative

method for solving the Newton equations for subsonic flow. This uses a preconditioning which effectively decouples the convective entropy equation from the elliptic pressure equation. Chapter 8 introduces the concept of global variables and constraints which are important in giving the overall method a great amount of flexibility. The global variables can be variables such as inlet and outlet flow angle, for casades, or angle of attack and circulation, for isolated airfoils. The corresponding global constraints can be specified lift and a Kutta condition, which specifies that there is no jump in pressure across the trailing edge of the airfoil. The global variables could also represent a change in the pitch, for a cascade, or a change in the freestream Mach number, for an airfoil, allowing one to examine the linear sensitivity of the solution to different global parameters. To obtain the same information with a time-marching method would require a series of calculations with slightly different global parameters, which would be much more expensive. Chapter 9 presents a number of test cases to demonstrate the robustness of the Newton method and the accuracy of the solutions obtained. The calculated results are compared to values obtained theoretically, experimentally, or by other numerical methods. Chapter 10 discusses the results and draws some final conclusions.

2. STEADY STATE EQUATIONS

In this chapter the discrete steady state Euler equations are derived, together with the boundary conditions required for a two-dimensional duct problem. The first section shows how the discrete Euler equations for a single computational cell are derived from the integral form of the Euler equations. The second section discusses the inlet, outlet and solid wall boundary conditions necessary for a duct problem. The third section shows that an additional relation is required to achieve consistency, and to match the number of equations and unknown variables. Finally the fourth section discusses different approaches to solving this set of steady-state equations.

2.1 Euler Equations

The starting point for the derivation of the discrete Euler equations is the integral form of the steady state, two dimensional Euler equations. For a closed curve **C** with outward normal $\vec{n}$ the integral equations are [19]

Mass equation
$$\oint_{\mathbf{C}} \rho\vec{q}\cdot\vec{n}\, ds = 0 \tag{2.1}$$

Momentum equation
$$\oint_{\mathbf{C}} \rho(\vec{q}\cdot\vec{n})\vec{q} + p\vec{n}\, ds = 0 \tag{2.2}$$

Energy equation
$$\oint_{\mathbf{C}} \rho\vec{q}\cdot\vec{n}\, h_t\, ds = 0 \tag{2.3}$$

The discrete finite-volume Euler equations are a discrete approximation to these equations, in which the curve **C** is the boundary of an area usually referred to as a conservation cell. This approach is standard in computational fluid dynamics, but a unique feature of this implementation is that the cells are defined such that one pair of opposing faces are streamlines of the flow and so there is no mass flux

across them. Hence the only contribution to the above integrals from these two faces is the pressure term in equation (2.2). This means that the density and velocity need only be defined on the other two faces, and also the direction of the velocity must somehow be related to the local geometry to be consistent with the statement that two of the faces are streamlines. A further consequence is that unlike most numerical methods the grid geometry is not known a priori and must be determined as part of the solution.

A typical conservation cell is shown in Figure 2.1. The geometry variables (x,y) are located at the grid nodes marked X. The nodes marked as · are defined to be at the midpoints of the lines connecting the grid nodes. The upper and lower bent faces of the cell are the streamline faces across which there is no mass flux. Figure 2.1 shows the node numbering convention used when discussing the discrete Euler equations for a particular cell, and is the same as that used in the program.

Four vectors which need to be defined are the vectors along the faces of the cell, which are illustrated in Figure 2.2. For the bent streamline faces the vectors are the vector sum of the two parts.

$$A_{x1} = \tfrac{1}{2}(x_1^+ + x_2^+) - \tfrac{1}{2}(x_1^- + x_2^-) \quad , \quad A_{y1} = \tfrac{1}{2}(y_1^+ + y_2^+) - \tfrac{1}{2}(y_1^- + y_2^-) \qquad (2.4a,b)$$

$$A_{x2} = \tfrac{1}{2}(x_2^+ + x_3^+) - \tfrac{1}{2}(x_2^- + x_3^-) \quad , \quad A_{y2} = \tfrac{1}{2}(y_2^+ + y_3^+) - \tfrac{1}{2}(y_2^- + y_3^-) \qquad (2.4c,d)$$

$$B_x^- = [\tfrac{1}{2}(x_3^- + x_2^-) - x_2^-] + [x_2^- - \tfrac{1}{2}(x_2^- + x_1^-)] = \tfrac{1}{2}(x_3^- - x_1^-) \qquad (2.4e)$$

$$B_y^- = [\tfrac{1}{2}(y_3^- + y_2^-) - y_2^-] + [y_2^- - \tfrac{1}{2}(y_2^- + y_1^-)] = \tfrac{1}{2}(y_3^- - y_1^-) \qquad (2.4f)$$

$$B_x^+ = [\tfrac{1}{2}(x_3^+ + x_2^+) - x_2^+] + [x_2^+ - \tfrac{1}{2}(x_2^+ + x_1^+)] = \tfrac{1}{2}(x_3^+ - x_1^+) \qquad (2.4g)$$

$$B_y^+ = [\tfrac{1}{2}(y_3^+ + y_2^+) - y_2^+] + [y_2^+ - \tfrac{1}{2}(y_2^+ + y_1^+)] = \tfrac{1}{2}(y_3^+ - y_1^+) \qquad (2.4h)$$

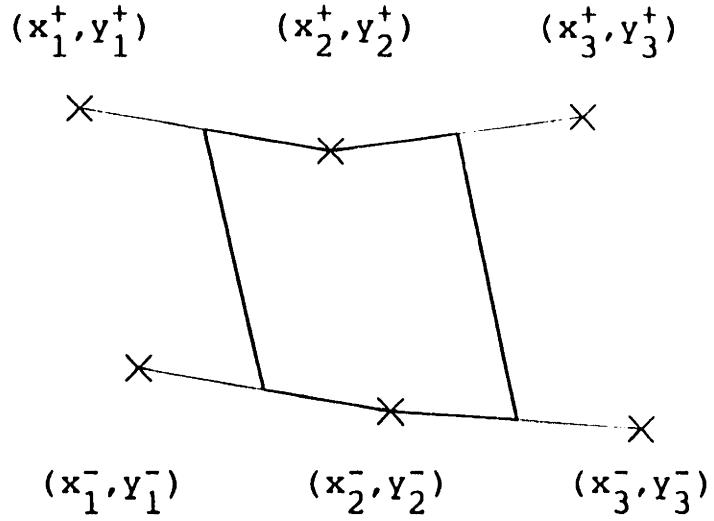


Figure 2.1: Location of grid nodes and cell geometry.

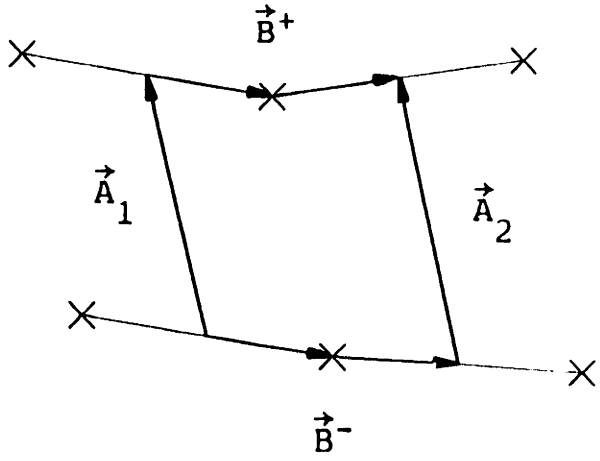


Figure 2.2: Definition of vectors $\vec{A}_1$, $\vec{A}_2$, $\vec{B}^-$, $\vec{B}^+$.

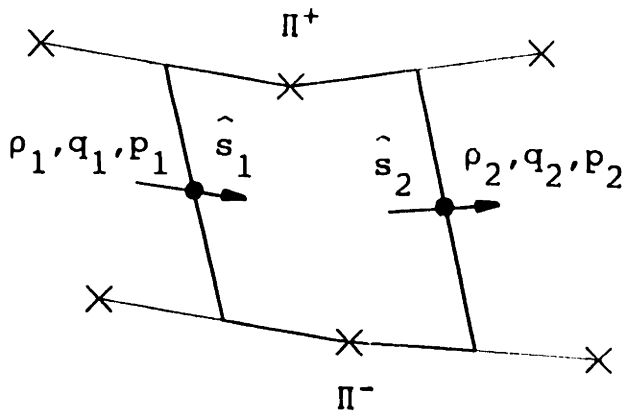


Figure 2.3: Location of flow variables.

An additional two vectors which it is convenient to define are $\vec{S}$, which is the average of $\vec{B}^-$ and $\vec{B}^+$, and $\vec{N}$, which is the average of $\vec{A}_1$ and $\vec{A}_2$.

$$S_x = \frac{1}{2}(B_x^- + B_x^+) \quad , \quad S_y = \frac{1}{2}(B_y^- + B_y^+) \tag{2.5a,b}$$

$$N_x = \frac{1}{2}(A_{x1} + A_{x2}) \quad , \quad N_y = \frac{1}{2}(A_{y1} + A_{y2}) \tag{2.5c,d}$$

Since the nodes $(x_1^+,y_1^+),(x_2^+,y_2^+),(x_3^+,y_3^+)$ and $(x_1^-,y_1^-),(x_2^-,y_2^-),(x_3^-,y_3^-)$ are defined to be streamlines, the local velocity direction must be related to them. This is accomplished by defining the unit flow vector $\hat{s}_1$ to be tangent to the line joining the average of nodes 1^+ and 1^- to the average of nodes 2^+ and 2^-. The unit flow vector $\hat{s}_2$ is defined similarly. Both are shown in Figure 2.3, and their definitions are given below.

$$s_{x1} = (x_2^- + x_2^+)/2 - (x_1^- + x_1^+)/2 \quad , \quad s_{y1} = (y_2^- + y_2^+)/2 - (y_1^- + y_1^+)/2 \tag{2.6a,b}$$

$$s_1 = (s_{x1}^2 + s_{y1}^2)^{1/2} \quad , \quad \hat{s}_{x1} = s_{x1}/s_1 \quad , \quad \hat{s}_{y1} = s_{y1}/s_1 \tag{2.6c-e}$$

$$s_{x2} = (x_3^- + x_3^+)/2 - (x_2^- + x_2^+)/2 \quad , \quad s_{y2} = (y_3^- + y_3^+)/2 - (y_2^- + y_2^+)/2 \tag{2.7a,b}$$

$$s_2 = (s_{x2}^2 + s_{y2}^2)^{1/2} \quad , \quad \hat{s}_{x2} = s_{x2}/s_2 \quad , \quad \hat{s}_{y2} = s_{y2}/s_2 \tag{2.7c-e}$$

The velocity at the midpoint of the left face of the conservation cell is thus defined to have direction $\hat{s}_1$ and magnitude q_1 so that $\vec{q}_1 = q_1\hat{s}_1$. The final geometric quantities to be defined are the normal areas A_{n1} and A_{n2}, which are the vector dot products of the unit velocity vectors and the normal area vectors of the faces. Since $\vec{A}_1$ and $\vec{A}_2$ are defined along the faces, not normal to them, the actual definitions of A_{n1} and A_{n2} are

$$A_{n1} = s_{x1}A_{y1} - s_{y1}A_{x1} \equiv |\hat{s}_1 \times \vec{A}_1| \quad , \quad A_{n2} = s_{x2}A_{y2} - s_{y2}A_{x2} \equiv |\hat{s}_2 \times \vec{A}_2| \tag{2.8a,b}$$

where the operator | | is defined to mean taking the scalar component in the third (out of plane) dimension, and will only be applied to vectors having only a component in the third dimension. It is important to note that the scalar value given by | | may be positive or negative, depending on the direction of the vector. The operator | | does <u>not</u> return the absolute value and so is an unconventional operator, but one that is very convenient in this application.

The flow variables ρ, density, q, speed, and p, pressure are located at the midpoints of the faces which are not streamlines, as shown in Figure 2.3, and another pressure variable, denoted differently by Π for clarity, is located on the streamline faces where, as previously noted, no other flow variables are required.

The discrete mass equation is simply a statement that the mass flux along a streamtube is a constant.

$$m = \rho_1 q_1 A_{n1} = \rho_2 q_2 A_{n2} \tag{2.9}$$

With due regard to the directions of the vectors $\vec{A}_1, \vec{A}_2, \vec{B}^-$ and $\vec{B}^+$, the discrete approximations to the x and y-components of the integral form of the momentum equation (2.2) are,

$$\begin{aligned} &\rho_1 q_1^2 A_{n1} s_{x1} - \rho_2 q_2^2 A_{n2} s_{x2} + p_1 A_{y1} - p_2 A_{y2} + \Pi^+ B_y^+ - \Pi^- B_y^- \\ &= \rho_1 q_1^2 A_{n1} s_{x1} - \rho_2 q_2^2 A_{n2} s_{x2} + (p_1 - p_2) N_y + (\Pi^+ - \Pi^-) S_y + \tfrac{1}{2}(\Pi^+ + \Pi^- - p_1 - p_2)(B_y^+ - B_y^-) \\ &= 0 \end{aligned} \tag{2.10}$$

$$\begin{aligned} &\rho_1 q_1^2 A_{n1} s_{y1} - \rho_2 q_2^2 A_{n2} s_{y2} - p_1 A_{x1} + p_2 A_{x2} - \Pi^+ B_x^+ + \Pi^- B_x^- \\ &= \rho_1 q_1^2 A_{n1} s_{y1} - \rho_2 q_2^2 A_{n2} s_{y2} - (p_1 - p_2) N_x - (\Pi^+ - \Pi^-) S_x - \tfrac{1}{2}(\Pi^+ + \Pi^- - p_1 - p_2)(B_x^+ - B_x^-) \\ &= 0 \end{aligned} \tag{2.11}$$

The step from the first line to the second line in these two equations is achieved using the definitions of $\vec{S}$ and $\vec{N}$, and the identity $\vec{B}^+ - \vec{B}^- = \vec{A}_2 - \vec{A}_1$.

The energy equation reduces to the statement that the stagnation enthalpy is constant along a streamtube, although of course the value of the stagnation enthalpy, like the value of the mass flux, may vary from one streamtube to another.

$$h_t = \frac{\gamma}{\gamma-1}\frac{p_1}{\rho_1} + \frac{1}{2}q_1^2 = \frac{\gamma}{\gamma-1}\frac{p_2}{\rho_2} + \frac{1}{2}q_2^2 \qquad (2.12)$$

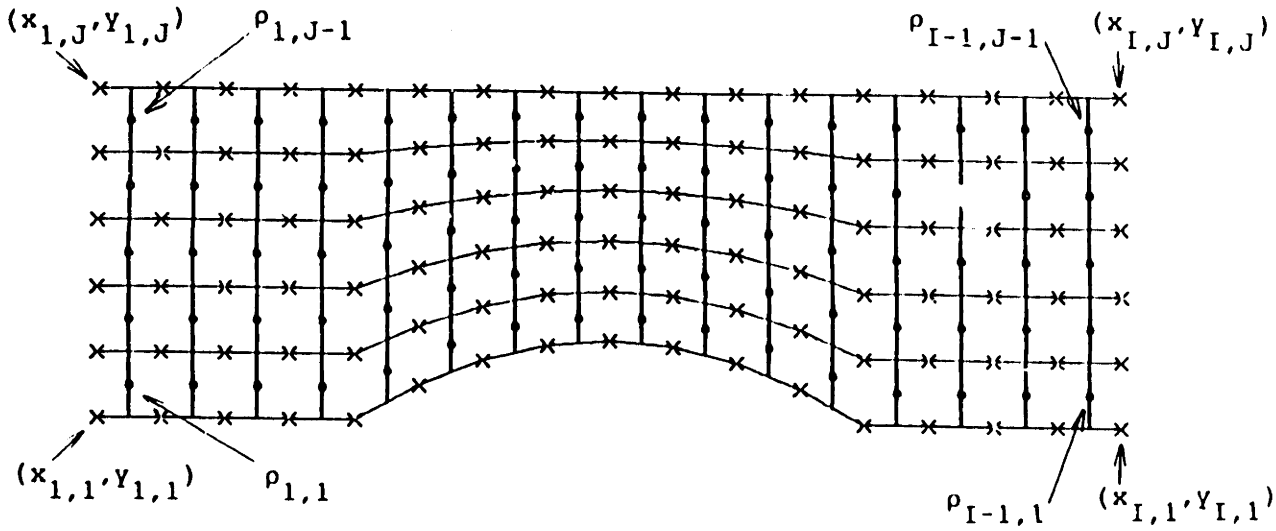


Figure 2.4: Indexing for duct geometry.

2.2 Duct boundary conditions

This section presents the boundary conditions necessary for the solution of a two-dimensional duct problem. As illustrated in Figure 2.4, there are I grid nodes in the streamwise direction, and J grid nodes in the normal direction. Thus there are J-1 streamtubes, and each streamtube has I-2 conservation cells.

The solid wall boundary conditions are very simple. The position of the grid nodes is specified. No other boundary conditions are required, in contrast to almost all other Euler methods which require pressure extrapolation, or other special treatment, at solid walls. In addition it is just as simple to specify either a displacement thickness which allows the calculation of coupled viscous-inviscid flows, or the wall pressure which allows the solution of the inverse flow problem in which one determines the geometry which produces a specified desired pressure distribution. These two developments are the subject of the Mark Drela's Ph.D. thesis [11].

There are three flow quantities specified at the inlet. The first two are the mass flux and stagnation enthalpy of each streamtube. These remain constant along each streamtube, and are treated as such in the discrete Euler equations in the last chapter. The third quantity is the inlet stagnation density, which is defined as,

$$\rho_t = \rho_1 \left(1 + \frac{\gamma-1}{2} M_1^2\right)^{1/(\gamma-1)} = \rho_1 \left(1 - q_1^2/2h_t\right)^{-1/(\gamma-1)} \qquad (2.13)$$

where the subscript 1 denotes the variables at the inlet face.

An important thing to note with this set of inlet boundary conditions is that it is only well-posed if the flow is not choked. If the flow is choked the total mass flow is determined uniquely by the inlet stagnation enthalpy and density and duct geometry. For example, the choked mass flow for a quasi-1-D converging/diverging duct is given by

$$m = \left(\frac{2}{\gamma+1}\right)^{1/(\gamma-1)} \left(\frac{2(\gamma-1)}{\gamma+1}\right)^{1/2} \rho_t (h_t)^{1/2} A^* \qquad (2.14)$$

where A^* is the area of the sonic throat. The correct boundary conditions for choked flow are discussed in Chapter 6, along with the necessary modifications to the solution procedure. The reason they are not discussed earlier is that the solution procedure becomes significantly harder to understand, and also is approximately four times as expensive computationally. Thus it is preferable to use the above unchoked boundary conditions wherever possible.

The discrete Euler equations also require boundary conditions for the position of the grid nodes at the inlet and outlet planes. In all the cases to be presented in this thesis the inlet flow is uniform and so the inlet nodes are spaced according to the fractional mass flow in each streamtube. Thus,

$$y_{1,j+1} - y_{1,j} = (y_{1,J} - y_{1,1})\, m_j / m_{total} \qquad (2.15)$$

Since the inlet area, $y_{1,J} - y_{1,1}$, and the streamtube mass fluxes are all specified, this implies for the duct problem that the inlet node positions are fixed.

At the outlet the same approach cannot be used because the flow is no longer uniform due to entropy production at shocks. Instead the boundary condition is that the streamtube area at the I^{th} streamwise station is the same as the area at the $I\text{-}1^{th}$ station.

$$(y_{I,j+1} - y_{I,j}) - (y_{I-1,j+1} - y_{I-1,j}) = 0 \qquad (2.16)$$

2.3 Auxiliary pressure relation

At this point it is instructive to count the number of variables and the number of equations. Including all of the boundary nodes, there are IJ grid nodes. Each grid node has both x and y variables, but for the duct problem the nodes are constrained to move along lines x=constant so there is really only 1 unknown per grid node, giving a total of IJ grid point variables. In more general geometries such as cascades and airfoils the grid nodes are constrained to move in a direction approximately normal to the local streamline. In addition there are three unknown variables, p,q,ρ, at each of the (I-1)(J-1) normal streamtube faces, and one unknown variable, Π, at each of the (I-2)J streamline faces, giving a total of 5IJ-3I-5J+3 unknown variables.

Counting the equations now, there are two momentum equations for each of the (I-2)(J-1) conservation cells, and there are two equations, the mass equation and the stagnation enthalpy equation, at each of the (I-1)(J-1) normal streamtube faces. In addition there are 2I solid wall boundary conditions, J-1 inlet stagnation density conditions, and 2(J-2) inlet and outlet grid node equations, giving a total of 4IJ-2I-3J+1 equations. Thus IJ-I-2J+2 = (I-2)(J-1) additional equations are required to make the numer of equations equal to the number of unknown variables. This is exactly one additional equation per computational cell.

The origin of the additional equation comes from the realization that as yet nothing constrains the average value of Π. For two-dimensional uniform flow in a constant area duct the discrete Euler equations are perfectly satisfied by a solution in which the p variables are equal to one constant value and the Π are equal to a different constant value. This is because the momentum equations essentially are concerned with pressure differences. The x-momentum equation for

uniform ρ,q gives $p_1-p_2=0$ and the y-momentum equation gives $\Pi^- - \Pi^+ = 0$. The average p value is constrained through the mass and stagnation enthalpy equations, but there is nothing which constrains the average Π value, and in this example it can take any value. For consistency the average local value of Π must be approximately equal to the average local value of p with the equality becoming exact in the limit that $\Delta x, \Delta y \to 0$. Thus, bearing in mind that exactly one equation for each computational cell is required to match the number of unknowns and equations, this is achieved most simply by the following equation, which will be referred to as the auxiliary pressure relation.

$$\Pi^- + \Pi^+ = p_1 + p_2 \tag{2.17}$$

A more general form for the auxiliary pressure relation is,

$$\Pi^- + \Pi^+ = p_1 + p_2 + 2P_c \tag{2.18}$$

where P_c is a function which approaches zero in the limit $\Delta x, \Delta y \to 0$. The reason this more general form is sometimes required is that the discrete equations including (2.17) are satisfied by a solution which has uniform density, velocity and pressure, and a grid which has a "sawtooth" oscillation in both the streamwise and normal directions, as shown in Figure 2.5. Usually this does not appear in a direct solution because it is inhibited by the far-field and solid-body boundaries, but it sometimes can be seen in regions in which there is a strong local streamline curvature, and also it causes problems in the iterative solution procedure to be given in Chapter 7. Considering the particular conservation cell shown in Figure 2.6, the physical reason why the "sawtooth" solution should not be valid is that the cross-sectional area A_c at the middle of the cell is greater than the average of the areas A_1 and A_2 at the two ends, and for subsonic flow this means that the average of Π^+ and Π^- should be greater than the average of p_1 and p_2 by an amount P_c which

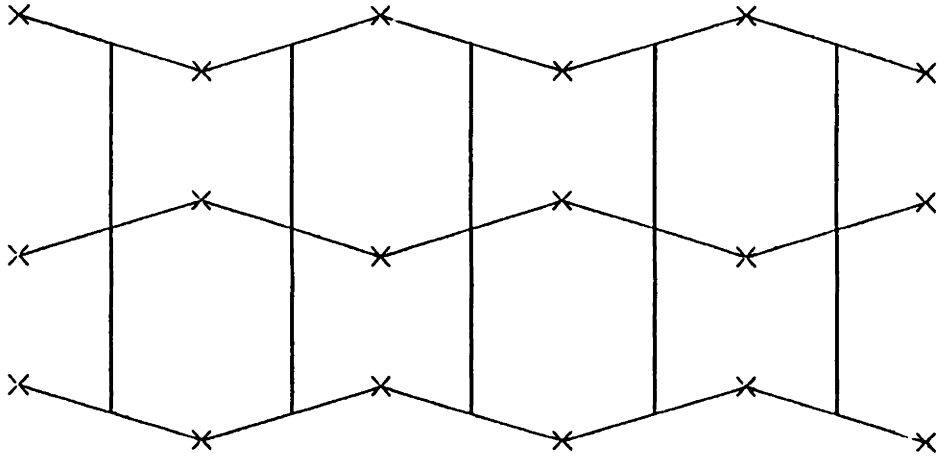


Figure 2.5: Unconstrained "sawtooth" mode.

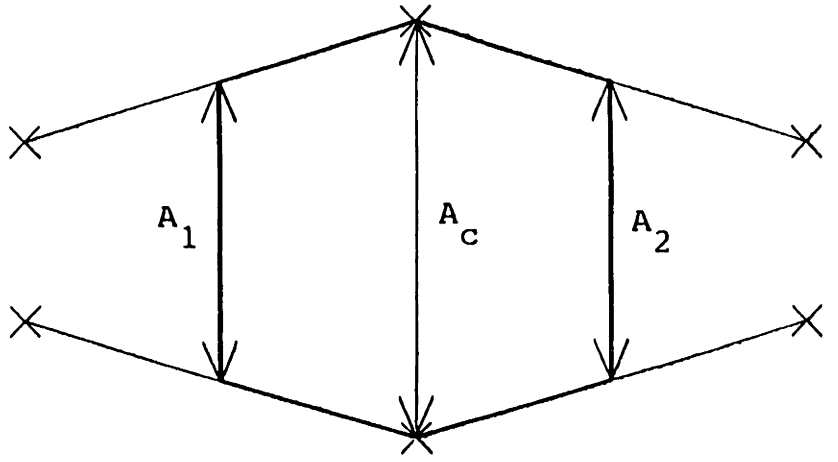


Figure 2.6: Cross-sectional areas for pressure correction.

can be determined by assuming an isentropic expansion. By the same reasoning the Π values on the neighboring streamtube need to be reduced due to the pinching of that conservation cell, and it is the resultant mismatch in the Π values on either side of the steamline which prevents the "sawtooth" solution from being valid.

This discussion suggests that P_c should be defined by

$$p_c = \frac{\partial p}{\partial A}_{s,h_t=const} (A_c - \frac{1}{2}(A_1+A_2)) = \frac{p}{A} \gamma M^2/(1-M^2) \ (A_c - \frac{1}{2}(A_1+A_2)) \qquad (2.19)$$

This definition, however, has $P_c \to \infty$ as $M \to 1$ which is undesirable. In practice it was found that no pressure correction was needed in supersonic regions, so it was decided to bring P_c smoothly to zero at M=1. Also the area terms are replaced by an expression based on streamline segments which is equivalent for an approximately uniform grid. The third modification is that an arbitrary multiplicative constant k is introduced. It was found in numerical test cases that a value for k of 0.05-0.2 was sufficient to prevent the sawtooth mode from appearing. Thus the final chosen form for P_c is

$$P_c = \begin{cases} k \ p \ \gamma M^2(1-M^2) \ \dfrac{|\vec{s}_1^- x \vec{s}_2^-| \ - \ |\vec{s}_1^+ x \vec{s}_2^+|}{2 \ |\vec{S} x \vec{N}|} & , \quad M^2<1 \\ 0 & , \quad M^2>1 \end{cases} \qquad (2.20)$$

where

$$p = \frac{1}{2}(p_1+p_2) \quad , \quad M^2 = \frac{1}{2}(M_1^2+M_2^2) \qquad (2.21a,b)$$

and

$$s_{1x}^{\pm} = x_2^{\pm}-x_1^{\pm} \ , \ s_{1y}^{\pm} = y_2^{\pm}-y_1^{\pm} \quad , \quad s_{2x}^{\pm} = x_3^{\pm}-x_2^{\pm} \ , \ s_{2y}^{\pm} = y_3^{\pm}-y_2^{\pm} \qquad (2.22a-d)$$

For smooth flows with smooth grids $P_c=O(\Delta x^2,\Delta y^2)$, which satisfies the condition that $P_c \to 0$ as $\Delta x, \Delta y \to 0$ and preserves global second order accuracy.

2.4 Possible solution methods

This chapter has presented the steady state equations obtained from the particular choice of discretization of the Euler equations, and their associated boundary conditions. The next question is how to solve this system of nonlinear equations. There are two general classes of iterative solution methods for a system of nonlinear equations.

The first class uses Newton's method applied to the entire nonlinear system of equations. Each one of the nonlinear equations is linearized about the current approximate solution to obtain a linear system of N equations in N unknowns, the corrections needed to produce a better approximate solution. N here is the total number of equations, and variables, which is very large, so the question then is how to solve this system of equations. Jespersen [18], who has tried this general approach for the Euler equations with a different form of discretization, considered the system to be too large to be solved efficiently by direct methods, and so uses a method which he labels "Newton-Multigrid," in which he solves the linear Newton equations by a iterative Gauss-Seidel method accelerated using multigrid, a technique first developed by Brandt [5] for solving elliptic equations.

The second class of methods iterates directly on the nonlinear equations. At each step of each iteration, all of the variables are held fixed except for those at a particular grid node (point Gauss-Seidel), or a particular line (line Gauss-Seidel). The nonlinear equations which correspond to this point or line are linearized to obtain a relatively small system of linear equations, which are solved to get the corrections for those variables. This iterative procedure converges slowly usually, so Jespersen [18] accelerates it using the nonlinear full multigrid algorithm developed by Brandt [5]. Jespersen labels this method "Multigrid-Newton" since it uses the nonlinear multigrid algorithm with a Newton method at each node as the local relaxation method.

In this thesis the first approach is followed for several reasons. Firstly, the Newton method is conceptually very straightforward. Provided the initial trial solution is not "too far" from the true solution, the method always converges and asymptotically the convergence is quadratic. Secondly, the Newton method allows the introduction of global variables and global constraints. As will be discussed in Chapter 8, these are special variables, such as circulation, and special equations, such as the Kutta condition at the trailing edge, which in some sense have a global span or influence. In the Newton method they are treated simply as additional variables and equations, although in the programming implementation they are handled separately. It is not clear how they would be included under the second class of methods.

3. ARTIFICIAL COMPRESSIBILITY

3.1 Introductory discussion

The concept of artificial compressibility was first introduced by Eberle [12], Harten [17] and Hafez et al. [16] for the solution of the full potential equation in transonic regimes. Following the normalization adopted by Hafez, the full potential equation for steady two-dimensional flow is

$$\frac{\partial}{\partial x}(\rho\frac{\partial \phi}{\partial x}) + \frac{\partial}{\partial y}(\rho\frac{\partial \phi}{\partial y}) = 0 \tag{3.1}$$

where, due to the isentropic assumption, ρ is given by

$$\rho = (M_\infty^2 c^2)^{1/(\gamma-1)} = [1 - \frac{\gamma-1}{2} M_\infty^2(\frac{\partial \phi}{\partial x}^2 + \frac{\partial \phi}{\partial y}^2 - 1)]^{1/(\gamma-1)} \tag{3.2}$$

The conservative discretization of (3.1) on a cartesian grid, as shown in Figure 3.1, is

$$[\rho_{i+\frac{1}{2},j}(\phi_{i+1,j} - \phi_{i,j}) - \rho_{i-\frac{1}{2},j}(\phi_{i,j} - \phi_{i-1,j})]/\Delta x^2$$

$$+ [\rho_{i,j+\frac{1}{2}}(\phi_{i,j+1} - \phi_{i,j}) - \rho_{i,j-\frac{1}{2}}(\phi_{i,j} - \phi_{i,j-1})]/\Delta y^2 = 0 \tag{3.3}$$

where

$$\rho_{i+\frac{1}{2},j} = \frac{1}{2}(\rho_{i+1,j} + \rho_{i,j}) \quad , \text{ etc.} \tag{3.4}$$

$$\rho_{i,j} = (M_\infty^2 c_{i,j}^2)^{1/(\gamma-1)} = [1 - \frac{\gamma-1}{2} M_\infty^2(q_{i,j}^2 - 1)]^{1/(\gamma-1)} \tag{3.5}$$

$$q_{i,j}^2 = [(\phi_{i+1,j} - \phi_{i-1,j})/2\Delta x]^2 + [(\phi_{i,j+1} - \phi_{i,j-1})/2\Delta y]^2 \tag{3.6}$$

This set of equations is satisfactory for the subsonic regions of the flow field which are elliptic, but in the supersonic regions of the flow field, which are hyperbolic, equation (3.3) needs to be modified.

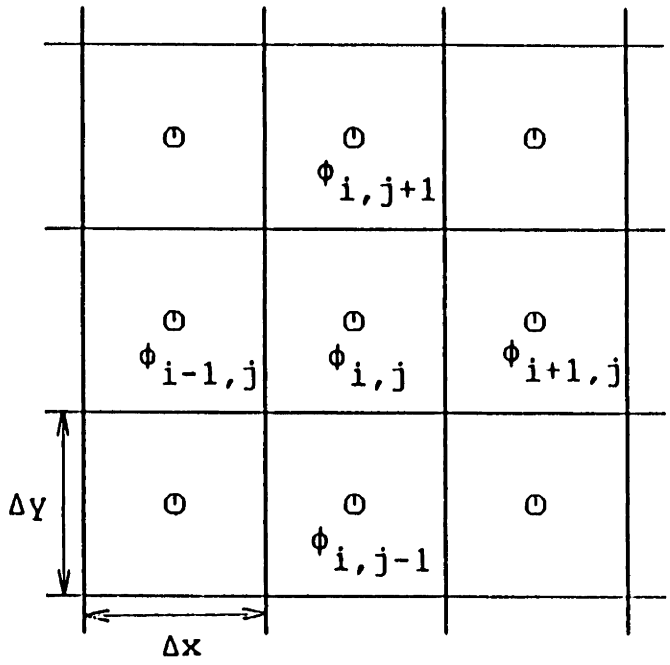


Figure 3.1: Location of grid nodes and variables for Potential equation.

As described in [16], the modified equation is

$$[\tilde{\rho}_{i+\frac{1}{2},j}(\Phi_{i+1,j}-\Phi_{i,j}) - \tilde{\rho}_{i-\frac{1}{2},j}(\Phi_{i,j}-\Phi_{i-1,j})]/\Delta x^2$$

$$+ [\tilde{\rho}_{i,j+\frac{1}{2}}(\Phi_{i,j+1}-\Phi_{i,j}) - \tilde{\rho}_{i,j-\frac{1}{2}}(\Phi_{i,j}-\Phi_{i,j-1})]/\Delta y^2 = 0 \qquad (3.7)$$

where

$$\tilde{\rho}_{i+\frac{1}{2},j} = \frac{1}{2}(\tilde{\rho}_{i+1,j}+\tilde{\rho}_{i,j}) \quad , \text{ etc.} \qquad (3.8)$$

and

$$\tilde{\rho}_{i,j} = \rho_{i,j} - \mu_{i,j}\Delta s \frac{\partial \rho}{\partial s} \qquad (3.9)$$

$\frac{\partial \rho}{\partial s}$ is an approximation to the streamwise derivative of ρ, evaluated on the upstream side of the node (i,j), so that $\tilde{\rho}_{i,j}$ is a weighted average of the values of ρ at (i,j) and the adjacent upstream nodes. $\mu_{i,j}$ is a function of the local Mach number.

$$\mu_{i,j} = \max(0, 1-M_{i,j}^{-2}) = \max(0, 1-c_{i,j}^2/q_{i,j}^2) \qquad (3.10)$$

This choice of definition for μ appears to have been based on numerical experience rather than a numerical stability analysis.

Artificial compressibility has also been used by Wornom [28] in solving the quasi-one-dimensional Euler equations. The steady state version of his discrete equations is

$$(\tilde{\rho}qA)_i - (\tilde{\rho}qA)_{i-1} = 0 \qquad (3.11)$$

$$(pA+\rho q^2 A)_i - (pA+\rho q^2 A)_{i-1} - \frac{1}{2}(p_i+p_{i-1})(A_i-A_{i-1}) = 0 \qquad (3.12)$$

where A is the cross-sectional area of the streamtube, and p is determined by assuming uniform stagnation enthalpy.

$$p_i = \frac{\gamma-1}{\gamma}\rho_i(h_t - \frac{1}{2}q_i^2) \qquad (3.13)$$

Wornom defined $\tilde{\rho}$, which he called a retarded density, as

$$\tilde{\rho}_i = \rho_i - \mu_i(\rho_i - \rho_{i-1}) \tag{3.14}$$

with

$$\mu_i = \max(0, 1 - M_c^2/M_{i-\frac{1}{2}}^2) \tag{3.15}$$

M_c is a reference Mach number taken to be slightly less than unity and $M_{i-\frac{1}{2}}$ is the average of M_i and M_{i-1}. Again no explanation is given for the choice of formula for μ, and the results presented demonstrate a certain amount of smearing of shocks, indicating that possibly too much dissipation is being added by the artificial compressibility.

The discretization of the steady two-dimensional Euler equations, presented in the last chapter, can be viewed as a natural extension of Wornom's one-dimensional equations. In particular the discretization applied to a single streamtube with a straight centerline reduces to Wornom's equations for a quasi-one-dimensional streamtube. Hence it was decided to follow Wornom's approach in introducing artificial compressibility into the mass equation (2.9).

$$m = \rho_1 q_1 A_{n1} = \rho_2 q_2 A_{n2} \tag{2.9}$$

The modified mass equation is

$$m = \tilde{\rho}_1 q_1 A_{n1} = \tilde{\rho}_2 q_2 A_{n2} \tag{3.16}$$

where

$$\tilde{\rho}_2 = \rho_2 - \mu_2(\rho_2 - \rho_1) \tag{3.17}$$

Rather than using Wornom's definition for μ, however, an analysis was performed to determine the optimum amount of artificial compressibility required to produce a well-posed discrete problem, with sharp shocks and a minimum of undesirable stagnation pressure errors in the smooth supersonic region.

3.2 One-dimensional analysis

The one-dimensional analysis considers linearized perturbations about uniform flow in a constant area duct. To gain insight it is helpful to first analyze the analytic Euler equations with a term in the mass equation corresponding to the artificial compressibility.

$$(\rho - \mu\Delta x \frac{\partial \rho}{\partial x})\, q = \text{const} = m \tag{3.18}$$

$$\rho q^2 + p = \text{const} = P \tag{3.19}$$

$$\frac{\gamma}{\gamma-1}\frac{p}{\rho} + \frac{1}{2} q^2 = \text{const} = h_t \tag{3.20}$$

Now each variable is expressed as a sum of steady uniform part, denoted with an overbar, and a unsteady perturbation.

$$p = \bar{p}(1+p') \quad , \quad \rho = \bar{\rho}(1+\rho') \quad , \quad q = \bar{q} + \bar{c}q' \tag{3.21a-c}$$

When these are substituted into equations (3.18)-(3.20), and second order terms are neglected, the resultant linearized equations are

$$(\rho' - \mu\Delta x \frac{\partial \rho'}{\partial x})\, M + q' = 0 \tag{3.22}$$

$$\gamma M^2 \rho' + 2\gamma M q' + p' = 0 \tag{3.23}$$

$$\frac{1}{\gamma-1}(p'-\rho') + Mq' = 0 \tag{3.24}$$

where M is the Mach number of the steady flow. Eliminating p' and q' using all three equations yields a first order differential equation for ρ'.

$$(M^2-1)\, \rho' - \mu\Delta x(\gamma+1)M^2 \frac{\partial \rho'}{\partial x} = 0 \tag{3.25}$$

This equation has an exponential solution.

$$\rho' \propto \exp(kx) \quad , \quad k = \frac{M^2-1}{\mu\Delta x(\gamma+1)M^2} \tag{3.26}$$

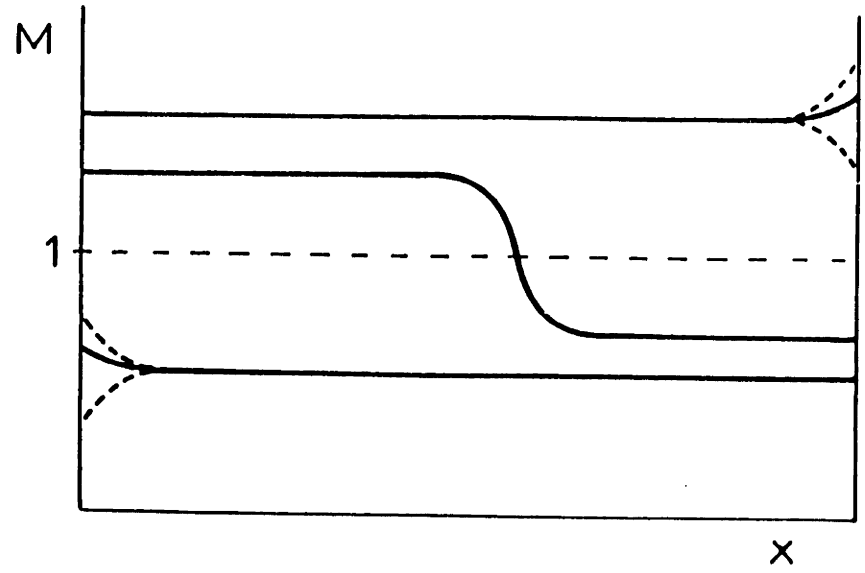


Figure 3.2: Mach number distributions illustrating analytic "boundary layer" behavior.

The most important feature of k is that it changes sign when M passes through M=1. When M<1, k<0, which means that for subsonic flow the linear perturbation produces a "boundary layer" type phenomenon at the inlet boundary, with an exponential solution decaying to zero away from the inlet. When M>1, k>0, which means that for supersonic flow there is a "boundary layer" phenomenon at the outlet boundary. Of particular interest is what happens when there is a shock. In this case the linear perturbation approach is not valid in the shock itself, but it remains valid on either side of the shock. On the supersonic side k>0, and on the subsonic side k<0. Hence on both sides the perturbation from uniform flow decays exponentially away from the shock. Figure 3.2 illustrates all of these features.

This analysis of the modified differential equations is important for interpreting the analysis of the discrete equations, which is performed next, because the behavior of the discrete solution must be qualitatively the same as the behavior of the analytic solution for the discrete problem to be well-posed. Thus the discrete solution should also exhibit the exponential decay away from the shock on both sides and the "boundary layer" type behavior at subsonic inlets and supersonic outlets.

The discrete equations for a straight, uniform area streamtube reduce to

$$[\rho_2 - \mu(\rho_2-\rho_1)]\, q_2 = \text{const} = m \tag{3.27}$$

$$\rho_2 q_2^2 + p_2 = \rho_1 q_1^2 + p_1 = \text{const} = P \tag{3.28}$$

$$\frac{\gamma}{\gamma-1}\frac{p_2}{\rho_2} + \frac{1}{2} q_2^2 = \frac{\gamma}{\gamma-1}\frac{p_1}{\rho_1} + \frac{1}{2} q_1^2 = \text{const} = h_t \tag{3.29}$$

The corresponding linearized perturbation equations, obtained by the same procedure used for the analytic equations, are

$$[\rho_2' - \mu(\rho_2'-\rho_1')]\, M + q_2' = 0 \tag{3.30}$$

$$\gamma M^2 \rho_2' + 2\gamma M\, q_2' + p_2' = 0 \tag{3.31}$$

$$\frac{1}{\gamma-1}(p_2'-\rho_2') + M\, q_2' = 0 \tag{3.32}$$

When q' and p' are eliminated using all three equations the resultant first order difference equation for ρ' is

$$(\mu_{crit} - \mu)\, \rho_2' + \mu\, \rho_1' = 0 \tag{3.33}$$

where

$$\mu_{crit} = \frac{M^2-1}{(\gamma+1)M^2} \tag{3.34}$$

Equation (3.33) has an exponential-type solution

$$\rho_j \propto z^j \qquad , \qquad z = \frac{\mu}{\mu - \mu_{crit}} \tag{3.35}$$

Numerical well-posedness requires that the exponential behavior is qualitatively the same as in the analytic case. For $M<1$, $k<0$, and the corresponding behavior for the discrete problem is $|z|<1$, which is automatically satisfied for $\mu \geq 0$ since for $M<1$, $\mu_{crit}<0$.

For $M>1$, $k>0$, and the corresponding behavior for the discrete problem is $|z|>1$. In this case however $|z|$ can be greater than or less than unity, depending on the value of μ.

a) $\mu > \mu_{crit}$ $\Rightarrow$ $1<z<\infty$

For this range of values of μ, the solution is well-posed and there is a smooth exponential decay away from shocks on the supersonic side.

b) $\mu_{crit} > \mu > \frac{1}{2}\mu_{crit}$ $\Rightarrow$ $-\infty<z<-1$

In this range $|z|$ is still greater than unity so the problem is

well-posed, but now because z is negative there is an oscillatory exponential decay away from the shock.

c) $\frac{1}{2}\mu_{crit} > \mu > 0 \quad \Rightarrow \quad -1<z<0$

In this range, there is insufficient artificial compressibility because $|z|<1$. In actual computations with this level of artificial compressibility the iterative Newton procedure fails because the Jacobian $\partial R/\partial U$ (defined at the beginning of chapter 4) becomes very nearly singular.

The sharpest possible shocks correspond to $z=0$ for $M<1$, and $z=\pm\infty$ for $M>1$, so in one sense the optimal choice for μ is $\mu=\max(0,\mu_{crit})$. However another consideration is the amount of stagnation pressure error produced in the smooth supersonic region by the artificial compressibility, and since it is proportional to μ it suggests that μ should be chosen to be closer to $\mu_{crit}/2$ in smooth flow regions. In actual computations it is found to be preferable to introduce artificial compressibility at high subsonic speeds to prevent the Jacobian $\partial R/\partial U$ from becoming nearly singular, and so the robust definition of μ which is used is

$$\begin{aligned} &M<M_{crit}, \quad \mu = 0 \\ &M>M_{crit}, \quad \mu = \mu_{con}\,\frac{M^2-M_{crit}^2}{(\gamma+1)M^2} \end{aligned} \tag{3.36}$$

where M_{crit} is slightly less than 1.0 and μ_{con} lies between 0.5 and 1.0. Typical values used are $M_{crit}=0.95$ and $\mu_{con} = 0.9$.

To verify the predictions of the analysis, a quasi-one dimensional version of the code was developed by considering only one variable area streamtube with a straight centerline. This code was applied to a Laval nozzle problem with subsonic inlet and outlet, and a choked throat at x=0.5. Figures 3.3 to 3.5 present results from three calcula-

tions with the same boundary conditions and the same M_{crit}=0.9. Each figure displays the compressibility μ, the Mach number M, and the fractional change in stagnation pressure. The plots of μ also have lines indicating μ_{crit} and $\mu_{crit}/2$ for reference purposes. The three cases differ in the values of μ_{con} which are used. The first case uses μ_{con}=1.5. Figure 3.3 shows that there the shock is rather smeared and there is an erroneous increase in stagnation pressure in the supersonic region. In the second case μ_{con}=1.0, and Figure 3.4 shows that the shock is much sharper,and the stagnation pressure errors are smaller, as predicted. In the final case μ_{con}=0.5, and so μ is only slightly above the predicted stability threshold. Both the Mach number and stagnation pressure plots show oscillations which decay slowly upstream away from the shock, as predicted by the analysis. The overall level of stagnation pressure error in the supersonic region is slightly less than for μ_{con}=1.0. When μ_{con} was further reduced to 0.45 the iterative Newton procedure failed and no solution could be obtained. Thus these cases verify the essential points of the analysis, that the stability threshold is $\mu=\mu_{crit}/2$, that the sharpest shocks are obtained by $\mu=\mu_{crit}$ and the minimum stagnation pressure errors are obtained by choosing μ to be slightly greater than $\mu_{crit}/2$.

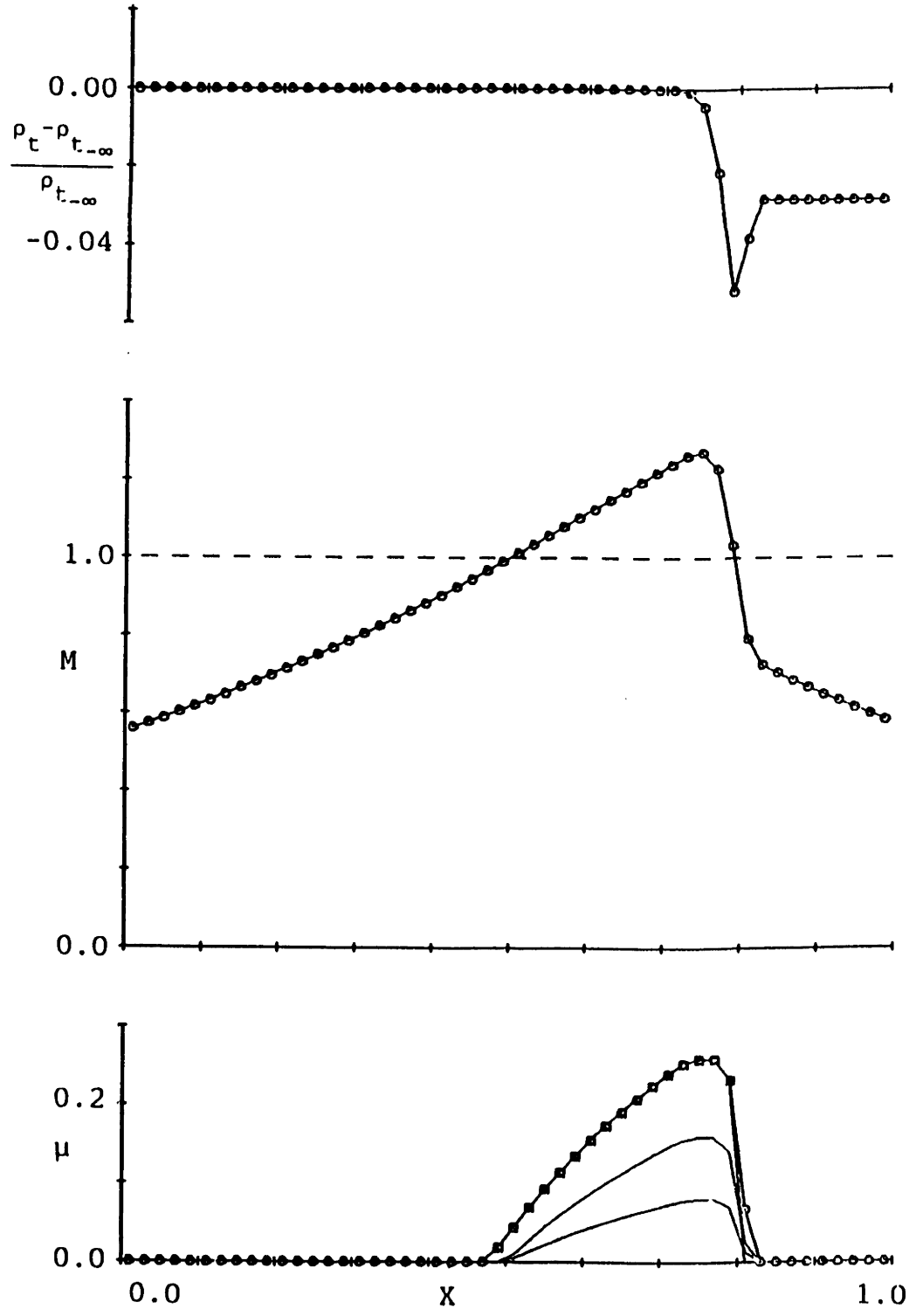


Figure 3.3: Results for 1-D streamtube, with μ_{con}=1.5

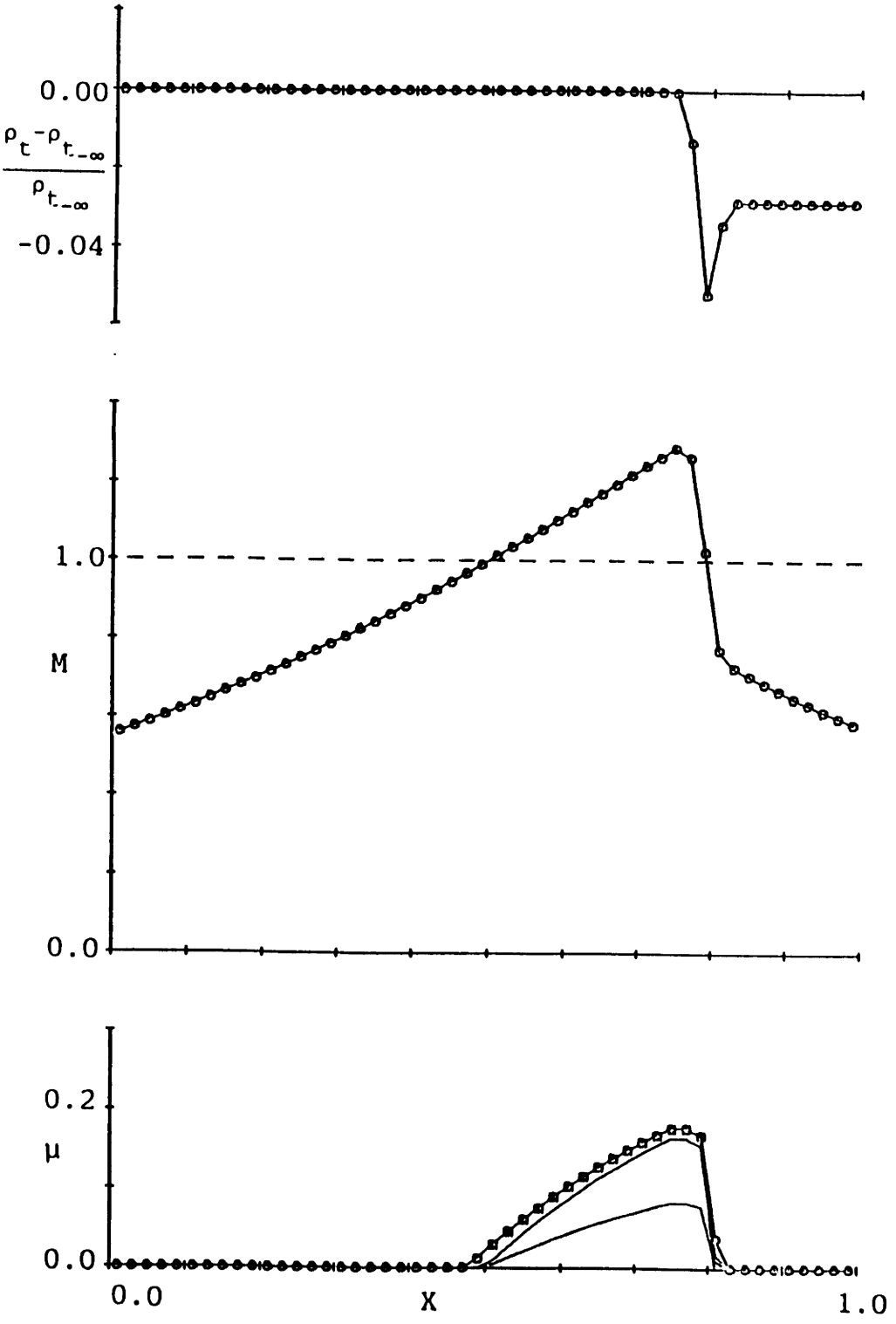


Figure 3.4: Results for 1-D streamtube, with μ_{con}=1.0

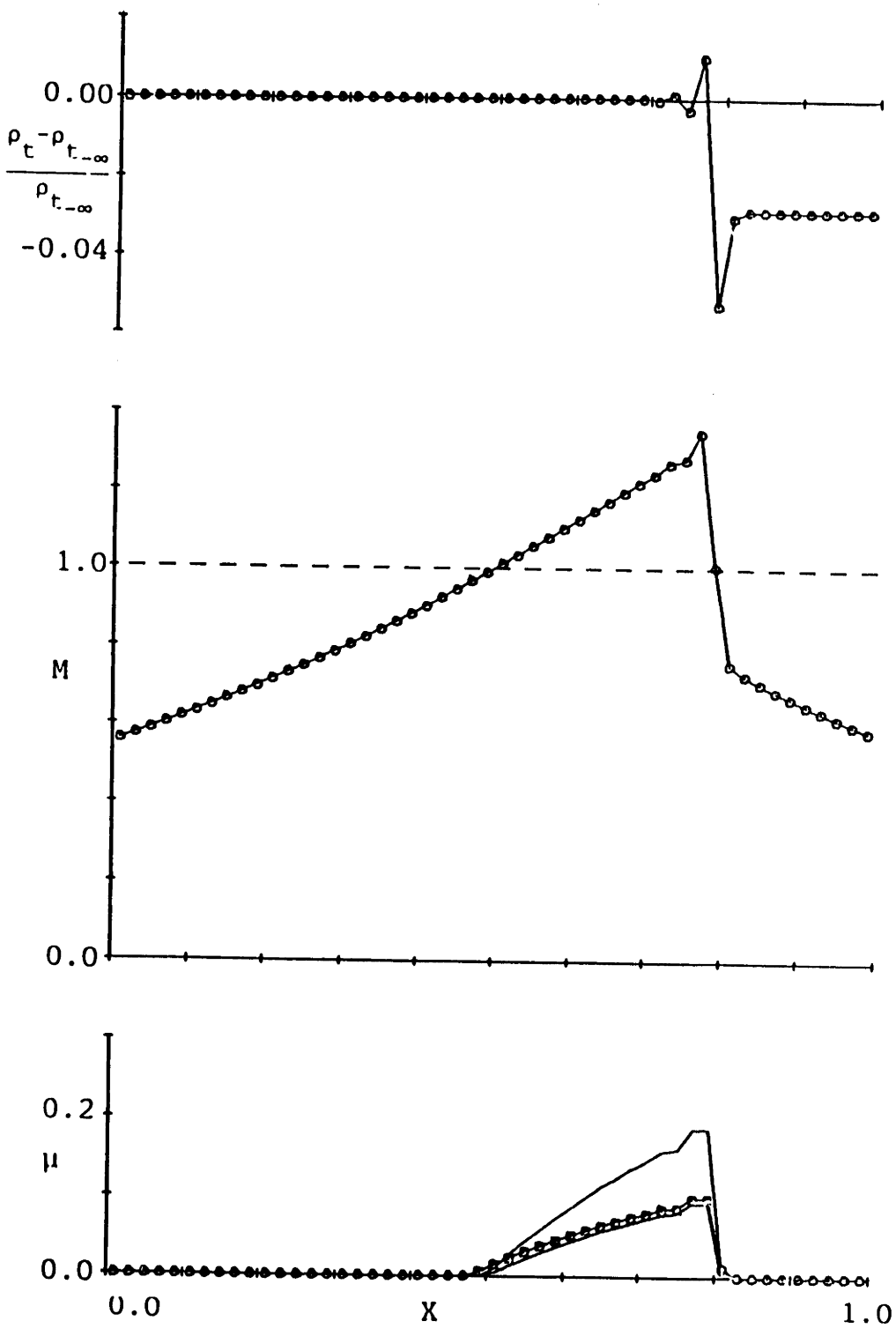


Figure 3.5: Results for 1-D streamtube, with μ_{con}=0.5

3.3 Second order corrections

The artificial compressibility in the mass equation produces a first order error, because the modified density differs from the true density by approximately $\mu\Delta s(\partial\rho/\partial s)$, where $\partial\rho/\partial s$ is the streamwise derivative of the density and Δs is the streamwise distance between two nodes. To correct for this a correction term can be added to the modified density.

$$\tilde{\rho}_2 = \rho_2 - u_2(\rho_2-\rho_1) + \mu_{c_2}(\rho_1-\rho_0) \tag{3.37}$$

When $\mu_{c_2}=0$, this definition for $\tilde{\rho}_2$ is exactly the same as (3.17), but when $\mu_{c_2}=\mu_2$, the modified density $\tilde{\rho}_2$ is approximately equal to $\rho_2-\mu(\Delta s)^2\frac{\partial^2\rho}{\partial s^2}$ and so the truncation error is reduced to being second order. This has the effect in actual calculations of greatly reducing the magnitude of the stagnation pressure errors introduced in the smooth supersonic region. To determine the well-posedness of the new modified mass equation, the analysis of Section 3.2 has to be repeated.

The model analytic mass equation is now

$$(\rho - (\mu-\mu_c)\Delta x\frac{\partial\rho}{\partial x} - \mu_c\Delta x^2\frac{\partial^2\rho}{\partial x^2})\,q = \text{const} = m \tag{3.38}$$

where μ_c is assumed to be less than μ.

After linearizing and eliminating q' and p', the resultant second order differential equation for ρ' is

$$\frac{M^2-1}{(\gamma+1)M^2}\rho' - (\mu-\mu_c)\Delta x\frac{\partial\rho'}{\partial x} - \mu_c\Delta x^2\frac{\partial^2\rho'}{\partial x^2} = 0 \tag{3.39}$$

This equation has two solutions of the form $\rho'\propto\exp(kx)$ where

$$k\Delta x = -\frac{1}{2}(\mu-\mu_c) \pm \frac{1}{2}\left[(\mu-\mu_c)^2 + \frac{4\mu_c(M^2-1)}{(\gamma+1)M^2}\right]^{1/2} \tag{3.40}$$

For M<1, either the two values of k are real and negative, or they are a complex conjugate pair of which the real part is negative. In both cases the result is a combination of two "boundary layer" phenomena at subsonic inlets and on the subsonic side of shocks. For M>1, both roots are real, one is positive and the other is negative, so there is "boundary layer" phenomenon at supersonic inlets and outlets, and on the supersonic side of shocks.

The discrete modified mass equation is

$$[\rho_2 - \mu(\rho_2-\rho_1) + \mu_c(\rho_1-\rho_0)]\, q_2 = \text{const} = m \tag{3.41}$$

After linearizing and eliminating q' and p', the resultant second order difference equation for ρ' is

$$(\mu_{crit} - \mu)\, \rho_2' + (\mu+\mu_c)\, \rho_1' - \mu_c \rho_0' = 0 \tag{3.42}$$

where μ_{crit} is again as defined in equation (3.34).

This equation has two solutions of the form $\rho' \propto z^j$ where

$$z = \frac{-(\mu+\mu_c) \pm [(\mu+\mu_c)^2 + 4\mu_c(\mu_{crit}-\mu)]^{1/2}}{2\,(\mu_{crit}-\mu)} \tag{3.43}$$

For M<1 it can be shown that, provided $0<\mu_c<\mu$, the two roots z_1 and z_2 may both be real, or may form a complex conjugate pair, but in either case they both have magnitude less than unity, and so the qualitative behavior is the same as for the analytic solution.

For M>1 it can be shown that both roots are real, and one root, z_1, has magnitude less than unity, while the other root, z_2, depends on the values of μ and μ_c.

a) $\mu_{crit} < \mu$ => $1 < z_2$

b) $\mu < \mu_{crit} < 2(\mu+\mu_c)$ => $z_2 < -1$

c) $2(\mu+\mu_c) < \mu_{crit}$ => $-1 < z_2 < 0$

Thus for the discrete behavior to be qualitatively the same as the analytic behavior, which requires $|z_1|<1$ and $|z_2|>1$, the artificial compressibility must satisfy $2(\mu+\mu_c)>\mu_{crit}$, and the sharpest shocks are again obtained by $\mu=\mu_{crit}$.

In applications it has been found to be desirable to set $\mu_c=0$ close to shocks, so an algorithm is used which sets $\mu_c=\mu$ in the smooth supersonic region and then smoothly reduces μ_c to zero just before the shock. Full details can be obtained from the program listing. To illustrate the effect of the second order corrections the three test cases presented in the last section were recalculated with the second order corrections. Figures 3.6-3.8 show μ, μ_c, the Mach number M and the fractional change in the stagnation pressure. It can be seen that the stagnation pressure errors in the smooth supersonic region have been almost totally eliminated. The oscillations on the supersonic side of the shock for the case $\mu_{con}=0.5$ are more evident, and the sharpest shock is still obtained by choosing $\mu_{con}=1.0$, although the solution corresponding to $\mu_{con}=1.5$ is not much worse.

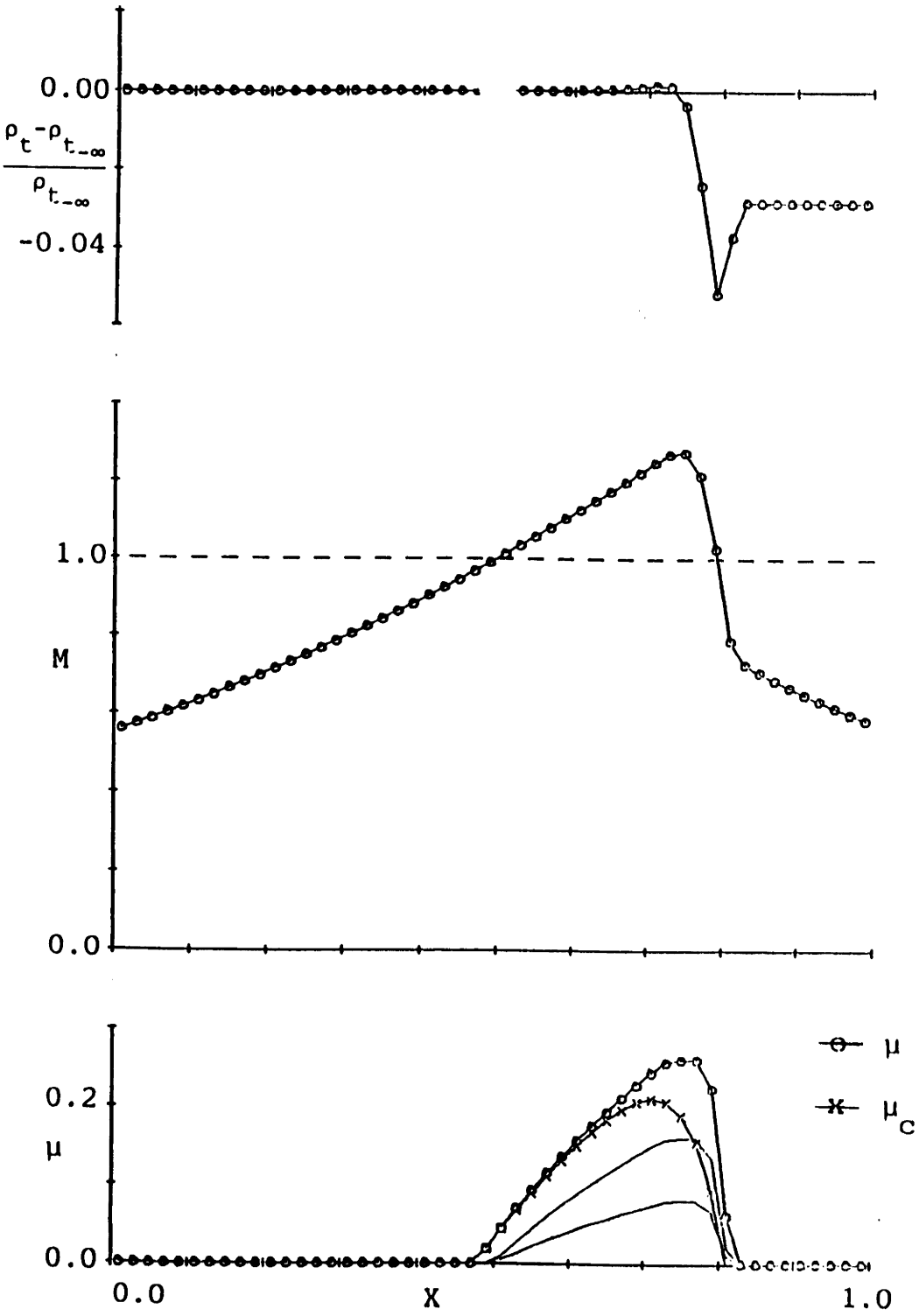


Figure 3.6: Results for 1-D streamtube, with μ_{con}=1.5 and second order density corrections.

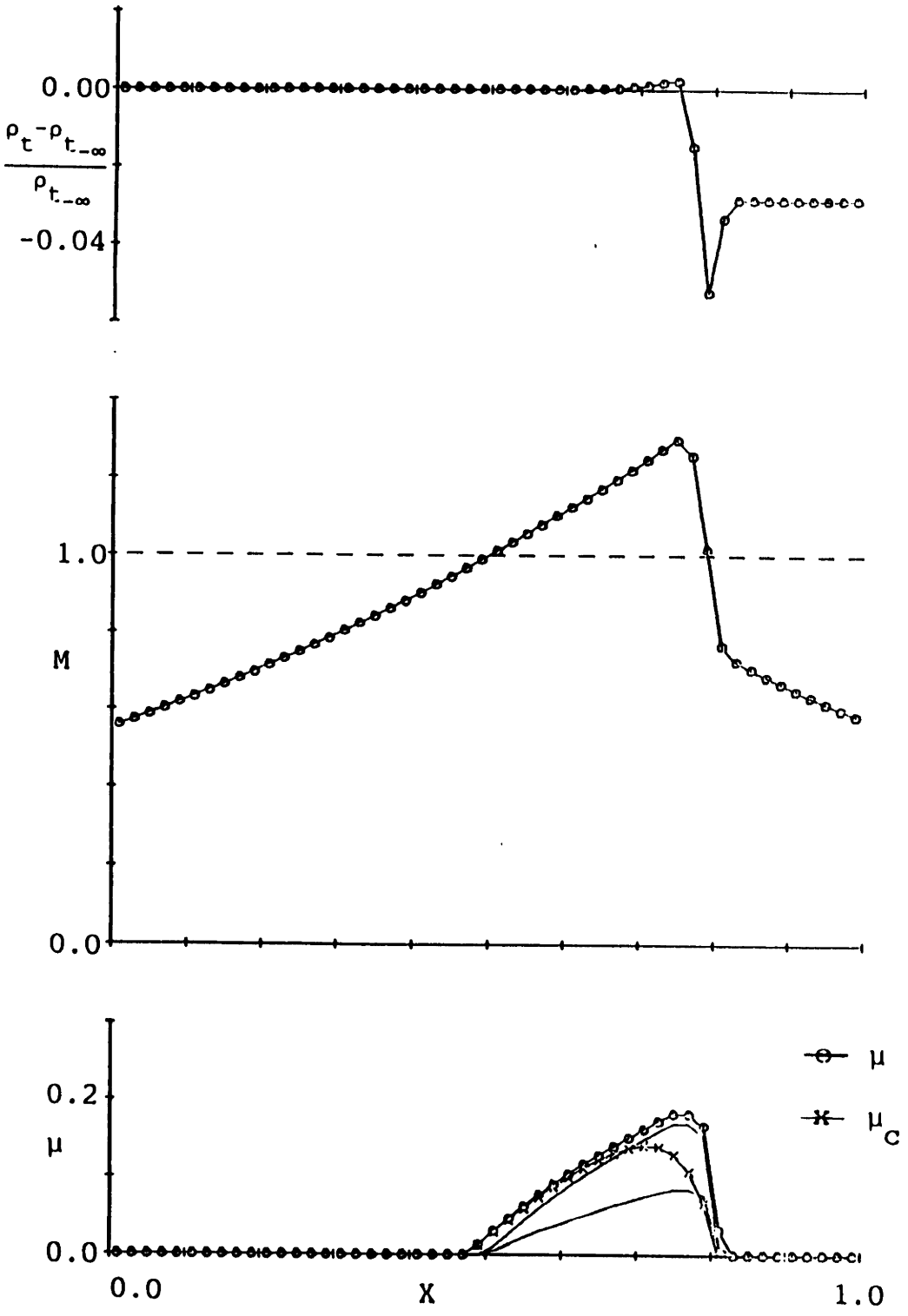


Figure 3.7: Results for 1-D streamtube, with μ_{con}=1.0 and second order density corrections.

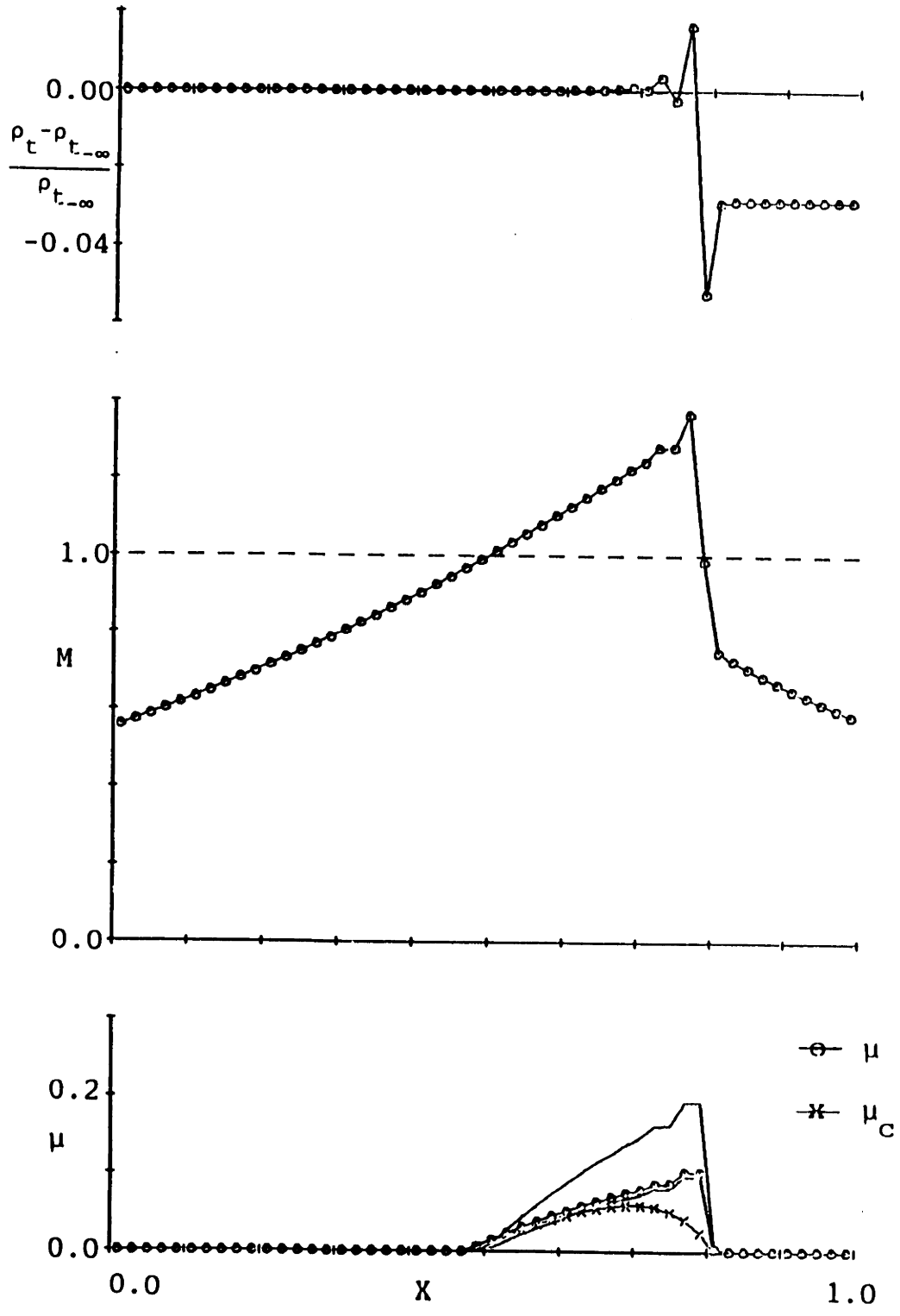


Figure 3.8: Results for 1-D streamtube, with μ_{con}=0.5 and second order density corrections.

4. NEWTON LINEARIZATION

The Newton method for solving the nonlinear scalar equation

$$R(U) = 0 \tag{4.1}$$

is to linearize the function R about the current approximate solution U^n to obtain

$$R(U^{n+1}) = R(U^n+\delta U^n) \simeq R(U^n) + \left(\frac{dR}{dU}\right)^n \delta U^n \tag{4.2}$$

and then setting this equal to zero determines the correction δU.

$$\delta U^n = - R(U^n)/\left(\frac{dR}{dU}\right)^n \tag{4.3}$$

The same approach can be used to solve a system of N nonlinear equations with N variables. The difference is that equation (4.2) is now a vector equation and $\frac{dR}{dU}$ becomes an NxN matrix. Thus δU^n is determined by solving the linearized equation

$$R(U^n) + \left(\frac{\partial R}{\partial U}\right)^n \delta U^n = 0 \tag{4.4}$$

where the i^{th} component of the vector $R(U^n)$ is the i^{th} function R_i evaluated at the current approximate solution U^n, and the $(i,j)^{th}$ element of $\left(\frac{\partial R}{\partial U}\right)^n$ is $\frac{\partial R_i}{\partial U_j}$, the partial derivative (or sensitivity) of R_i with respect to a variation in U_j, the j^{th} component of U, evaluated again at the current approximate solution U^n.

In the application here, the vector of functions R includes all of the discrete Euler equations and the appropriate boundary conditions. Section 4.1 describes the process of linearizing the discrete Euler equations about the current approximate solution to obtain the appropriate rows of the linearized equation system (4.4). Section 4.2 discusses the linearization of the duct boundary conditions. Section 4.3 presents the modifications neccesary due to artificial compressibility. Section 4.4 presents the initialization procedure for the Newton method

and finally Section 4.5 discusses some issues involved in the updating of the solution using the corrections calculated by the Newton method.

4.1 Euler equations

To minimize the computational cost of solving equation (4.4) it is desirable to use the smallest possible number of independent variables, and so the equations are formulated with only the density, ρ, and grid node positions, x,y, as independent variables.

In this section it will first be shown how the other variables at a given iteration level n can be derived from the known values of ρ,x and y. It is then shown that there are still two steady-state equations per computational cell which have not been exactly satisfied. It is the residuals corresponding to these two equations which drive the corrections $\delta\rho$, the change in the density, and δn, the displacement of the node normal to the local streamline. Next, the two equations are linearized with respect to variations in both the independent variables, ρ,x,y, and the dependent variables, p,Π,q and various geometric variables. Lastly, the variations in each of the dependent variables are expressed as linear functions of the variations in the independent variables to obtain the final form of the linearized equations.

At the beginning of iteration n, ρ^n,x^n,y^n are known, together with the mass flux and stagnation enthalpy in each streamtube, which are invariant throughout the iterative process. For each streamtube all the geometric quantities are evaluated using equations (2.4-2.8). Each q is then determined from the mass equation

$$m = \rho_2 q_2 A_{n2} \tag{2.9}$$

which implies

$$q_2^n = \frac{m}{\rho_2^n A_{n2}^n} \quad . \tag{4.5}$$

Next p_2^n is calculated from the energy equation

$$h_t = \frac{\gamma}{\gamma-1}\frac{p_2^n}{\rho_2^n} + \frac{1}{2}q_2^{n2} \tag{2.12}$$

which implies

$$p_2^n = \frac{\gamma-1}{\gamma}\rho_2^n\left(h_t - \frac{1}{2}q_2^{n2}\right) \quad . \tag{4.6}$$

The auxiliary pressure relation

$$\Pi^- + \Pi^+ = p_1 + p_2 + 2P_c \tag{2.18}$$

gives the sum of Π^- and Π^+, the pressures on the streamline faces. The difference between them is obtained by taking a linear combination of the x and y momentum equations (2.10,2.11). Using the auxiliary pressure relation, (2.18), the momentum equations are

$$\rho_1 q_1^2 A_{n1} s_{x1} - \rho_2 q_2^2 A_{n2} s_{x2} + (p_1-p_2)N_y + (\Pi^+-\Pi^-)S_y + P_c(B_y^+-B_y^-) = 0 \tag{4.7}$$

$$\rho_1 q_1^2 A_{n1} s_{y1} - \rho_2 q_2^2 A_{n2} s_{y2} - (p_1-p_2)N_x - (\Pi^+-\Pi^-)S_x - P_c(B_x^+-B_x^-) = 0 \tag{4.8}$$

Taking N_x times (4.7) + N_y times (4.8), and dividing by $|\vec{S}x\vec{N}|$ gives the equation which is termed the normal momentum equation

$$\rho_1 q_1^2 g_1 - \rho_2 q_2^2 g_2 - \Pi^+ + \Pi^- + P_c g_3 = 0 \tag{4.9}$$

where

$$g_1 = \frac{A_{n1}\vec{N}\cdot\hat{s}_1}{|\vec{S}x\vec{N}|} \quad , \quad g_2 = \frac{A_{n2}\vec{N}\cdot\hat{s}_2}{|\vec{S}x\vec{N}|} \tag{4.10a,b}$$

and

$$g_3 = \frac{|\vec{N}x(\vec{B}^+-\vec{B}^-)|}{|\vec{S}x\vec{N}|} = \frac{|\vec{A}_1x\vec{A}_2|}{|\vec{S}x\vec{N}|} \quad . \tag{4.10c}$$

From (4.9), the definition of P_c equation (2.19), and the auxiliary pressure relation, the values of Π^- and Π^+ are calculated, completing the calculation of all the variables at iteration level n.

There are two respects in which these variables do not satisfy the steady state equations. The first is that only one linear combination of the momentum equations, the normal momentum equation, has been satisfied. A second combination which gives the streamwise momentum equation has not been satisfied. Also the value of Π^- on the face of a cell in the j^{th} streamtube is not neccessarily equal to the value of Π^+ on the same face of the cell in the $j-1^{th}$ streamtube, as illustrated in Figure 4.1. In the the final steady state solution both of these conditions must be satisfied, and so it is these errors which are the forcing terms which drive the changes in the density and grid node positions.

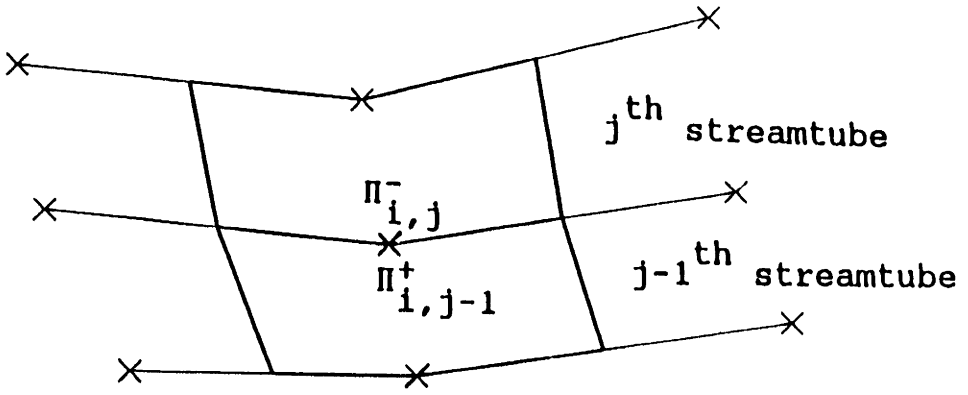


Figure 4.1: Pressures on shared streamline face.

Next the linearized equations have to be determined, expressed in terms of changes in the density and grid node positions. The first equation is obtained by linearizing equations (4.7) and (4.8) and taking S_x times the former plus S_y times the latter, divided by $|\vec{S}x\vec{N}|$. This will be referred to as the streamwise momentum equation since it is a vector dot product of a streamwise vector and the linearized momentum equation.

$$q_1^2 f_1\,\delta\rho_1 + 2\rho_1 q_1 f_1\,\delta q_1 + (\rho_1 q_1^2 f_1/A_{n1})\,\delta A_{n1} + \frac{\rho_1 q_1^2 A_{n1}}{|\vec{S}x\vec{N}|}\,\vec{S}\cdot\delta\hat{s}_1$$

$$- q_2^2 f_2\,\delta\rho_2 - 2\rho_2 q_2 f_2\,\delta q_2 - (\rho_2 q_2^2 f_2/A_{n2})\,\delta A_{n2} - \frac{\rho_2 q_2^2 A_{n2}}{|\vec{S}x\vec{N}|}\,\vec{S}\cdot\delta\hat{s}_2$$

$$+ \frac{p_1-p_2}{|\vec{S}x\vec{N}|}\,|\vec{S}x\delta\vec{N}| + \frac{\Pi^+-\Pi^-}{|\vec{S}x\vec{N}|}\,|\vec{S}x\delta\vec{S}| + \delta p_1 - \delta p_2 + f_3\delta P_c + \underline{P_c\,\frac{|\vec{S}x(\delta\vec{B}^- - \delta\vec{B}^+)|}{|\vec{S}x\vec{N}|}}$$

$$= -\,(\,\rho_1 q_1^2 f_1 - \rho_2 q_2^2 f_2 + p_1 - p_2 + P_c f_3\,) \qquad (4.11)$$

where

$$f_1 = \frac{A_{n1}\vec{S}\cdot\hat{s}_1}{|\vec{S}x\vec{N}|} \quad , \quad f_2 = \frac{A_{n2}\vec{S}\cdot\hat{s}_2}{|\vec{S}x\vec{N}|} \qquad (4.12a,b)$$

and

$$f_3 = \frac{|\vec{S}x(\vec{B}^+ - \vec{B}^-)|}{|\vec{S}x\vec{N}|} = \frac{|\vec{B}^- x\vec{B}^+|}{|\vec{S}x\vec{N}|} \quad . \qquad (4.12c)$$

The term underlined by dots is extremely small since $P_c = O(\Delta x^2)$ and so this term is neglected.

The second equation, which will be called the linearized normal momentum equation, is N_x times the linearized x-momentum equation (4.7) plus N_y times the linearized y-momentum equation (4.8) divided by $|\vec{S}x\vec{N}|$.

$$q_1^2 g_1\,\delta\rho_1 + 2\rho_1 q_1 g_1\,\delta q_1 + (\rho_1 q_1^2 g_1/A_{n1})\,\delta A_{n1} + \frac{\rho_1 q_1^2 A_{n1}}{|\vec{S}x\vec{N}|}\,\vec{N}\cdot\delta\vec{s}_1$$

$$- q_2^2 g_2\,\delta\rho_2 - 2\rho_2 q_2 g_2\,\delta q_2 - (\rho_2 q_2^2 g_2/A_{n2})\,\delta A_{n2} - \frac{\rho_2 q_2^2 A_{n2}}{|\vec{S}x\vec{N}|}\,\vec{N}\cdot\delta\vec{s}_2$$

$$- \frac{p_1 - p_2}{|\vec{S}x\vec{N}|}\,|\vec{S}x\delta\vec{N}| - \frac{\Pi^+ - \Pi^-}{|\vec{S}x\vec{N}|}\,|\vec{S}x\delta\vec{S}| + \delta\Pi^+ - \delta\Pi^- + g_3\delta P_c + P_c\,\frac{|\vec{N}x(\delta\vec{B}^- - \delta\vec{B}^+)|}{|\vec{S}x\vec{N}|}$$

$$= - \left(\rho_1 q_1^2 g_1 - \rho_2 q_2^2 g_2 + \Pi^+ - \Pi^- + P_c g_3 \right) \qquad (4.13)$$

g_1, g_2 and g_3 were previously defined in equations (4.10a-c). The term underlined by dots is again neglected since $P_c = O(\Delta x^2)$. Because of of equation (4.9), which was used to calculate Π^- and Π^+, the right hand side of equation (4.13) is zero.

Ultimately both of these linearized momentum equations need to be expressed solely in terms of variations in density and grid node position. To eliminate the $\delta\Pi$ terms in equation (4.13) two relations are used. The first is the linearized auxiliary pressure relation

$$\delta\Pi^- + \delta\Pi^+ = \delta p_1 + \delta p_2 + 2\delta P_c \qquad (4.14)$$

which means that either one of the $\delta\Pi$ can be eliminated leaving only the other one to be eliminated. The second relation is that in the converged steady state solution the Π^+ of a cell in the $j\text{-}1^{th}$ streamtube must be identical to the Π^- of a cell in the j^{th} streamtube as was illustrated in Figure 4.1. The linearized statement of this is

$$\Pi^+_{j-1} + \delta\Pi^+_{j-1} = \Pi^-_{j-1} + \delta\Pi^-_{j-1} \qquad (4.15)$$

which may be rewritten as

$$\delta\Pi^+_{j-1} - \delta\Pi^-_{j} = - (\Pi^+_{j-1} - \Pi^-_{j}) \quad . \qquad (4.16)$$

Using this relation to eliminate the $\delta\Pi^+$ of the $j\text{-}1^{th}$ streamtube cell and the $\delta\Pi^-$ of the j^{th} streamtube cell, one gets a combined normal

momentum equation covering the two cells (see Figure 4.1) and containing no $\delta\Pi$ terms.

$$\left[q_1^2 g_1\, \delta\rho_1 + 2\rho_1 q_1 g_1\, \delta q_1 + (\rho_1 q_1^2 g_1 / A_{n1})\, \delta A_{n1} + \frac{\rho_1 q_1^2 A_{n1}}{|\vec{S}\times\vec{N}|}\, \vec{N}\cdot\delta\hat{s}_1 \right.$$

$$- q_2^2 g_2\, \delta\rho_2 - 2\rho_2 q_2 g_2\, \delta q_2 - (\rho_2 q_2^2 g_2 / A_{n2})\, \delta A_{n2} + \frac{\rho_2 q_2^2 A_{n2}}{|\vec{S}\times\vec{N}|}\, \vec{N}\cdot\delta\hat{s}_2$$

$$\left. - \frac{p_1 - p_2}{|\vec{S}\times\vec{N}|}\, |\vec{S}\times\delta\vec{N}| - \frac{\Pi^+ - \Pi^-}{|\vec{S}\times\vec{N}|}\, |\vec{S}\times\delta\vec{S}| - \delta p_1 - \delta p_2 - (2-g_3)\delta P_c \right]_{j-1} +$$

$$\left[q_1^2 g_1\, \delta\rho_1 + 2\rho_1 q_1 g_1\, \delta q_1 + (\rho_1 q_1^2 g_1 / A_{n1})\, \delta A_{n1} + \frac{\rho_1 q_1^2 A_{n1}}{|\vec{S}\times\vec{N}|}\, \vec{N}\cdot\delta\hat{s}_1 \right.$$

$$- q_2^2 g_2\, \delta\rho_2 - 2\rho_2 q_2 g_2\, \delta q_2 - (\rho_2 q_2^2 g_2 / A_{n2})\, \delta A_{n2} + \frac{\rho_2 q_2^2 A_{n2}}{|\vec{S}\times\vec{N}|}\, \vec{N}\cdot\delta\hat{s}_2$$

$$\left. - \frac{p_1 - p_2}{|\vec{S}\times\vec{N}|}\, |\vec{S}\times\delta\vec{N}| - \frac{\Pi^+ - \Pi^-}{|\vec{S}\times\vec{N}|}\, |\vec{S}\times\delta\vec{S}| + \delta p_1 + \delta p_2 + (2+g_3)\delta P_c \right]_{j}$$

$$= -2(\Pi^+_{j-1} - \Pi^-_j) \qquad (4.17)$$

Equations (4.11) and (4.17) are the linearized equations to be solved, but there is still the matter of eliminating the δP_c, δp and δq terms. The δp terms are eliminated using the enthalpy equation.

$$h_t = \frac{\gamma}{\gamma-1}\frac{p}{\rho} + \frac{1}{2} q^2 \qquad (2.12)$$

Thus, by differentiating

$$\delta p = \left(\frac{\partial p}{\partial q}\right)_{\rho=\text{const}} \delta q + \left(\frac{\partial p}{\partial \rho}\right)_{q=\text{const}} \delta\rho \qquad (4.18a)$$

where

$$\left(\frac{\partial p}{\partial \rho}\right)_{q=\text{const}} = \frac{p}{\rho} \qquad (4.18b)$$

$$\left(\frac{\partial p}{\partial q}\right)_{\rho=\text{const}} = -\frac{\gamma-1}{\gamma}\rho q \quad . \tag{4.18c}$$

The δq terms are eliminated using the mass equation

$$m = \rho q A_n \tag{2.9}$$

which when linearized gives

$$\delta q = \left(\frac{\partial q}{\partial \rho}\right)_{A=\text{const}} \delta\rho + \left(\frac{\partial q}{\partial A}\right)_{\rho=\text{const}} \delta A_n \tag{4.19a}$$

where

$$\left(\frac{\partial q}{\partial \rho}\right)_{A=\text{const}} = -\frac{q}{\rho} \tag{4.19b}$$

$$\left(\frac{\partial q}{\partial A}\right)_{\rho=\text{const}} = -\frac{q}{A_n} \quad . \tag{4.19c}$$

Similarly the δP_c term can be eliminated by differentiating its defining equation, (2.20). The resulting expressions are rather complicated and can be obtained from the program listing.

Also the geometric variations have to be expressed in terms of the primitive grid node movements. Each node moves a distance δn in a direction $\hat{n}$ specified to be approximately normal to the streamline. Thus one gets the variational relations

$$\delta\vec{S} = -\tfrac{1}{2}\delta n_1^-\hat{n}_1^- - \tfrac{1}{2}\delta n_1^+\hat{n}_1^+ + \tfrac{1}{2}\delta n_3^-\hat{n}_3^- + \tfrac{1}{2}\delta n_3^+\hat{n}_3^+ \tag{4.20}$$

$$\delta\vec{N} = -\tfrac{1}{2}\delta n_1^-\hat{n}_1^- + \tfrac{1}{2}\delta n_1^+\hat{n}_1^+ - \delta n_2^-\hat{n}_2^- + \delta n_2^+\hat{n}_2^+ - \tfrac{1}{2}\delta n_3^-\hat{n}_3^- + \tfrac{1}{2}\delta n_3^+\hat{n}_3^+ \tag{4.21}$$

$$\delta\hat{s}_1 = \frac{1}{s_1}\delta\vec{s}_1 - \frac{1}{s_1}(\hat{s}_1\cdot\delta\vec{s}_1)\,\hat{s}_1 = \frac{1}{s_1}\hat{s}_1 x(\delta\vec{s}_1 x\hat{s}_1) \tag{4.22}$$

from which, for example, one gets

$$|\vec{S}x\delta\vec{S}| = -\tfrac{1}{2}|\vec{S}x\hat{n}_1^-|\ \delta n_1^- - \tfrac{1}{2}|\vec{S}x\hat{n}_1^+|\ \delta n_1^+ + \tfrac{1}{2}|\vec{S}x\hat{n}_3^-|\ \delta n_3^- + \tfrac{1}{2}|\vec{S}x\hat{n}_3^+|\ \delta n_3^+ \tag{4.23}$$

$$\delta A_{n1} = \frac{1}{s_1}(\vec{A}_1 \cdot \hat{s}_1)|\delta\vec{s}_1 x \hat{s}_1| + |\hat{s}_1 x \delta\vec{A}_1| = |\hat{s}_1 x (\delta\vec{A}_1 - \frac{\vec{A}_1 \cdot \hat{s}_1}{s_1} \delta\vec{s}_1)|$$

$$= (\frac{\partial A_{n1}}{\partial n_1^-}) \delta n_1^- + (\frac{\partial A_{n1}}{\partial n_1^+}) \delta n_1^+ + (\frac{\partial A_{n1}}{\partial n_2^-}) \delta n_2^- + (\frac{\partial A_{n1}}{\partial n_2^+}) \delta n_2^+ \qquad (4.24a)$$

where

$$\frac{\partial A_{n1}}{\partial n_1^-} = |\hat{s}_1 x \hat{n}_1^-|(-\frac{1}{2} + \frac{\vec{A}_1 \cdot \hat{s}_1}{2s_1}) \qquad (4.24b)$$

$$\frac{\partial A_{n1}}{\partial n_1^+} = |\hat{s}_1 x \hat{n}_1^+|(\frac{1}{2} + \frac{\vec{A}_1 \cdot \hat{s}_1}{2s_1}) \qquad (4.24c)$$

$$\frac{\partial A_{n1}}{\partial n_2^-} = |\hat{s}_1 x \hat{n}_2^-|(-\frac{1}{2} - \frac{\vec{A}_1 \cdot \hat{s}_1}{2s_1}) \qquad (4.24d)$$

$$\frac{\partial A_{n1}}{\partial n_2^+} = |\hat{s}_1 x \hat{n}_2^+|(\frac{1}{2} - \frac{\vec{A}_1 \cdot \hat{s}_1}{2s_1}) \quad . \qquad (4.24e)$$

4.2 Duct boundary conditions

The linearized form of the solid wall boundary condition is simply

$$\delta n_{i,1} = \delta n_{i,J} = 0 \tag{4.25}$$

since the grid nodes are fixed on the lower and upper sides of the duct.

The linearized form of the inlet stagnation density condition

$$\rho_t \equiv \rho_1 (1-q_1^2/2h_t)^{-1/(\gamma-1)} = (\rho_t)_{specified} \tag{2.13}$$

is

$$\frac{\rho_t}{\rho_1} \delta\rho_1 + \frac{q_1 \rho_t}{(\gamma-1)(h_t - q_1^2/2)} \delta q_1 = (\rho_t)_{specified} - \rho_t \quad . \tag{4.26}$$

The linear variation δq is expressed in terms of variations in the density and grid node positions using the linearized mass equation and geometric definitions, as done in the last section.

The linearized form of the inlet condition

$$y_{1,j+1} - y_{1,j} = (y_{1,J} - y_{1,1})\, m_j / m_{total} \tag{2.15}$$

is simply

$$\delta n_{1,j+1} - \delta n_{1,j} = 0 \quad . \tag{4.27}$$

since the inlet area $y_{1,J} - y_{1,1}$ and the mass fluxes are all constants.

The linearized form of the outlet condition

$$(y_{I,j+1} - y_{I,j}) - (y_{I-1,j+1} - y_{I-1,j}) = 0 \tag{2.16}$$

is simply

$$(\delta n_{I,j+1} - \delta n_{I,j}) - (\delta n_{I-1,j+1} - \delta n_{I-1,j}) = 0 \quad . \tag{4.28}$$

4.3 Artificial compressibility

When the local flow is supersonic, or close to supersonic, artificial compressibility appears in the mass equation, as explained in Chapter 3. This modifies the linearization in Section 4.1 in two ways. The first difficulty arises in calculating q^n using the modified mass equation.

$$m = \tilde{\rho}_2 q_2 A_{n2} \tag{3.16}$$

where

$$\tilde{\rho}_2 = \rho_2 - \mu_2(\rho_2-\rho_1) + \mu_{c2}(\rho_1-\rho_0) \tag{3.37}$$

and μ and μ_c are as defined in chapter 3.

The problem with using equation (3.16) to calculate q_2 is that μ_2, and hence $\tilde{\rho}_2$, are functions of q_2. This problem is overcome by calculating a temporary value q_2^* using μ_2^{n-1}, the value of μ_2 at level n-1.

$$q_2^* = m \,/\, A_{n2}^n[\rho_2^n - \mu_2^{n-1}(\rho_2^n-\rho_1^n) + \mu_{c2}^{n-1}(\rho_1^n-\rho_0^n)] \tag{4.29}$$

This value of q_2^* is then used to calculate μ_2^n, which in turn is used to calculate q_2^n.

$$q_2^n = m \,/\, A_{n2}^n[\rho_2^n - \mu_2^n(\rho_2^n-\rho_1^n) + \mu_{c2}^{n-1}(\rho_1-\rho_0)] \tag{4.30}$$

Note that μ_c is held fixed at its value derived at the end of iteration level n-1.

The second change is in the linearized mass equation. Differentiating (3.16) and (3.37) gives

$$\delta q_2 = - \frac{q_2}{\rho_2} \left[(1-\mu_2)\delta\rho_2 + (\mu_2+\mu_{c2})\delta\rho_1 - \mu_{c2}\delta\rho_0 - (\rho_2-\rho_1)\delta\mu_2 + (\rho_1-\rho_0)\delta\mu_{c2} \right]$$

$$- \frac{q_2}{A_{n2}} \delta A_{n2} \quad . \tag{4.31}$$

Three approximations are now made. The first two are that $\delta\rho_0$ and $\delta\mu_{c_2}$ are both zero. The third approximation is that $\delta\mu_2$ is expressed in terms of variations in density, by assuming that velocity changes are isentropic. This is necessary since μ_2 is actually a function of q_2 and q_1, not ρ_2 and ρ_1. Thus $\delta\mu_2$ is approximately given by

$$\delta\mu_2 = \frac{\partial\mu_2}{\partial q_2}\left(\frac{\partial q_2}{\partial\rho_2}\right)_{isen} \delta\rho_2 + \frac{\partial\mu_2}{\partial q_1}\left(\frac{\partial q_1}{\partial\rho_1}\right)_{isen} \delta\rho_1 \quad . \tag{4.32}$$

With these three approximations, δq_2 is given by

$$\delta q_2 = \left(\frac{\partial q_2}{\partial\rho_2}\right) \delta\rho_2 + \left(\frac{\partial q_2}{\partial\rho_1}\right) \delta\rho_1 + \left(\frac{\partial q_2}{\partial A_2}\right) \delta A_{n2} \tag{4.33a}$$

where

$$\frac{\partial q_2}{\partial\rho_2} = - \frac{q_2}{\rho_2} \left[(1-\mu_2) - (\rho_2-\rho_1)\frac{\partial\mu_2}{\partial q_2}\left(\frac{\partial q_2}{\partial\rho_2}\right)_{isen} \right] \tag{4.33b}$$

$$\frac{\partial q_2}{\partial\rho_1} = - \frac{q_2}{\rho_2} \left[(\mu_2+\mu_{c2}) - (\rho_2-\rho_1)\frac{\partial\mu_2}{\partial q_1}\left(\frac{\partial q_1}{\partial\rho_1}\right)_{isen} \right] \tag{4.33c}$$

and

$$\frac{\partial q_2}{\partial A_n} = - \frac{q_2}{A_{n2}} \quad . \tag{4.34}$$

4.4 Initialization of Newton Solution

The Newton iterative procedure has to be supplied with an initial, approximate solution. If this initial guess is very poor then it could affect the robustness of the method. In particular it is important for this problem to have a reasonably good initial position for the streamlines. This is achieved by using an elliptic grid generator which produces a grid, one set of whose grid lines correspond to streamlines of incompressible flow. The method was first developed by Thompson [26]. A transformed set of coordinates (ξ,η) is defined by the equations

$$\xi_{xx} + \xi_{yy} = 0 \qquad (4.35a)$$

$$\eta_{xx} + \eta_{yy} = 0 \qquad (4.35b)$$

After changing coordinate systems these two equations become

$$\alpha\, x_{\xi\xi} - 2\beta\, x_{\xi\eta} + \gamma\, x_{\eta\eta} = 0 \qquad (4.36a)$$

$$\alpha\, y_{\xi\xi} - 2\beta\, y_{\xi\eta} + \gamma\, y_{\eta\eta} = 0 \qquad (4.36b)$$

where,

$$\alpha = x_\eta^2 + y_\eta^2 \quad , \quad \beta = x_\xi x_\eta + y_\xi y_\eta \quad , \quad \gamma = x_\xi^2 + y_\xi^2 \quad . \qquad (4.37a\text{-}c)$$

The line $\xi=0$ is chosen to correspond to the physical inlet boundary and the line $\xi=1$ corresponds to the outlet boundary. The line $\eta=0$ corresponds to the lower duct wall, or in the case of a cascade the suction surface of the blade together with the stagnation streamline whose position is guessed to be a straight line of the correct angle coming from the leading and trailing edges of the blade. Similarly the line $\eta=1$ corresponds to the upper duct wall, or the pressure surface in the case of a cascade. The computational grid is defined to be uniformly spaced in the ξ direction, and in the η direction the spacing is proportional to the mass flux in each streamtube.

$$\xi_{i,j} = i \qquad (4.38a)$$

$$\eta_{i,j} = \sum_{k=1}^{j-1} \frac{m_k}{m_{total}} \quad . \tag{4.38b}$$

The (x,y) coordinates of the grid boundaries are specified, and then equations (4.36a,b) are solved by a standard SLOR relaxation procedure to obtain the coordinates of the interior nodes. Because of the choice of the Laplace equation, (4.35b), to define η, the η=const lines are streamlines of the correspoding incompressible flow, and so the grid produced is an excellent approximation to the grid corresponding to the compressible flow solution.

The initialization of the densities is much less important, and all the calculations presented in this thesis were obtained by initializing all densities to be equal to the value corresponding to a Mach number of 0.5. This was an arbitrary choice, but it was found that alternative initializations made a difference of no more than one or two in the total number of Newton iterations.

4.5 Updating of Newton Solution

For most Newton iterations the solution is updated by simply adding the calculated changes in density and node position.

$$\delta\rho_{i,j}^{n+1} = \rho_{i,j}^{n} + \delta\rho_{i,j} \tag{4.39a}$$

$$x_{i,j}^{n+1} = x_{i,j}^{n} + (\hat{n}_x)_{i,j}\delta n_{i,j} \tag{4.39b}$$

$$y_{i,j}^{n+1} = y_{i,j}^{n} + (\hat{n}_y)_{i,j}\delta n_{i,j} \tag{4.39c}$$

For some iterations however equation (4.39a) would produce some densities that would be negative and would cause the Newton procedure to fail. To prevent this from happening an under-relaxation factor is employed and equations (4.39a-c) are modified to become

$$\delta\rho_{i,j}^{n+1} = \rho_{i,j}^{n} + r\,\delta\rho_{i,j} \tag{4.40a}$$

$$x_{i,j}^{n+1} = x_{i,j}^{n} + r\,(\hat{n}_x)_{i,j}\delta n_{i,j} \tag{4.40b}$$

$$y_{i,j}^{n+1} = y_{i,j}^{n} + r\,(\hat{n}_y)_{i,j}\delta n_{i,j} \tag{4.40c}$$

where r is chosen to be either 1 or the value which produces a maximum density change of factor 2, if this value is less than 1. Chapter 9, which lists iteration histories for all the calculated test cases, also lists the value of r whenever it is not 1. It will be seen that this clamp is usually only needed in the first iteration and during the formation and movement of a strong shock. Other methods of clamping, such as making r a local quantity instead of a global quantity, are also possible but have not been investigated since the current clamping procedure works very well.

In cases with blunt leading edges, the clamp r is also used to prevent the position of the stagnation point node changing by more than half the node spacing. In addition to this, whenever the stagnation

node moves half the node spacing, or whenever the total movement over a number of iterations is this large, an adjustment algorithm is employed which adjusts the position of the nodes by moving them along their streamline in order to maintain a smooth distribution of nodes along each streamline. The iteration histories in Chapter 9 also indicate when this streamwise adjustment algorithm is used. Typically it is only used once or twice during the earlier Newton iterations when the changes are largest.

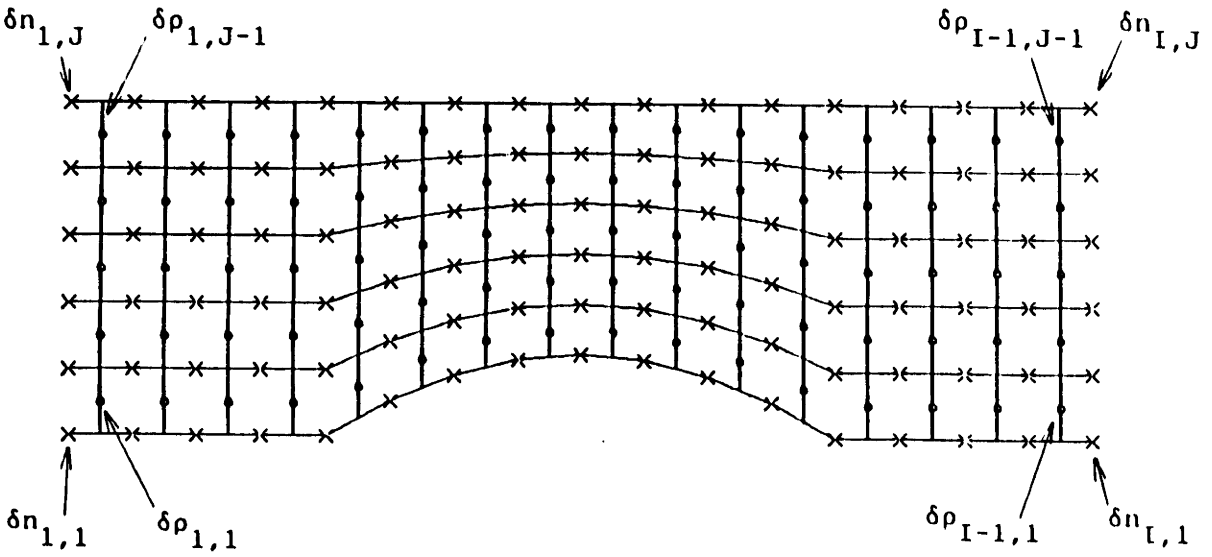


Figure 5.1: Global indexing system for unknown variables.

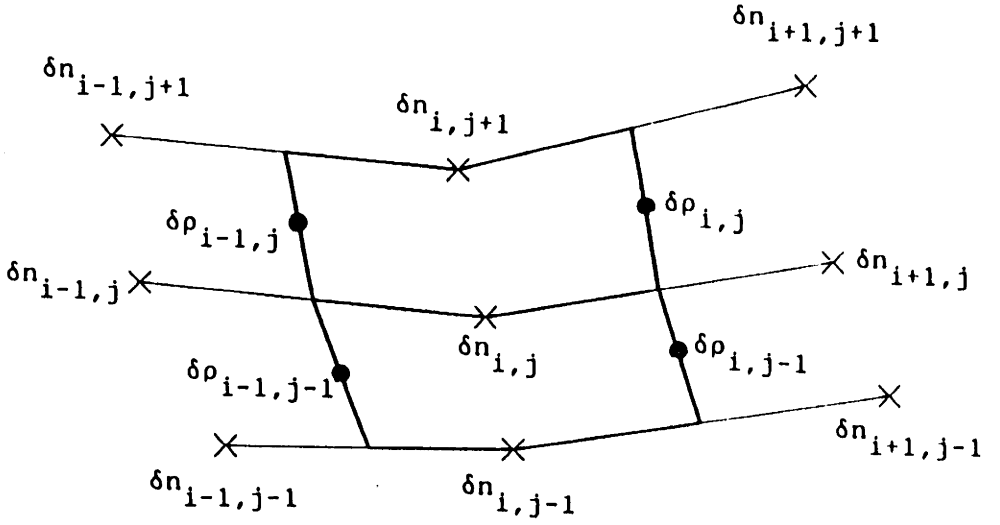


Figure 5.2: Indexing system for a particular pair of cells.

5. DIRECT METHOD

The simplest approach to solving the linear system of Newton equations is to solve the system directly. This chapter shows that the equations can be assembled into a block tridiagonal form in which the variables correponding to each block are all of the density and node changes at one particular streamwise station. Due to artificial compressibility there is an additional fourth diagonal of blocks in the supersonic region, and so the solution procedure is slightly modified from a conventional block tri-diagonal algorithm [1,8].

5.1 Assembling the linearized equations

As indicated schematically in Figure 5.1 the unknowns are $\delta n_{i,j}$ for $1 \leq i \leq I$, $1 \leq j \leq J$, and $\delta \rho_{i,j}$ for $1 \leq i \leq I-1$, $1 \leq j \leq J-1$. Figure 5.2 shows the indexing system for a particular pair of conservation cells. Thus for each i, that is for each streamwise station, there are 2J-1 unknowns. The modified block tridiagonal form into which the linearized equations are assembled is

$$\begin{bmatrix} A_1 & C_1 & & & & \\ B_2 & A_2 & C_2 & & & \\ Z_3 & B_3 & A_3 & C_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & Z_{I-1} & B_{I-1} & A_{I-1} & C_{I-1} \\ & & & Z_I & B_I & A_I \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \\ \\ D_{I-1} \\ D_I \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \\ \\ R_{I-1} \\ R_I \end{bmatrix} \tag{5.1}$$

Each of the matrices Z,B,A,C is square and has dimension 2J-1, the number of unknowns at each streamwise station. Likewise each vector D

and R has length 2J-1. The matrix D_i contains all of the unknown linear perturbations at the i^{th} streamwise station, arranged in the following order:

$$D_i = \begin{bmatrix} \delta n_{i,1} \\ \delta n_{i,2} \\ \vdots \\ \delta n_{i,J-1} \\ \delta n_{i,J} \\ \delta \rho_{i,1} \\ \delta \rho_{i,2} \\ \vdots \\ \delta \rho_{i,J-2} \\ \delta \rho_{i,J-1} \end{bmatrix} \tag{5.2}$$

The matrices Z_i, B_i, A_i, C_i, for all values of i except 1 and I, have the same banded structure, shown in Figure 5.3 with a reminder above each column of the variable in D which corresponds to it. A cross, "x," in the matrix means that there is a non-zero entry. A circle, "o," means the entry is zero if there is no artificial compressibility. A blank means the entry is always zero. Rows 1 and J are the solid wall boundary conditions, (4.25). Rows 2 to J-1 are the linearized normal momentum equation (4.17) for j going from 2 to J-1. Finally rows J+1 to 2J-1 are the linearized streamwise momentum equation, (4.11) for j going from 1 to J-1. Note that the Z matrix is entirely zero if there is no artificial compressibility.

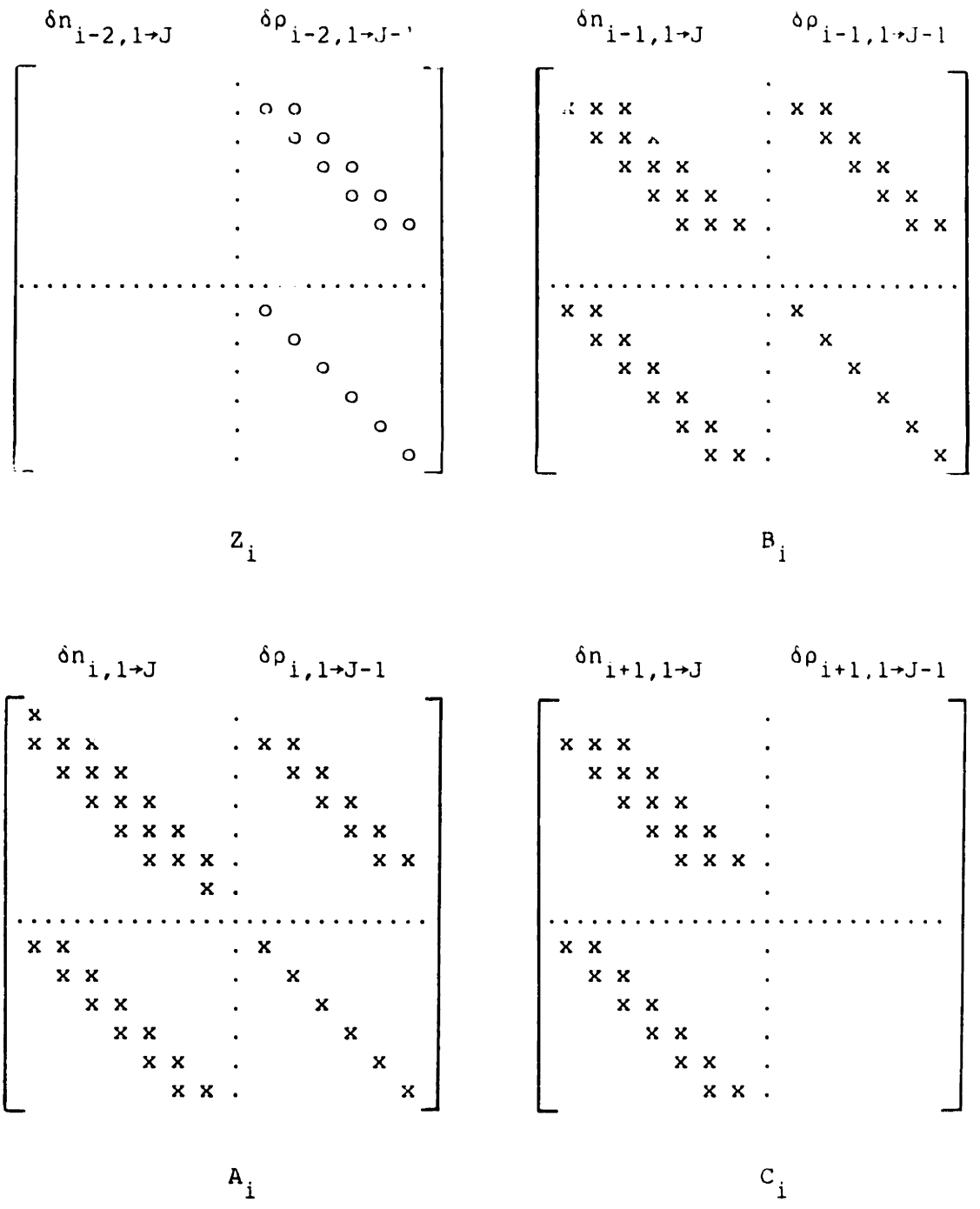


Figure 5.3: Structure of matrices Z_i,B_i,A_i and C_i.

Figure 5.4 shows A_1 and C_1 the matrices in the first block row. Rows 1 and J are the solid wall boundary condition, rows 2 to J-1 are the inlet conditions, (4.27), and rows J+1 to 2J-1 are the streamwise momentum equations for the first column of conservation cells.

Figure 5.5 shows Z_I, B_I and A_I the matrices in the last block row. Remember that there are no $\delta\rho$ variables at i=I so the last J-1 variables in the subvector D_I are dummy variables. As a consequence rows J+1 to 2J-1 simply correspond to setting these dummy variables equal to zero. Rows 1 and J are again the solid wall conditions, and rows 2 to J-1 are the outlet conditions, (4.28).

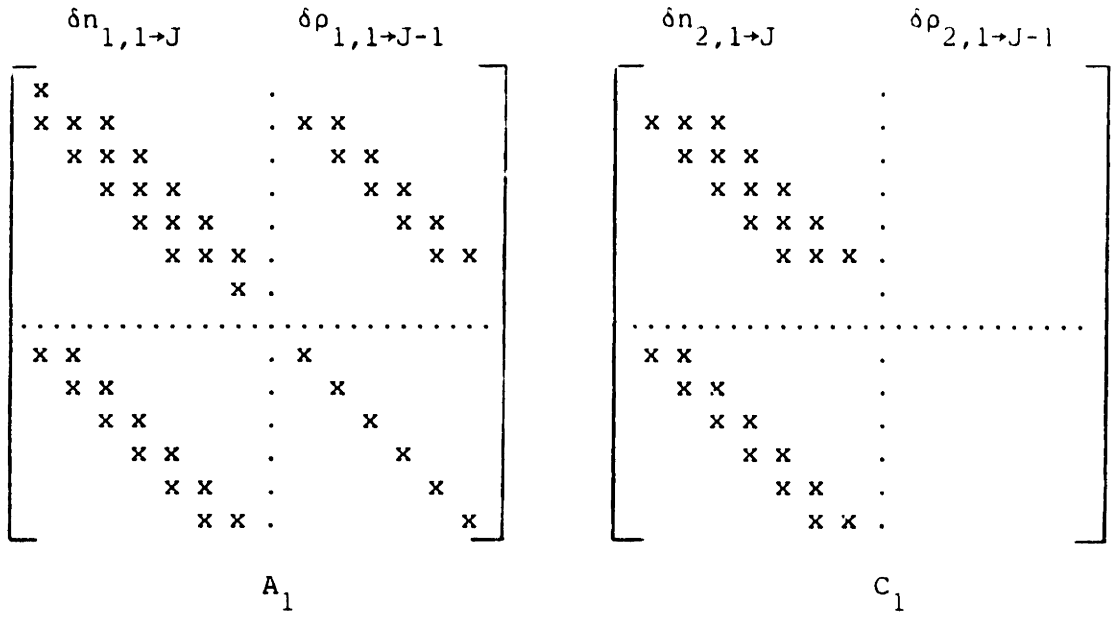


Figure 5.4: Structure of matrices A_1 and C_1.

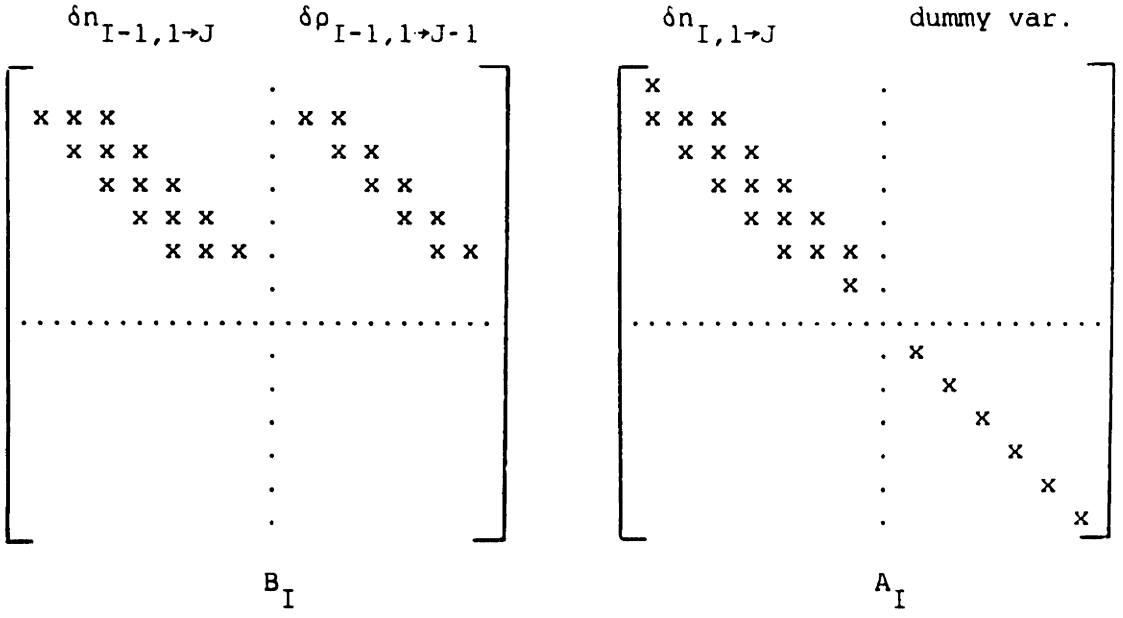


Figure 5.5: Structure of matrices B_I and A_I.

5.2 Block tridiagonal solution

The solution of this set of modified block tridiagonal equations is straightforward using a modification of well-established methods [1,8]. At the block level the algorithm is

Forward sweep

$i = 1$ to I

$$B'_i = B_i - Z_i C'_{i-2} \quad , \quad R'_i = R_i - Z_i R'''_{i-2} \qquad \text{eliminate } Z_i$$

$$A'_i = A_i - B'_i C'_{i-1} \quad , \quad R''_i = R'_i - B'_i R'''_{i-1} \qquad \text{eliminate } B'_i$$

$$C'_i = A'^{-1}_i C_i \quad , \quad R'''_i = A'^{-1}_i R''_i \qquad \text{normalize row}$$

Backward sweep

$i = I$ to 1

$$D_i = R'''_i - C'_i D_{i+1}$$

The above algorithm assumes that all matrices which have not otherwise been defined, such as $C_{-1}, C_0, Z_1, B_1, Z_2, C_I$, are zero.

Two things are done to improve the computational efficiency. The inverses of A are never explicitly calculated. Instead $A'^{-1}C$ and $A'^{-1}R''$ are calculated directly by Gaussian elimination, taking full advantage of the limited remaining sparseness in A' (see the program listing for details). Secondly, in calculating all matrix products such as $Z_i C'_{i-2}$, full advantage is taken of any sparseness in either matrix.

6. MODIFIED DIRECT METHOD FOR CHOKED FLOW

6.1 Boundary conditions

The boundary conditions presented in Section 2.2 need to be modified for choked transonic flow. For a single streamtube a well-posed choice of boundary conditions is to specify the mass flow and stagnation enthalpy, which remain constant along the streamtube, and the static pressure at the outlet. For the two-dimensional problem with coupled streamtubes, the choice of boundary conditions is to specify ,as usual, the mass flux and stagnation enthalpy in each streamtube, and in addition specify the difference in inlet stagnation densities between neighboring streamtubes, and the mass-averaged outlet stagnation density. This choice was based on two observations.

Firstly, the Crocco theorem [19,25] states that for steady flow

$$T\nabla s + \vec{q}\times\vec{\omega} = \nabla h_t \qquad (6.1)$$

which means that specifying the normal gradient of stagnation density and enthalpy at the inlet, which in turn gives the normal gradient of entropy, is equivalent to specifying the inlet vorticity, $\vec{\omega}$. Since vorticity is convected downstream by the flow, it clearly needs to be specified at the inlet, so the choice of boundary conditions satisfies this requirement. In all of the test cases the inlet flow is assumed to be irrotational and have uniform stagnation enthalpy, so the normal gradient of stagnation density is set equal to zero.

Secondly, the small perturbation potential equation for linearized irrotational perturbations to uniform flow is

$$\beta^2 \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \qquad , \qquad \beta^2 = 1-M_\infty^2 \quad . \qquad (6.2)$$

To investigate the effect of outlet boundary conditions for a duct problem, consider the solution of (6.2) on the domain $0<y<\pi$, $x<0$, with the boundary conditions

$$\frac{\partial \phi}{\partial y}(x,0) = \frac{\partial \phi}{\partial y}(x,\pi) = 0 \quad , \quad \phi(0,y) = f(y) \quad , \quad \phi \text{ bounded as } x \to -\infty \tag{6.3}$$

The solution can be written as a sum of Fourier modes,

$$\phi(x,y) = \sum_{n=0}^{\infty} F_n \cos(ny)\exp[n\beta x] \tag{6.4}$$

where

$$F_n = \begin{cases} \frac{1}{\pi}\int_0^{\pi} f(y)\, dy & , \quad n=0 \\ \\ \frac{2}{\pi}\int_0^{\pi} f(y)\cos(ny)\, dy & , \quad n>0 \end{cases} \tag{6.5}$$

The important point is that in the limit $x \to -\infty$, $\phi \to F_0$, which is the average value of ϕ at x=0. The effects of all the other Fourier modes, corresponding to n>0, decay exponentially away from the downstream boundary. Thus far upstream of the outlet boundary the flow only "sees" the average of the values specified at the outlet boundary. This suggests specifying the average stagnation density at the outflow boundary, since the average stagnation density is no longer specified at the inlet boundary.

A very similar analysis is valid for turbomachinery cascades, and demonstrates that for such flow problems the far-field boundaries need be as little as one chord length upstream and downstream from the cascade, because of the exponential decay in the far-field. This contrasts with the 1/r decay in the farfield of isolated airfoils, which requires that the far-field boundary be placed much further away in order to obtain accurate solutions.

Having justified the choice of boundary conditions, the numerical implementation of these conditions will now be described. The modification to the inlet boundary condition is achieved very simply. The old

boundary condition is specified inlet stagnation density, and the linearized form of this is equation (4.26). The new condition is a specified difference, usually zero, in the stagnation density of neighboring streamtubes. The linearized form of this is obtained by simply subtracting (4.26) for streamtube j from (4.26) for streamtube j+1.

$$\left[\frac{\rho_t}{\rho_1}\delta\rho_1 + \frac{q_1\rho_t}{(\gamma-1)(h_t-q_1^2/2)}\delta q_1\right]_{j+1} - \left[\frac{\rho_t}{\rho_1}\delta\rho_1 + \frac{q_1\rho_t}{(\gamma-1)(h_t-q_1^2/2)}\delta q_1\right]_j$$

$$= (\Delta\rho_t)_{specified} - [(\rho_t)_{j+1}-(\rho_t)_j] \qquad (6.6)$$

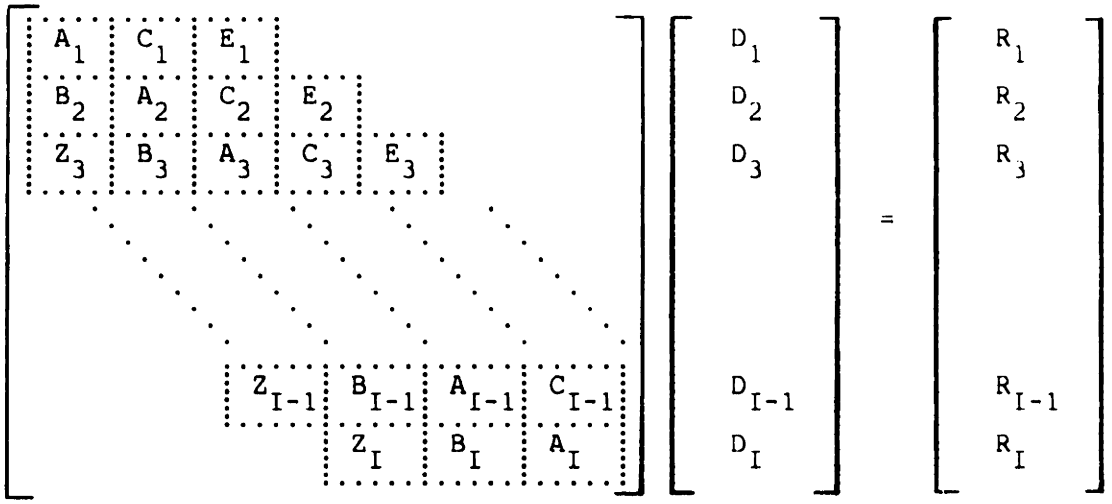
The outlet condition is specified mass-averaged stagnation density which, when linearized, is

$$\sum_{j=1}^{J-1}\frac{m_j\rho_{t_j}}{m_{tot}}\left[\frac{\delta\rho}{\rho} + \frac{q\,\delta q}{(\gamma-1)(h_t-q^2/2)}\right]_{I-1,j} = \rho_{t_{spec.}} - \sum_{j=1}^{J-1}\frac{m_j\rho_{t_j}}{m_{tot}} \qquad (6.7)$$

Note that the overall number of boundary condition equations has not altered, since the J equations specifying the inlet stagnation densities have been replaced by J-1 equations like (6.6) specifying the difference in inlet stagnation density between neighboring pairs of streamtubes, and one equation (6.7) at the outlet. This "shift" of one equation from the inlet boundary conditions to the outlet boundary conditions requires that the block tridiagonal solution procedure be altered as explained in the next section.

6.2 Solution procedure

The modified block tridiagonal form into which the linearized equations are assembled is



(6.8)

The vectors D are exactly the same as defined in Section 5.1, and the matrices Z,B,A,C are only slightly different from the matrices defined in Section 5.1. Z_I,B_I and A_I are exactly the same, and for i from 2 to I-1 the only difference is that the $J+j_{shift}{}^{th}$ row of Z_i,B_i,A_i and C_i ,which corresponds to the streamwise momentum equation for the $j_{shift}{}^{th}$ streamtube, is shifted up by one block to become the $J+j_{shift}{}^{th}$ row of B_{i-1},A_{i-1},$C_{i-1}$and E_{i-1}, as indicated in Figure 6.1. j_{shift} is chosen to correspond to the streamtube with the largest mass flow. The $J+j_{shift}{}^{th}$ row of Z_{I-1},B_{I-1},$A_{I-1}$and C_{I-1} now becomes the outlet static pressure equation (6.7). This leaves only the first block row. Rows 1 to J are unchanged, and row $J+j_{shift}$ is the row which has been shifted up from the i=2 blocks. This leaves rows J+1 to $J+j_{shift}-1$ which are equation (6.6) for j from 1 to $j_{shift}-1$, and rows $J+j_{shift}+1$ to 2J-1 which are equation (6.6) for j from j_{shift} to J.

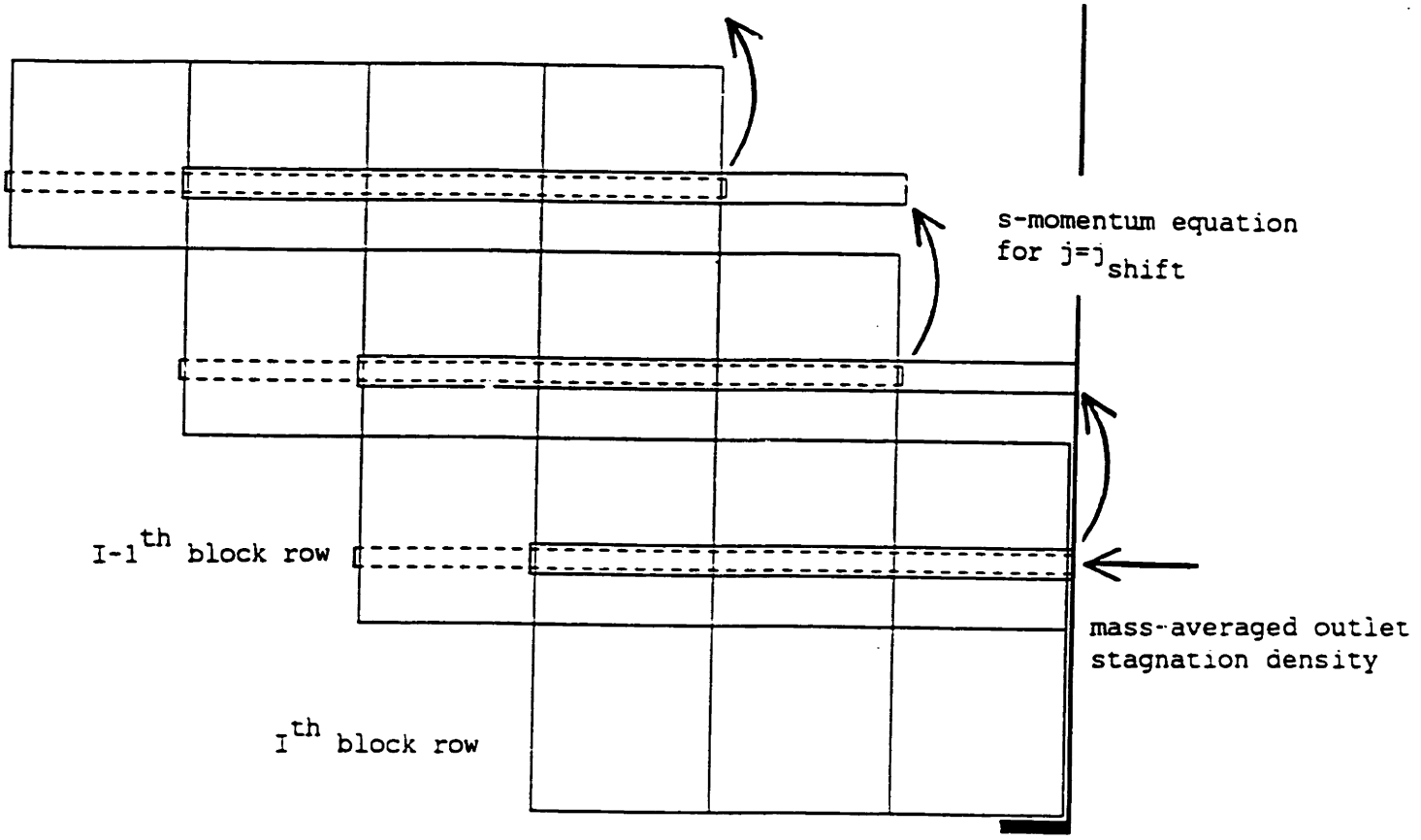


Figure 6.1: Shift of rows in choked-flow equations.

The block solution method outlined in Section 5.2 needs to be slightly modified because of the new E matrices.

Forward sweep

$$i = 1 \text{ to } I \qquad B_i' = B_i - Z_i C_{i-2}'' \;,\quad A_i' = A_i - Z_i E_{i-2}' \;,\quad R_i' = R_i - Z_i R_{i-2}'''$$

$$A_i'' = A_i' - B_i' C_{i-1}'' \;,\quad C_i' = C_i - B_i' E_{i-1}' \;,\quad R_i'' = R_i' - B_i' R_{i-1}'''$$

$$C_i'' = A_i''^{-1} C_i' \;,\quad E_i' = A_i''^{-1} E_i \;,\quad R_i''' = A_i''^{-1} R_i''$$

Backward sweep

$$i = I \text{ to } 1 \qquad D_i = R_i''' - C_i'' D_{i+1} - E_i' D_{i+2}$$

Since E has only two non-zero entries, E' has only two full columns and so, with some careful programming, this algorithm involves very little additional computational work compared to the procedure for unchoked transonic flows outlined in the last chapter. Full details can be obtained from the program listing.

7. ITERATIVE METHOD FOR SUBSONIC FLOW

Chapters 5 and 6 presented direct methods for solving the linear system of Newton equations. The alternative approach is an iterative method in which the calculated solution converges towards the exact solution, and becomes equal to it, to within machine accuracy, after an unknown number of iterations. This is the approach which has been followed by Jespersen [18], Childs and Pulliam [7] and Mulder and Van Leer [20].

The reason that iterative methods are employed, despite the great increase in programming complexity, is that they offer large potential savings in computational costs for large problems. The direct methods in Chapters 5 and 6 invert O(I) matrices, each of which is O(J) in size. This requires a total of $O(J^3I)$ operations to solve each set of Newton equations. Iterative methods by contrast require only O(JI) per iteration, and provided the number of iterations needed to solve the Newton equations to machine accuracy is independent of I and J then the total computational cost is O(JI). Thus for sufficiently large J, iterative methods will be more efficient than the direct method.

There are three points one can criticize in the above line of reasoning which justifies the use of iterative methods. The first is the assumption that an iterative method can be constructed. For each new application a considerable amount of effort may need to be spent to devise an efficient iterative method, whereas the direct solution method can be applied to any system of equations. The second is the assumption that the number of iterations to solve the Newton equations is independent of I and J. For elliptic problems, such as the classical Poisson equation, iterative methods such as Gauss-Seidel require $O(\max(I^2,J^2))$ iterations, while optimized SOR (Successive OverRelaxation) requires O(max(I,J)) iterations. To obtain an iterative method which is truly independent of I and J one must use a multigrid method. Multigrid was pioneered by Brandt [5] and has been successfully applied to a range of

problems, including the transonic full potential equation. The difficulty with multigrid methods is that they can become very complex, involving a considerable amount of development effort, especially if the computational domain is not very simple. The final point is the value of J at which the iterative methods become more efficient than the direct method. If the direct method requires aJ^3I operations, and the iterative method requires bJI operations, then the two are equal when $J=\sqrt{(b/a)}$. If b>>a then this value may be so large that for grids of practical interest, giving excellent accuracy, the direct method is more efficient than the iterative method.

In order to examine these issues this chapter presents an iterative method for solving the Newton equations for subsonic flow. A preconditioning of the Newton equations is employed to effectively separate the convective entropy equation from the elliptic pressure equation, so that suitable iterative methods can be applied to each half of the problem. This works well for subsonic flows but does not work for transonic flows because of the strong coupling between the entropy and pressure equations. However the subsonic test cases are sufficient to demonstrate that the iterative procedure is not any faster than the direct procedure for practical grids with good resolution. Thus no additional work has been performed on extending the method to handle transonic flow.

7.1 Preconditioning

Preconditioning of a system of linear equations, $Ax=b$, in general means converting the equations into a different, but equivalent, system of equations, $A'x'=b'$, where $x'=Cx$, $b'=Bb$ and $A'=BAC^{-1}$. The purpose of preconditioning for iterative solution procedures is to obtain a matrix A' which has certain desirable features, such as having very small off-diagonal elements. Such features accelerate the rate of convergence of the approximate solution towards the true solution.

The difficulty with preconditioning is that there is no general theory to guide the choice of B and C for a particular problem. This is especially true for the current problem being considered, since the linearized steady-state Euler equations implicitly contain two types of behavior. Firstly, there is the convection downstream of entropy, or vorticity (the two are equivalent for isoenergetic flow), which is a hyperbolic equation. Secondly, there is the two-dimensional pressure equation, which is elliptic for subsonic flow and hyperbolic for supersonic flow. It would be desirable to precondition the system so that these two different parts are clearly separated, and so iterative methods, suitable to each half, could be applied separately.

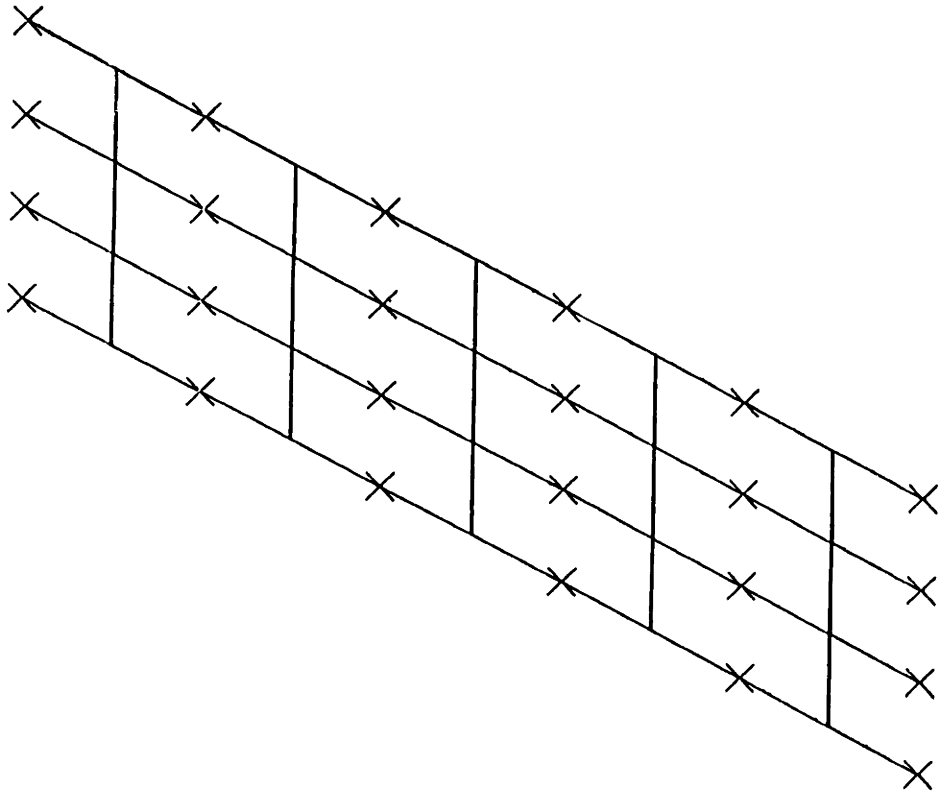


Figure 7.1: Regular sheared grid for perturbation analysis.

To find the correct form of preconditioning, it is helpful to perform a small perturbation analysis of linearized perturbations about a two-dimensional uniform flow with a regular sheared grid, as shown in Figure 7.1. In this case the streamwise momentum equation (4.11) reduces to

$$q^2\delta\rho_1 + 2\rho q\delta q_1 + \frac{\rho q^2}{A}\delta A_1 + \delta p_1 = q^2\delta\rho_2 + 2\rho q\delta q_2 + \frac{\rho q^2}{\Delta y}\delta A_2 + \delta p_2 \quad . \tag{7.1}$$

The enthalpy equation (2.12) gives

$$\delta p_i = -\frac{\gamma-1}{\gamma}\rho q\,\delta q_i + p\,\delta\rho_i \qquad , \quad i=1,2 \tag{7.2}$$

and the mass equation for subsonic flow (2.9) gives

$$\delta q_i = -\frac{q}{\rho}\delta\rho_i - \frac{1}{A}\delta A_i \qquad , \quad i=1,2 \quad . \tag{7.3}$$

When these are substituted into (7.1), the resultant equation after some simplification is

$$\delta\rho_1 - \frac{M^2}{1-M^2}\frac{\rho}{A}\delta A_1 = \delta\rho_2 - \frac{M^2}{1-M^2}\frac{\rho}{A}\delta A_2 \tag{7.4}$$

The significance of this equation is that the isentropic variation of density due to area changes is,

$$\left(\frac{\partial\rho}{\partial A}\right)_{s=const} = \frac{M^2}{1-M^2}\frac{\rho}{A} \tag{7.5}$$

so that by writing

$$\delta\rho = \left(\frac{\partial\rho}{\partial A}\right)_{s=const}\delta A + \left(\frac{\partial\rho}{\partial s}\right)_{A=const}\delta s \tag{7.6}$$

equation (7.4) becomes simply

$$\delta s_1 = \delta s_2 \quad . \tag{7.7}$$

which is a statement that entropy is convected along the streamtube.

This analysis suggests the following preconditioning. Define the variable $\delta\bar{\rho}$ as the variation in density due to a change in entropy.

$$\delta\rho_1 = \frac{M^2}{1-M^2}\frac{\rho_1}{A_1}\delta A_1 + \delta\bar{\rho}_1$$

$$= \frac{M^2}{1-M^2}\frac{\rho_1}{A_1}\left[\frac{\partial A_1}{\partial n_1^+}\delta n_1^+ + \frac{\partial A_1}{\partial n_1^-}\delta n_1^- + \frac{\partial A_1}{\partial n_2^+}\delta n_2^+ + \frac{\partial A_1}{\partial n_2^-}\delta n_2^-\right] + \delta\bar{\rho}_1 \qquad (7.8)$$

When this equation is substituted into all of the Newton equations, a system of equations for δn and $\delta\bar{\rho}$ is obtained, which has the property that for uniform flows the streamwise momentum equations contains only $\delta\bar{\rho}$ terms, and more generally for non-uniform flows the coefficients of the δn terms are extremely small. Note that once δn and $\delta\bar{\rho}$ have been calculated $\delta\rho$ is obtained from equation (7.8).

7.2 Solution procedure

With the preconditioning the Newton equations may be expressed symbolically in two halves.

$$A_{\rho\rho}\delta\overline{\rho} + A_{\rho n}\delta n = d_{\rho} \tag{7.9a}$$

$$A_{n\rho}\delta\overline{\rho} + A_{nn}\delta n = d_{n} \tag{7.9b}$$

The first half (7.9a) corresponds to the streamwise momentum equation, with the preconditioning ensuring that $A_{\rho n}$ is very small. The second half, (7.9b), corresponds to the normal momentum equation, with both $A_{n\rho}$ and A_{nn} being in general of the same magnitude. For the purposes of choosing an iterative solution technique, the interesting feature is that $A_{\rho\rho}$ is a discrete convective operator with no entries above the main diagonal, which means that if δn were known then $\delta\overline{\rho}$ could be calculated immediately. A_{nn} however is a discrete elliptic operator which requires some form of classic elliptic relaxation procedure.

The iterative algorithm which is therefore employed has two parts:

a) First assume δn is fixed, and solve equation (7.9a) to obtain a new value for $\delta\overline{\rho}$,

b) Then assume $\delta\overline{\rho}$ is fixed and perform several cycles of an SLOR (Successive Line Over-Relaxation) procedure to obtain an approximate solution for δn.

This algorithm is then repeated iteratively until $\delta\overline{\rho}$ and δn converge to the true solution. More complete details can be obtained from the program listing. The convergence rate is dominated by the relaxation procedure in part b), which can be slow for large problems. One way of improving it would be to use multigrid acceleration, which was first proposed by Brandt [5], and has been used by Jespersen and Childs [7], Pulliam [23] and Mulder and Leer [20]. However this would still not increase the speed sufficiently to change the conclusion drawn in the next chapter that iterative methods are not faster than direct

methods, for two-dimensional problems with sufficient grid resolution to obtain good solutions. For this reason also, little effort was made to extend this algorithm to handle transonic flows, in which the entropy and pressure equations do not uncouple, due to the presence of shocks and artificial compressibility.

8. GLOBAL VARIABLES AND EQUATIONS

8.1 Concept and Solution Procedure

For the duct problem considered so far each variable and equation is local in the sense that each variable is associated with a particular node and each equation involves only variables in a localized region of the flow. In other flow problems there are sometimes a few global variables which appear in many equations and a few global equations which involve variables at many nodes. In the case of the duct problem an example would be allowing the total mass flux to vary in order to achieve a specified average inlet Mach number. The total mass flux would be a global variable because through the linearized mass equation the change in the mass would enter into every one of the momentum equations. The equation specifying the inlet Mach number would be a global equation because it involves the inlet variables of all of the streamtubes. With this additional variable and equation the linearized equation set (5.1) becomes,

$$\begin{bmatrix} A_1 & C_1 & & & & & & x \\ B_2 & A_2 & C_2 & & & & & x \\ Z_3 & B_3 & A_3 & C_3 & & & & x \\ & \ddots & \ddots & \ddots & \ddots & & & x \\ & & \ddots & \ddots & \ddots & \ddots & & x \\ & & & \ddots & \ddots & \ddots & \ddots & x \\ & & & & Z_I & B_I & A_I & x \\ x & x & x & x & x & x & x & x \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \\ \\ \\ D_I \\ \delta m_{total} \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \\ \\ \\ R_I \\ R_m \end{bmatrix} \qquad (8.1)$$

In this equation the total mass variable and the inlet Mach number equation are really just an additional Newton variable and equation. It is just their span, in terms of how many equations and variables they involve, which makes them different from the other variables and equa-

tions. This difference becomes important only when solving the linear system of equations (8.1), since the algorithm detailed in Section 5.2 now needs modification. The simplest way to solve (8.1) is to rearrange the equations into the following form.

$$\begin{bmatrix} A_1 & C_1 & & & & \\ B_2 & A_2 & C_2 & & & \\ Z_3 & B_3 & A_3 & C_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & Z_I & B_I & A_I & \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \\ \\ D_I \end{bmatrix} = \begin{bmatrix} R_1^1 \\ R_2^1 \\ R_3^1 \\ \\ \\ R_I^1 \end{bmatrix} + \delta m_{total} \begin{bmatrix} R_1^2 \\ R_2^2 \\ R_3^2 \\ \\ \\ R_I^2 \end{bmatrix} \qquad (8.2)$$

and

$$(x \quad x \quad x \quad x \quad x \quad x \quad x \quad x) \begin{bmatrix} D_1 \\ D_2 \\ . \\ . \\ D_I \\ \delta m_{total} \end{bmatrix} = R_m \; . \qquad (8.3)$$

Equation (8.2) is solved to obtain an equation of the form

$$\begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \\ \\ D_I \end{bmatrix} = \begin{bmatrix} D_1^1 \\ D_2^1 \\ D_3^1 \\ \\ \\ D_I^1 \end{bmatrix} + \delta m_{total} \begin{bmatrix} D_1^2 \\ D_2^2 \\ D_3^2 \\ \\ \\ D_I^2 \end{bmatrix} \; . \qquad (8.4)$$

This is accomplished by essentially the same algorithm explained

in Section 5.2.

Forward sweep

$i = 1$ to I

$$B_i' = B_i - Z_i C_{i-2}' \quad , \quad R_i^{1'} = R_i^1 - Z_i R_{i-2}^{1'''} \quad , \quad R_i^{2'} = R_i^2 - Z_i R_{i-2}^{2'''}$$

$$A_i' = A_i - B_i' C_{i-1}' \quad , \quad R_i^{1''} = R_i^{1'} - B_i R_{i-1}^{1'''} \quad , \quad R_i^{2''} = R_i^{2'} - B_i R_{i-1}^{2'''}$$

$$C_i' = (A_i')^{-1} C_i \quad , \quad R_i^{1'''} = (A_i')^{-1} R_i^{1''} \quad , \quad R_i^{2'''} = (A_i')^{-1} R_i^{2''}$$

Backward sweep

$$i = I \text{ to } 1 \qquad D_i^1 = R_i^{1'''} - C_i' D_{i+1}^1 \quad , \quad D_i^2 = R_i^{2'''} - C_i' D_{i+1}^2$$

Once equation (8.4) is obtained, it is substituted into equation (8.3) to eliminate the D vectors and obtain a simple scalar equation of the form

$$a\ \delta m_{total} = b \tag{8.5}$$

from which δm_{total} is calculated. With δm_{total} now known equation (8.4) gives the final value of the D vectors and the density and node positions are updated.

The above example had only one global variable and equation. In general there may be more than one global variable, with the corresponding number of global equations. If there are M global variables then an equation of similar form to (8.2) will be solved by the same method to obtain an equation like (8.4).

$$\begin{bmatrix} D_1 \\ D_2 \\ \cdot \\ \cdot \\ D_I \end{bmatrix} = \begin{bmatrix} D_1^1 \\ D_2^1 \\ \\ \\ D_I^1 \end{bmatrix} + \delta_1 \begin{bmatrix} D_1^2 \\ D_2^2 \\ \\ \\ D_I^2 \end{bmatrix} + \delta_2 \begin{bmatrix} D_1^3 \\ D_2^3 \\ \\ \\ D_I^3 \end{bmatrix} + \cdots + \delta_M \begin{bmatrix} D_1^{M+1} \\ D_2^{M+1} \\ \\ \\ D_I^{M+1} \end{bmatrix} \tag{8.6}$$

When this is substituted into the M linearized global equations a system of M equations is obtained from which the M unknowns, $\delta_1, \delta_2 .. \delta_M$ are calculated. Then equation (8.6) above gives the final linearized corrections with which to update the density and grid node positions.

8.2 Cascade Boundary Conditions

In this thesis only duct geometries have been considered until this moment. The solution of flow over turbomachinery cascades or isolated airfoils requires several changes to the boundary conditions presented in Section 2.2. Full details are presented in the Ph.D. thesis of M. Drela [11] but for completeness the boundary conditions for the cascade problem are also presented here. They illustrate the use of global unknowns and equations, and the role of the Kutta condition in determining the circulation around each blade.

The first change to the boundary conditions is the periodicity conditions which are imposed across the stagnation streamline, upstream and downstream of the blade. The two conditions are

$$x_{i,J} = x_{i,1} \quad , \quad y_{i,J} = y_{i,1} + \text{Pitch} \tag{8.8a,b}$$

$$\Pi^{+}_{i,J} = \Pi^{-}_{i,1} \tag{8.9}$$

The first condition states that the J^{th} streamline is identical to the first streamline except that it is displaced by the pitch (the blade-to-blade distance) in the y-direction. The second condition is that the pressures match across the stagnation streamline, which is identical to the matching conditions across all of the other streamlines. Thus away from the blades the stagnation streamlines are treated exactly the same as any other streamline. When linearized equations (8.8a,b) reduce to a statement that the δn for the two streamlines are equal, and the linearization of (8.9) produces an equation analogous to (4.16) which is then incorporated into an n-momentum equation spanning across the stagnation streamline.

These conditions are applied to all of the grid nodes on the stagnation streamline which do not lie on the blades. They are also applied to the leading edge stagnation point for blades with blunt leading edges with the modification that $\hat{n}$, the direction in which the grid node is

constrained to move, is directed along the surface of the blade, instead of normal to the streamline as it is usually defined.

The inlet and outlet boundary conditions are identical to those in Sections 2.2 and 4.2 with the exception that $\delta n_{1,1}$ and $\delta n_{I,1}$, the movement of the nodes at the ends of the stagnation streamline, are no longer set equal to zero, but are instead set equal to two global variables δn_{inlet} and δn_{outlet}. Corresponding to these two varaiables there need to be two global equations. The first is usually chosen to be a specified inlet flow angle (defined in a mass-averaged sense) but can alternatively be chosen to be a specified lift on each blade element, or any other suitable condition. The second is chosen to be the Kutta condition which states that the pressures on the two sides of the trailing edge should be equal. This condition is based on experimental evidence that this is the condition that determines the circulation around airfoils for attached flows. Normally this condition is not applied in numerical Euler calculations performed using time-marching methods. It is believed that the numerical viscosity which is used in such methods mimics, in some fashion, the physical viscous mechanisms which produce the Kutta condition. In the present method there is no such mechanism and so the Kutta condition must be explicitly enforced. In this regard the present method is identical to Potential methods and to the method of Wu [31] for solving the Euler equations, in which there is no artificial viscosity and the circulation is determined by applying the Kutta condition.

In the present method the Kutta condition is applied by requiring

$$\Pi^{+}_{i_{t.e.},J} = \Pi^{-}_{i_{t.e.},1} \tag{8.10}$$

When linearized, this is incorporated into an n-momentum equation, as usual, and it is this equation which is used as the global equation.

One advantageous consequence of the solution procedure involving

global variables and global equations is that one obtains the linear sensitivity of the flow field to variations in the global parameters. For example, in the above case the solution vectors D_i^2 and D_i^3 obtained during the numerical procedure represent the change in the flow field due to unit changes in δn_{inlet} and δn_{outlet}. Through the global equations, δn_{inlet} and δn_{outlet} can in turn be related to the change in the inlet flow angle α_{inlet}. Hence one can find the linear variation of the entire flow field due to variations in α_{inlet}. In particular one can obtain $\frac{\partial C_l}{\partial \alpha}$, the variation in the lift coefficient with respect to the inlet flow angle, or other important quantities of engineering interest.

The boundary conditions for the flow over an isolated airfoil are handled in a very similar manner. The far-field in represented as the combination of a flow at an angle of attack together with the far-field of a compressible vortex. The values of the angle of attack and the vortex strength are the two global variables, and the two global equations are the Kutta condition together with either specifed angle of attack (which results in a trivial identity equation) or specified lift (which is a useful option when comparing to experimental results in which it is better to match the lift than the angle of attack due to uncertainties associated with wind tunnel wall corrections). The full details are available in [11].

9. RESULTS

The primary aim of the test cases is to prove that the algorithm which has been developed works. This requires showing both that the Newton iterative procedure converges to the solution of the discretized equations, and that this discrete solution is an accurate approximation to the solution of the analytic equations. The first part is achieved by presenting iteration histories, which usually consist of the maximum and r.m.s. (root mean square) values of the changes $\delta\rho$ and δn at each Newton iteration. The second part is more difficult since a simple closed form solution to the analytic equations can not usually be obtained. In this chapter four different approaches are used. In one case a closed form solution is known for an incompressible cascade flow constructed by a conformal transformation, and a comparison with the numerical solution can be made by using a inlet Mach number sufficiently low to avoid compressibility effects. In a second case a comparison is made with quasi-one-dimensional analytic theory for a choked Laval nozzle. In a third case comparison is made with experimental results for a subsonic turbine cascade. In two other cases the numerical solution is compared to numerical solutions obtained by completely different numerical method. Finally, in some other subsonic cases, the stagnation density changes are used as an indication of numerical errors, since the analytic solution for subsonic, inviscid flow has uniform stagnation density, assuming it is uniform at the inlet. This test is particularly useful for determining the order of convergence, which is the rate at which the numerical truncation errors, due to the finite grid size $\Delta x, \Delta y$, go to zero as $\Delta x, \Delta y$ are reduced to zero. Another accuracy issue is the stagnation density errors produced by the artificial compressibility in the smooth supersonic region of transonic flows, and the effectiveness of the second order corrections in reducing these errors.

A second purpose of these tests is to demonstrate the solution of the Newton equations using each of the three solution methods, the

direct method, the modified direct method for choked flows, and the iterative method for subsonic flows, and compare their relative efficiencies.

A third objective is to demonstrate the flexibility offered by the Newton approach through the use of global variables and global constraints. For example, instead of specifying the angle of attack for an airfoil problem, as one does for normal time-marching Euler methods, one can allow the angle of attack to be a global variable and prescribe the lift as a global constraint. Also it is possible to very efficiently calculate a number of solutions corresponding to different values of a particular parameter. For example, one might wish to calculate the C_l α curve, relating the lift coefficient to the angle of attack, for an airfoil for some given freestream Mach number, or alternatively, one might wish to calculate the C_d-M curve relating the inviscid drag coefficient (due to shocks) to the freestream Mach number for a given value of C_l. The C_l-α curve is important in aircraft stability analysis, while the C_d-M curve is important for determining the total inviscid drag for an aircraft as a function of the cruise Mach number.

Table 9.1 presents a list of the test cases, with a summary of the test geometry, flow conditions, basis for determining accuaracy, and principal purposes of the test.

Table 9.1: Summary of test cases

Geometry	Flow Conditions	Accuracy Criterion	Point of Interest
duct with $\sin^2(\pi x)$ bump	subsonic	stagnation density errors	order of convergence
duct with elliptic bump	subsonic	stagnation density errors	order of convergence
Gostelow cascade	subsonic	analytic solution	accuracy, iterative solver
T7 turbine	subsonic	experimental results	grid with high shear
Garabedian cascade	transonic	numerical hodograph solution	accuracy, artificial compressibility
NACA 0012	transonic	numerical test data	strong normal shocks
2-D Laval nozzle	choked transonic	Quasi-1-D Laval nozzle theory	modified direct solver for choked flow

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

The test geometry for this case is,

Inlet $x=-1.0$

Outlet $x= 2.0$

Upper wall $$y=\begin{cases} 0.5 & x<0 \ , \ x>1 \\ 0.5-0.1\sin^2(\pi x) & 0<x<1 \end{cases}$$

Lower wall $$y=\begin{cases} 0. & x<0 \ , \ x>1 \\ 0.1\sin^2(\pi x) & 0<x<1 \end{cases}$$

Figure 9.1a shows the geometry with a 61x11 grid, corresponding to a converged solution. The flow conditions are,

$$h_t=1/(\gamma-1) \ , \qquad \rho_{t_{inlet}}=1. \ , \qquad m_{tot}=0.1 \ , \qquad \gamma=1.4$$

which gives an inlet Mach number of approximately 0.20, and a maximum Mach number of approximately 0.40. Figures 9.1b and 9.1c show contours of Mach number and stagnation density changes, respectively.

The purpose of this case is to investigate the order of accuracy of the discretization of the Euler equations for a subsonic flow with no stagnation points. The accuracy can be gauged from the stagnation density changes since they are zero for the analytic solution because the flow is inviscid and subsonic, and hence isentropic and isenthalpic. There are several points to note about the stagnation density errors. The overall level of error is vey small, with the maximum being approximately 2.5e-4, and mostly the errors are positive, representing an increase in stagnation density. For this symmetric geometry the errors are also symmetric, which is to be expected since, with the exception of the inlet/outlet boundary conditions, the discrete equations are symmetric and do not know the direction of the flow for subsonic flow. For transonic flow this is no longer the case because the direction of the flow is linked to the direction of the density upwinding due to the

artificial compressibility. Finally, the maximum errors are in the region of the strongest flow curvature, which is a typical feature of this discretization applied to subsonic flows, and in general means that the maximum stagnation errors are generated in the neighbornood of leading edge stagnation points where the flow is strongly curved.

Table 9.2 lists the 'average' stagnation density errors for grids of different sizes. The 'average' error is defined by a weighted root-mean-square average.

$$E = \left[\sum_{i,j} m_j [(\rho_{t_{i,j}} - \rho_{t_{inlet}})/\rho_{t_{inlet}}]^2 \; / \; \sum_{i,j} m_j \right]^{1/2} \qquad (9.1)$$

The mass flux weighting ensures that narrow streamtubes with small mass fluxes do not contribute as much to the overall average as thicker streamtubes with larger mass fluxes.

Table 9.2: Stagnation density errors for $\sin^2(\pi x)$ bump

E	J = 11	J = 21	J = 31
I = 31	1.10e-4	1.11e-4	1.11e-4
I = 61	3.11e-5	3.15e-5	3.15e-5
I = 121	8.09e-6	8.15e-6	8.13e-6

Table 9.2 shows that the error E is approximately independent of J, and proportional to I^{-2}, so the discretization of the Euler equations is second order accurate for subsonic flows with no stagnation points.

Also of interest in this example is the rate at which the Newton iteration converges to the steady-state discrete solution. Table 9.3

presents the root-mean-square changes in the density and node positions at each iteration of the cases using the 31x11 and 121x31 grids. It can be seen that after three iterations the solution has already converged to machine accuracy, thus verifying the extremely fast quadratic convergence feature of the Newton iterative procedure. Also it is clear that the number of iterations is approximately independent of the size of the grid, which is to be expected since for a linear problem the solution would be obtained in one iteration independent of the grid size.

These calculations were performed on a Perkin-Elmer 3242 computer. The CPU time per iteration was approximately 3 secs. for the 31x11 grid and 2.2 mins. for the 121x31 grid.

Table 9.3: Newton iteration histories

Iteration number	31x11		121x31	
	$\delta\rho/\rho$	δn	$\delta\rho/\rho$	δn
1	1.01e-1	2.06e-4	1.01e-1	4.36e-4
2	2.47e-4	2.83e-5	2.54e-4	6.02e-5
3	4.99e-7	4.49e-6	5.02e-7	2.02e-6
4	4.35e-7	2.83e-6	4.39e-7	2.61e-6
5	4.81e-7	2.19e-6	4.24e-7	2.75e-6

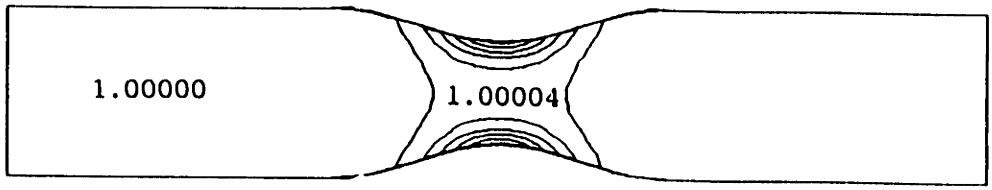


Figure 9.1c: Stagnation density contours with increments of 0.00004.

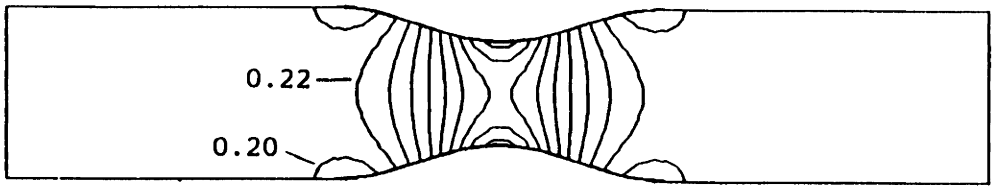


Figure 9.1b: Mach number contours with increments of 0.02.

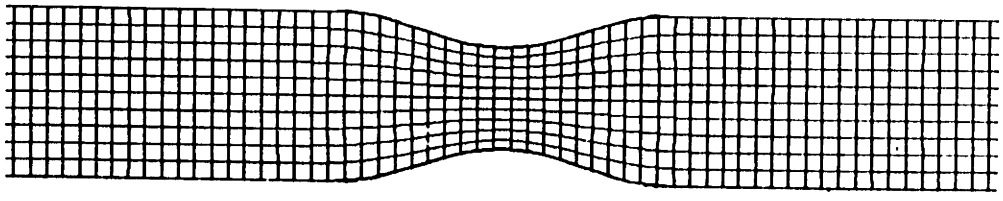


Figure 9.1a: Duct and grid geometry for test case 1: duct with $\sin^2(\pi x)$ bump.

9.2 Duct with elliptic bump

The test geometry for this case is,

Inlet $x=-1.0$

Outlet $x= 2.0$

Upper wall $$y=\begin{cases} 0.5 & x<0\ ,\ x>1 \\ 0.5-0.1[1-(x-0.5)^2]^{1/2} & 0<x<1 \end{cases}$$

Lower wall $$y=\begin{cases} 0. & x<0\ ,\ x>1 \\ 0.1[1-(x-0.5)^2]^{1/2} & 0<x<1 \end{cases}$$

Figure 9.2a shows the geometry with a 61x11 grid, corresponding to a converged solution. The flow conditions are,

$h_t=1/(\gamma-1)$, $\rho_{t_{inlet}}=1.$, $m_{tot}=0.1$, $\gamma=1.4$

which gives an inlet Mach number of approximately 0.20, and a maximum Mach number of approximately 0.70. Figures 9.2b and 9.2c show contours of Mach number and stagnation density changes, respectively.

The purpose of this case is to examine the problems associated with a stagnation point. If the mass flux in each streamtube is the same, as in the example just presented, the streamtube at the stagnation point becomes very large, as can be seen more clearly in Figure 9.3. To quantify the errors resulting from this lack of grid resolution at the stagnation point, Table 9.4 lists the average stagnation density errors for grids of different sizes, with the average errors defined as before.

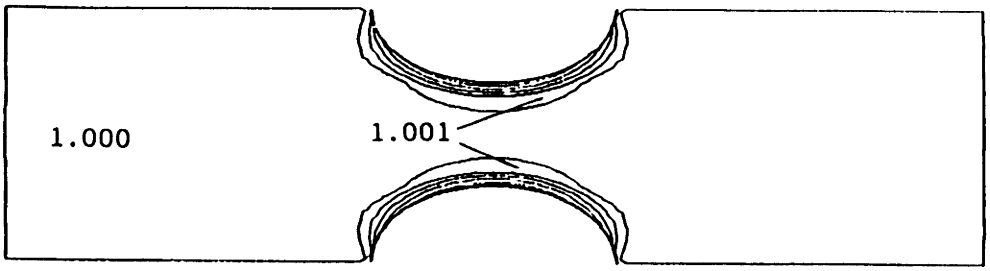


Figure 9.2c: Stagnation density contours with increments of 0.001.

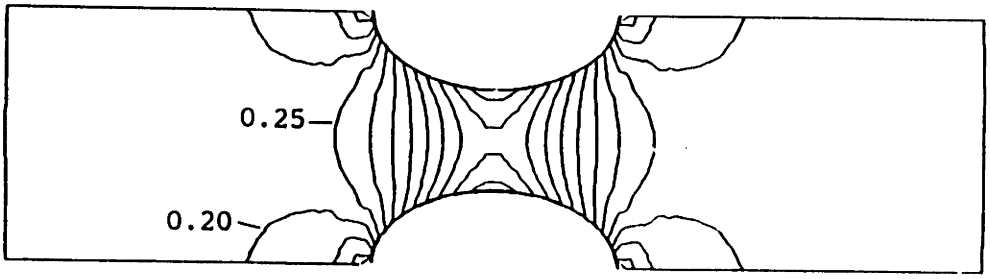


Figure 9.2b: Mach nu..ber contours with increments of 0.05.

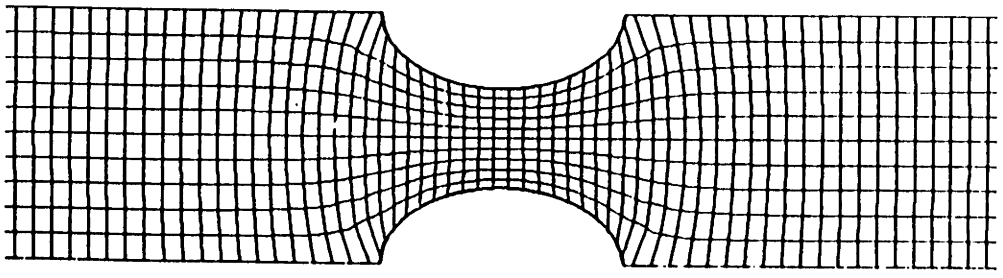


Figure 9.2a: Duct and grid geometry for test case 2: duct with elliptic bump.

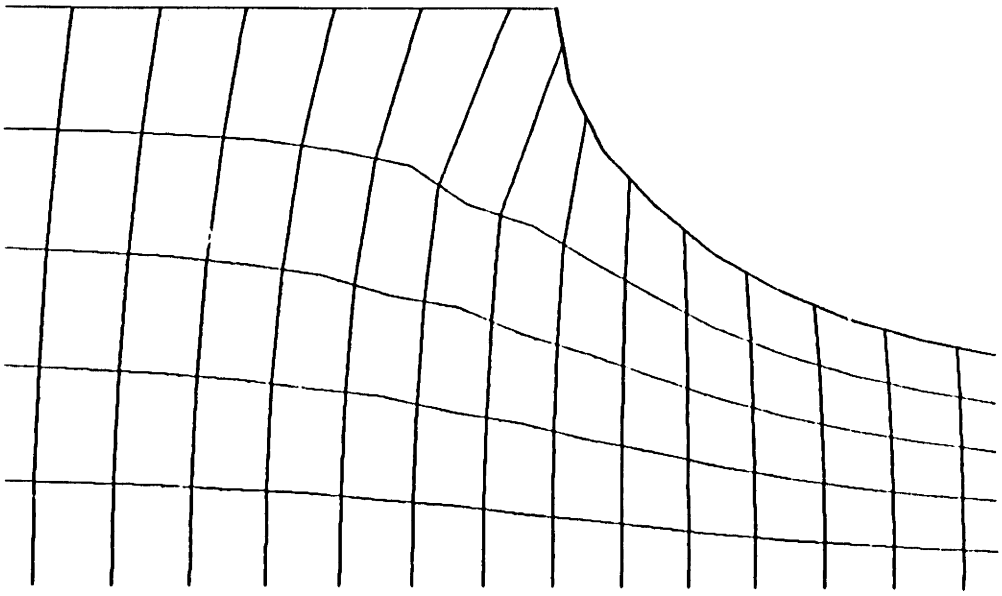


Figure 9.3: Close-up of grid near stagnation point on elliptic bump.

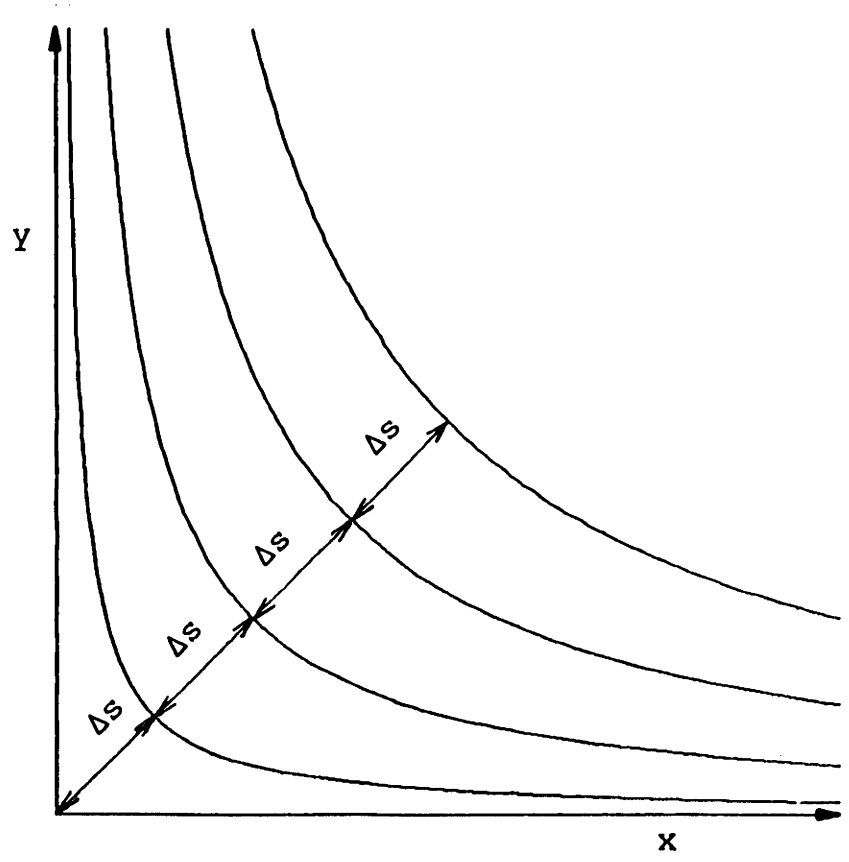


Figure 9.4: Streamlines in a stagnation point flow.

Table 9.4: Stagnation density errors for elliptic bump

		J		
	E	11	21	31
	31	2.64e-3	2.84e-3	2.94e-3
I	61	8.52e-4	8.39e-4	8.35e-4
	121	2.90e-4	2.55e-4	2.41e-4

Table 9.4 shows that, compared to the $\sin^2(\pi x)$ bump case in the last section, the error is still much more strongly dependent on I than J, but now a weak dependence on J is apparent for I=121, and also the rate of convergence, keeping I/J fixed, is now less than second order, although still more than first order. Thus the stagnation point has caused some deterioration in the order of convergence, at least when maintaining equal mass fluxes in each streamtube.

To improve this situation it is clearly desirable to use streamtubes with varying mass fluxes, so that the streamtubes near the stagnation point have less mass flux and so give better resolution of the flow in the neighborhood of the stagnation point.

The stream-function for the incompressible flow in a stagnation corner, as shown in Figure 9.4, is given by [2],

$$\Psi(x,y) = Bxy \quad , \quad B=\text{constant} \tag{9.2}$$

Now suppose that one chooses a set of streamlines that cross the line x=y (the line of maximum separation between streamlines) at equal intervals Δs, that is the j^{th} streamline passes through $(j\Delta s/\sqrt{2}, j\Delta s/\sqrt{2})$. The mass flux in the corresponding set of streamtubes is given by,

$$
\begin{aligned}
m_j &= \Psi(j\Delta s/\sqrt{2}, j\Delta s/\sqrt{2}) - \Psi((j-1)\Delta s/\sqrt{2}, (j-1)\Delta s/\sqrt{2}) \\
&= \tfrac{1}{2} B(j\Delta s)^2 - \tfrac{1}{2} B((j-1)\Delta s)^2 \\
&= B\Delta s^2 (j - \tfrac{1}{2})
\end{aligned}
\tag{9.3}
$$

This suggests that to obtain uniform resolution in the neighborhood of the stagnation point one should choose the mass fluxes to be linear in the index of the streamtube. Thus the case of the duct with the elliptic bump was redone with the mass fluxes defined by,

$$m_j = B \min(j, J-j) \tag{9.4}$$

with B chosen to match the total mass flux.

$$B = m_{tot} \Big/ \sum_{j=1}^{J-1} \min(j, J-j) \tag{9.5}$$

Figure 9.5a shows the modified 31x11 grid for a converged solution, and Figures 9.5b and 9.5c show the Mach number and stagnation density error contours. Figure 9.6 shows a close-up of the stagnation point region, from which it is clear that there is now much better resolution. Table 9.5 lists the average stagnation density errors. The results are now much more similar to the results for the $\sin^2(\pi x)$ bump, in that there is little dependence on J, and the error is approximately proportional to I^{-2}. Thus, with careful treatment of stagnation points, global second order accuracy can be maintained for subsonic flows.

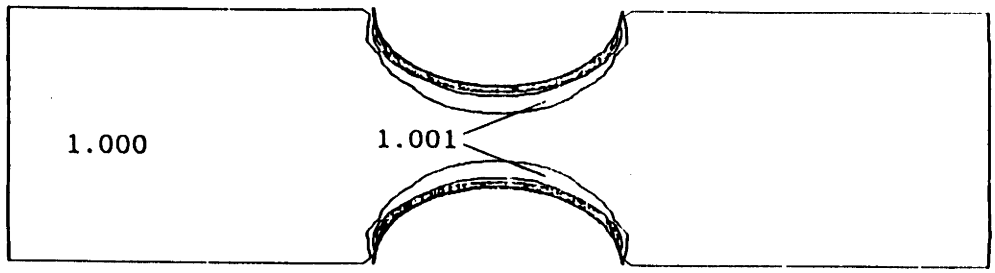


Figure 9.5c: Stagnation density contours with increments of 0.001.

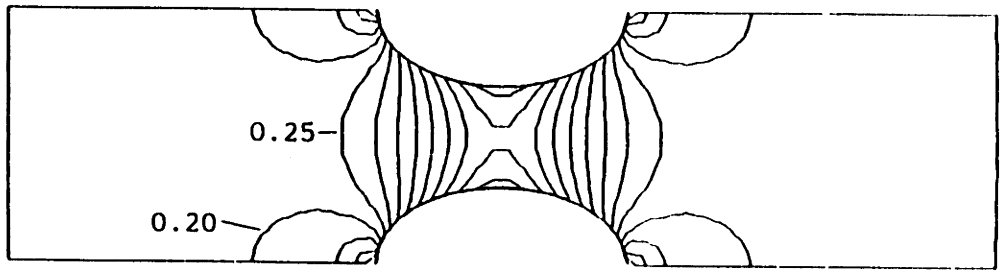


Figure 9.5b: Mach number contours with increments of 0.05.

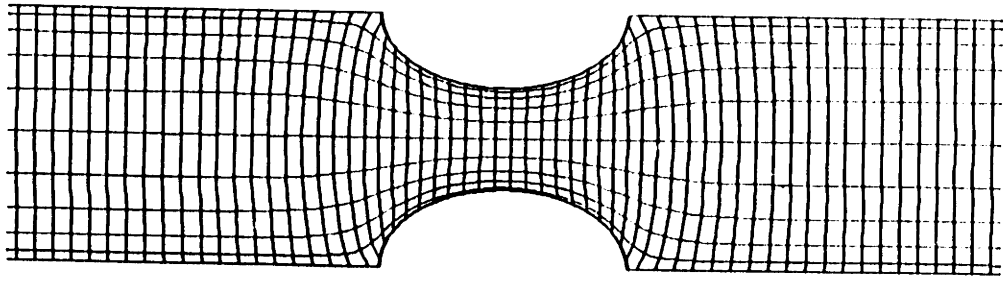


Figure 9.5a: Duct and grid geometry for test case 2: duct with elliptic bump using modified grid.

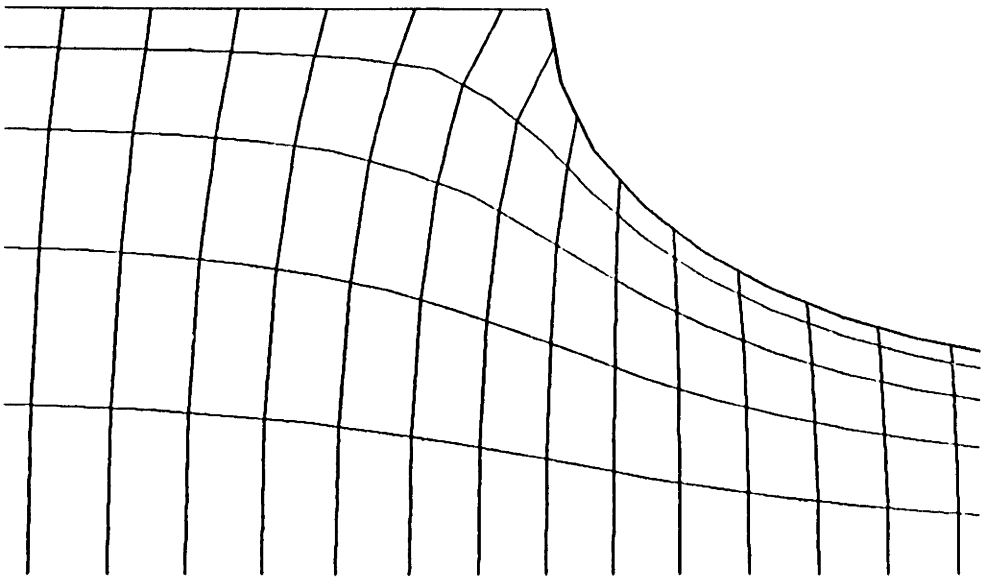


Figure 9.6: Close-up of grid near stagnation point on elliptic bump with modified grid.

Table 9.5: Stagnation density errors for elliptic bump; modified grid

		J		
	E	11	21	31
	31	2.95e-3	3.15e-3	3.18e-3
I	61	8.32e-4	8.47e-4	8.62e-4
	121	2.45e-4	2.18e-4	2.16e-4

Table 9.6 presents the Newton iteration histories for two of the solutions with the modified mass fluxes. The Newton procedure again converges to machine accuracy in three iterations, independent of the size of the grid. The CPU time per iteration was approximately 3 secs. for the 31x11 grid, and 2.2 mins. for the 121x31 grid.

Table 9.6: Newton iteration histories; modified grid

Iteration number	31x11		121x31	
	$\delta\rho/\rho$	δn	$\delta\rho/\rho$	δn
1	9.45e-2	1.60e-3	9.49e-2	4.96e-4
2	7.30e-4	2.44e-4	5.30e-4	7.11e-5
3	2.17e-5	4.92e-6	2.13e-6	5.11e-6
4	6.28e-7	1.47e-6	1.46e-6	3.79e-6
5	7.20e-7	2.06e-6	1.51e-6	2.10e-6

9.3 Incompressible Gostelow cascade

This test case is a closed-form analytic solution of incompressible flow past a cascade, which was derived by Gostelow using a conformal transformation method [15]. This is thus an excellent test case for comparison purposes provided the compressible solution is obtained for an inlet Mach number which is sufficiently low to avoid compressibility effects. Figure 9.7a shows the geometry, with a converged 122x23 grid. Figures 9.7b and 9.7c show the corresponding contours of Mach number and stagnation density errors. The maximum Mach number is approximately 0.1, so that the compressibility is negligible. Figure 9.8 shows the excellent agreement between the calculated surface pressure coefficient distribution and Gostelow's tabulated results. The inlet flow angle was specified to be the same as that used by Gostelow, 53.5^0 relative to the axial direction, but the outlet flow angle was determined through the Kutta condition applied at the trailing edge, and so is a particularly sensitive measure of the accuracy of the method. The calculated value is 30.06^0 compared to Gostelow's exact value of 30.025^0. To achieve this level of agreement required good resolution both at the leading edge, as shown in Figure 9.9a, where there are large gradients in the flow quantities, and at the trailing edge, as shown in Figure 9.9b, where there is an analytic square-root singularity in the pressure.

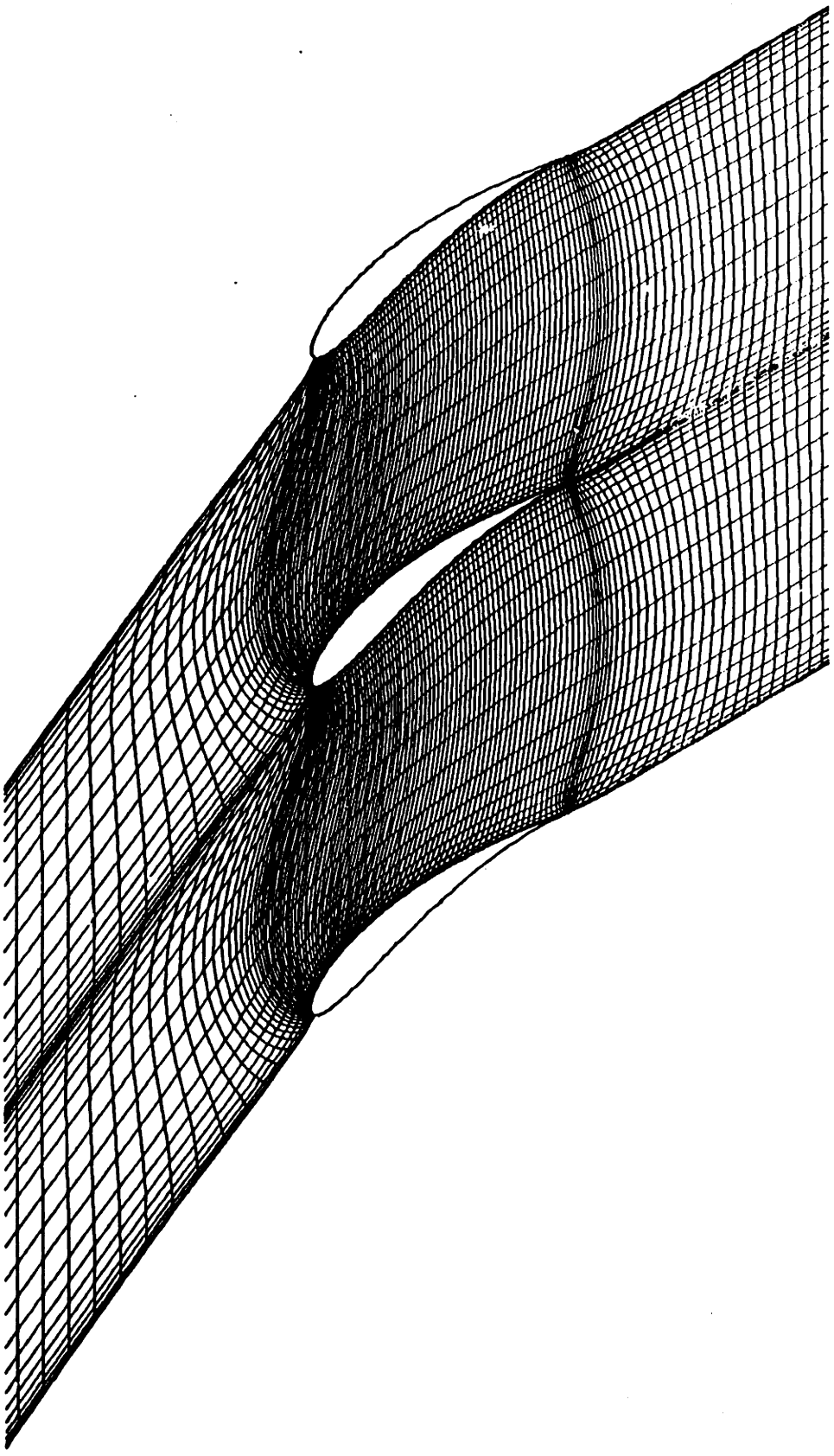


Figure 9.7a: Airfoil and grid geometry for Gostelow cascade.

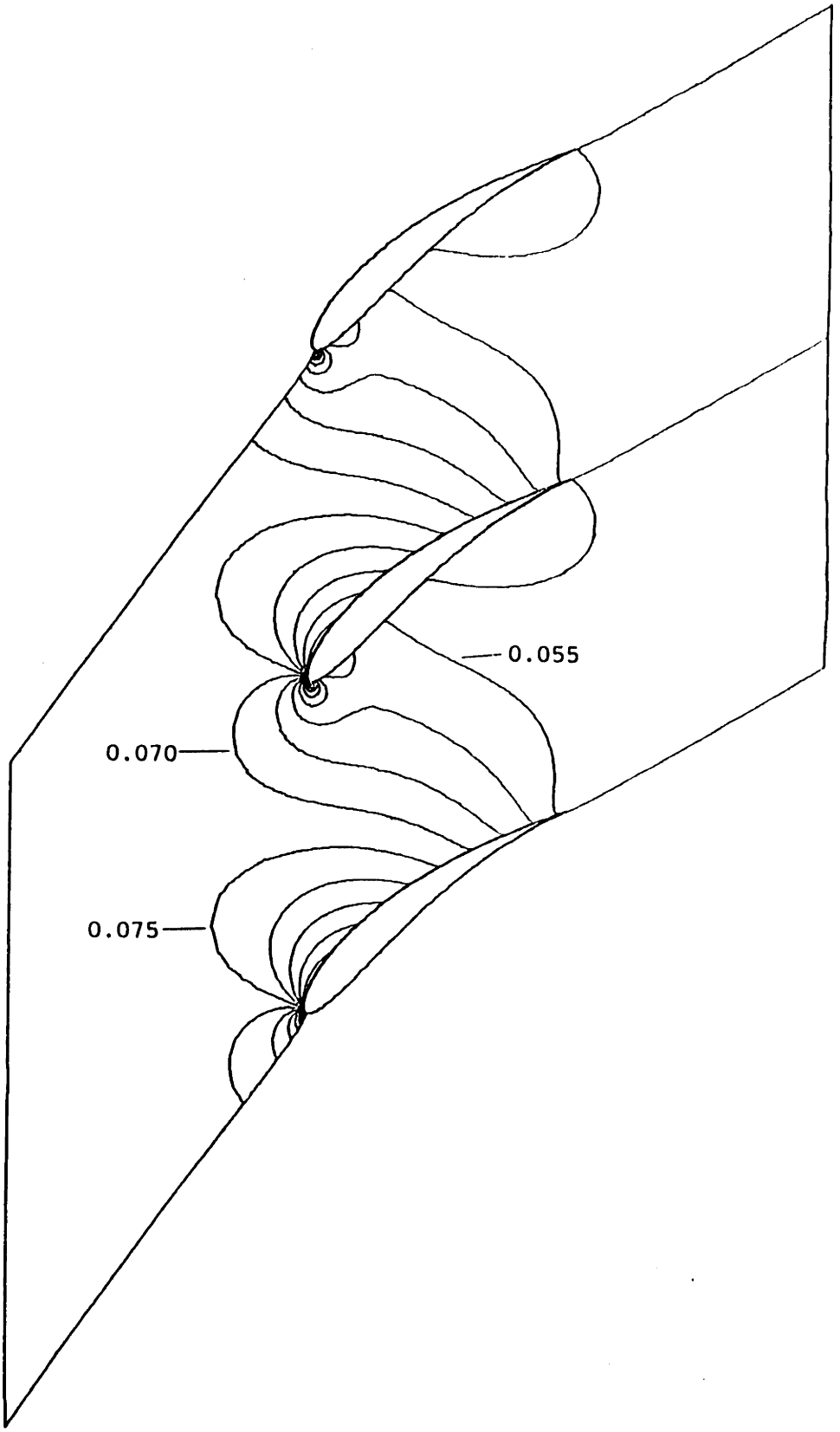


Figure 9.7b: Mach number contours for Gostelow cascade with increments of 0.005.

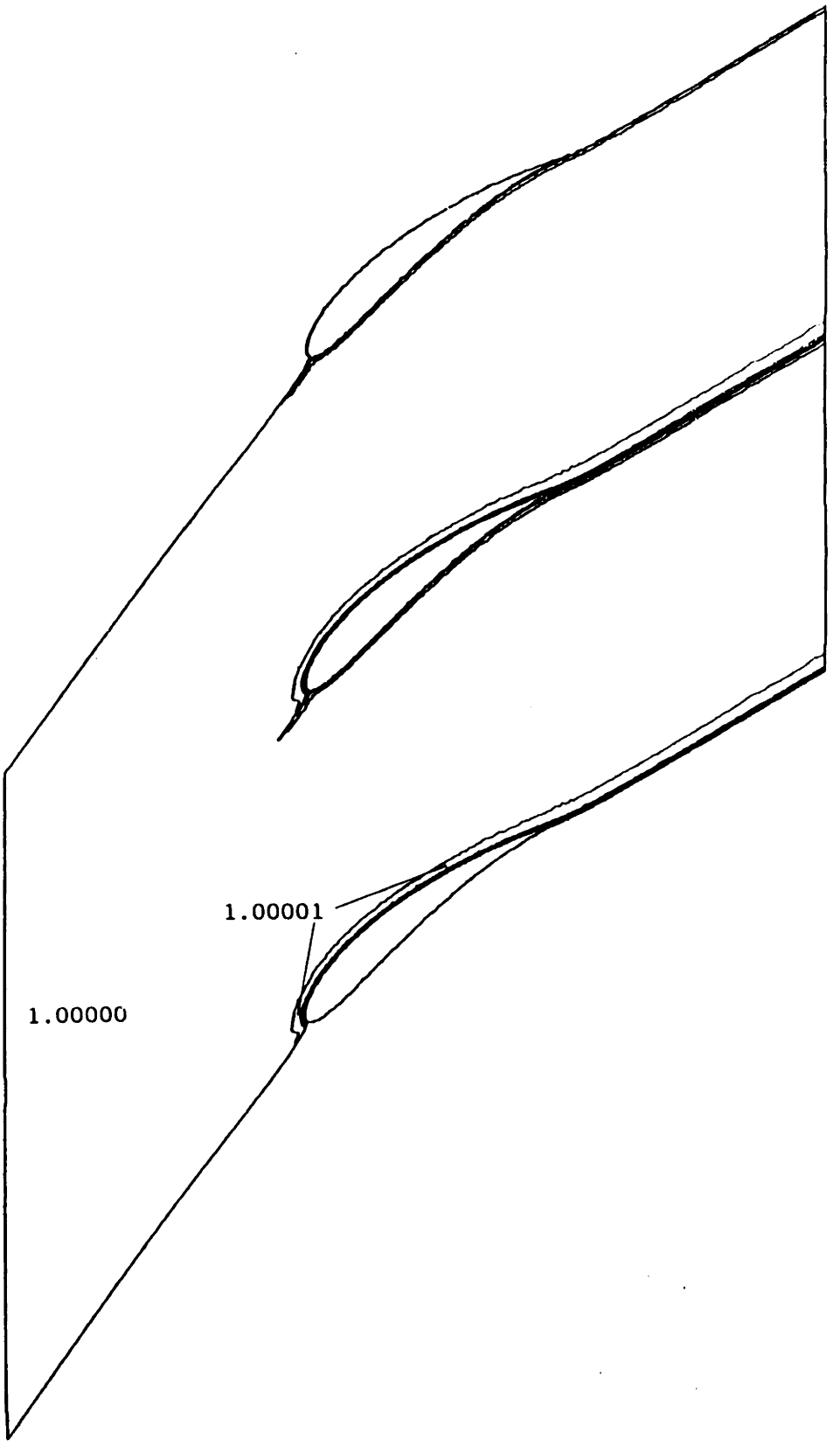


Figure 9.7c: Stagnation density contours for Gostelow cascade with increments of 0.00001.

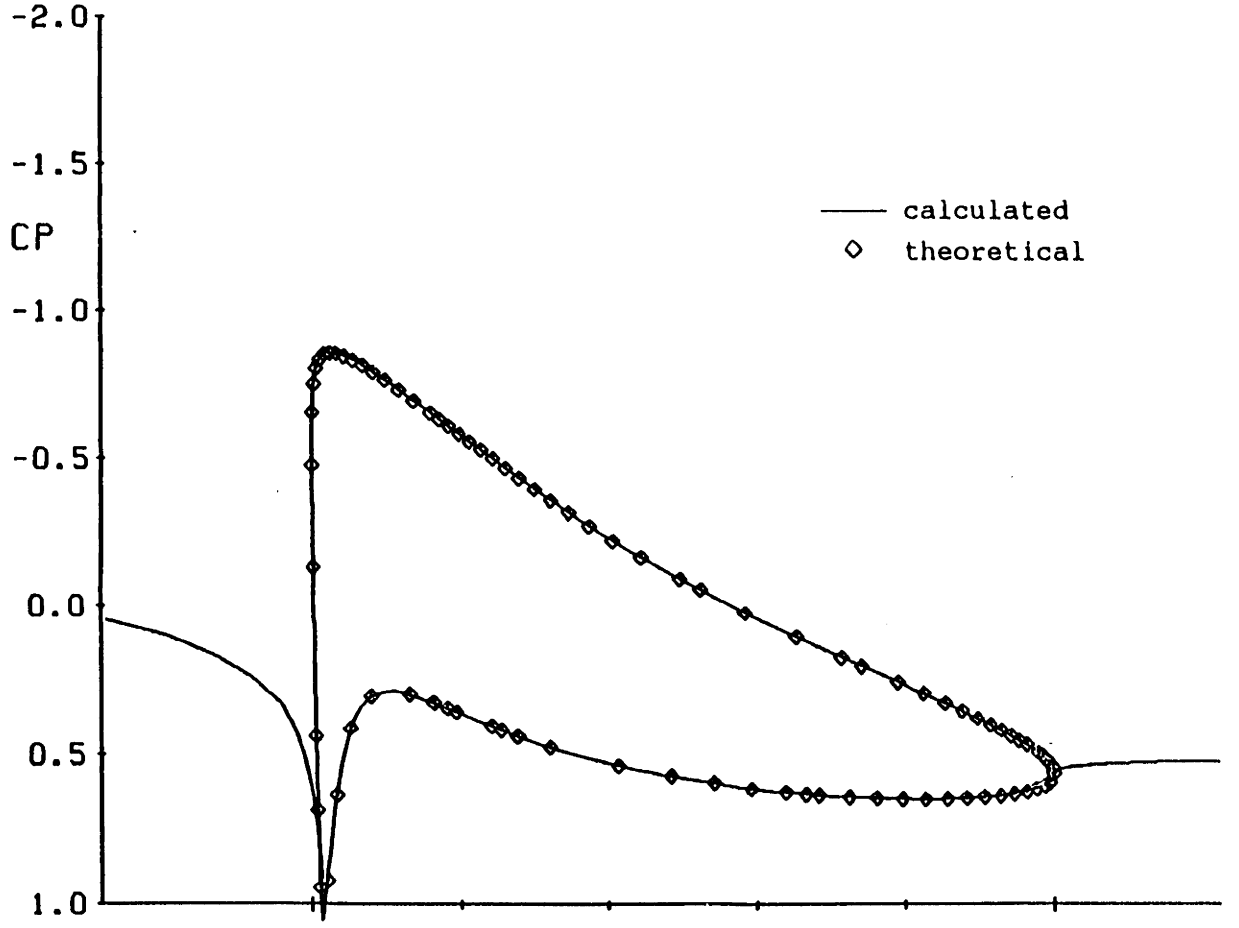


Figure 9.8: Comparison of calculated and theoretical surface pressure coefficients for Gostelow cascade.

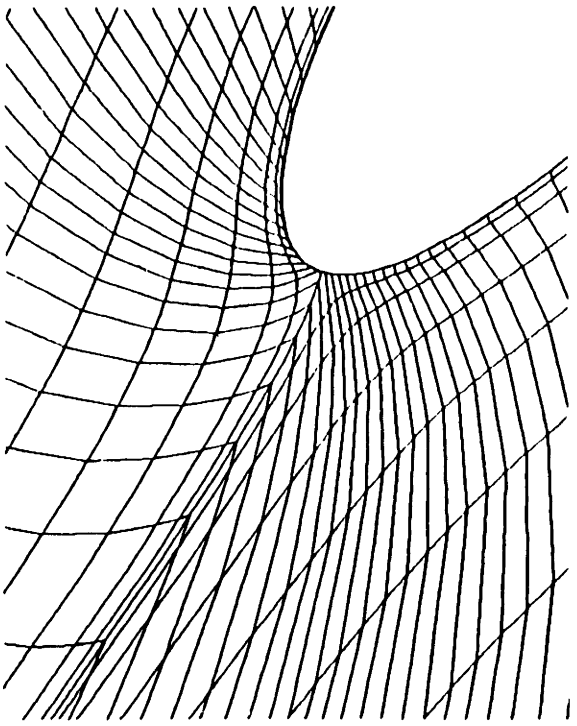


Figure 9.9a: Close-up of the grid near the leading edge stagnation point of the Gostelow cascade.

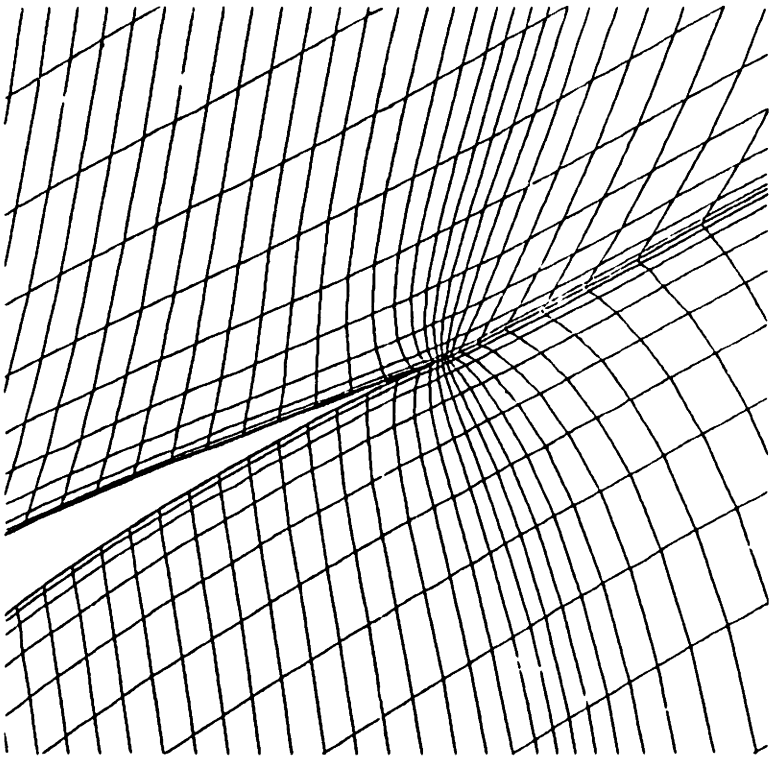


Figure 9.9b: Close-up of the grid near the trailing edge of the Gostelow cascade.

Table 9.7 lists the root-mean-square variation in density and node position at each iteration of the Newton procedure. It can be seen that after six iterations the solution is converged to machine accuracy. At the very first iteration an under-relaxation factor of 0.39 had to be employed to prevent an excessive change in the location of the leading edge stagnation point. Also after iterations 1, 2 and 4 the grid nodes had to be adjusted in the streamwise direction to maintain a 'good' grid. Both of these procedures were discussed in section 4.5.

Table 9.7: Newton iteration history for Gostelow cascade using direct solver for Newton equations

Iteration number	$\delta\rho/\rho$	δn	Comments
1	1.28e-1	6.64e-3	S, RLX=0.39
2	7.47e-2	3.95e-3	S
3	4.38e-5	1.19e-3	
4	6.17e-6	5.33e-5	S
5	1.65e-5	3.52e-5	
6	3.54e-6	6.81e-5	
7	1.19e-6	5.30e-5	
8	5.60e-7	2.49e-5	

Several calculations were performed with different positions of the inlet and outlet boundaries to determine the effects on the solution. Table 9.8 lists the results with the distances of the inlet boundary from the leading edge, and the outlet boundary from the trailing edge, non-dimensionalized by the axial chord length. These results show that the solution is relatively insensitive to the position of the inlet and outlet boundaries, due to the exponential decay of disturbaces in the streamwise direction, which was shown in section 6.1. The results also show that the solution is more sensitive to the position of the inlet boundary than the outlet boundary, which is consistent with the Mach number contours shown in Figure 9.7b which show stronger disturbances propagating upstream than downstream.

Table 9.8: Effects of position of inlet/outlet boundaries

Inlet	Outlet	Outlet flow angle
Gostelow		30.025^0
1.2	1.0	30.06^0
1.2	0.5	30.07^0
0.6	1.0	30.08^0
0.6	0.5	30.08^0

This case is also used to test the effectiveness of the iterative solver for solving the Newton equations. At each Newton iteration the Newton equations are approximately solved using twenty relaxation iterations, with each relaxation iteration applying the procedure described in Section 7.2. Table 9.9 lists the Newton iteration history. t can be seen that initially the residuals decay rapidly as in Table 9.7 which gives the iteration history for the direct method, but that after five iterations the residuals decay rather slowly with machine accuracy being reached in eleven iterations, compared to the five iterations required by the direct method. The total CPU time required is also greater than for the direct method since each Newton iteration using the iterative solver took approximately 1.7 mins., compared to 1.2 mins. for each iteration using the direct solution method. The total CPU time for the iterative method is independent of the choice of the number of relaxation iterations per Newton iteration, because the limiting feature is the rate-of-convergence of the linear relaxation process, not the quadratic Newton procedure. Thus decreasing the number of relaxation iterations per Newton iteration will increase the number of Newton iterations while keeping fixed the total number of realaxation iterations and the total CPU time.

Table 9.9: Newton iteration history for Gostelow cascade using iterative solver for Newton equations

Iteration number	$\delta\rho/\rho$	δn	Comments
1	1.28e-1	6.64e-2	S, RLX=0.92
2	9.14e-3	3.20e-2	
3	3.10e-5	3.61e-3	
4	1.28e-5	8.25e-4	S
5	1.19e-5	5.61e-4	
6	6.65e-6	3.65e-4	
7	4.60e-6	2.54e-4	
8	3.33e-6	1.82e-4	
9	1.89e-6	1.10e-4	
10	1.51e-6	8.93e-5	
11	1.18e-6	6.49e-5	
12	6.03e-7	4.71e-5	
13	6.90e-7	3.48e-5	
14	4.85e-7	2.48e-5	
15	8.16e-7	4.93e-5	

9.4 T7 turbine cascade

The T7 turbine cascade is a subsonic linear cascade designed by Rolls-Royce, for which there are experimental surface pressure measurements [27]. This case is presented here to test the robustness of the algorithm. In particular, the inflow angle is approximately 50^0, and the outflow angle is approximately -70^0, so the grid is badly sheared in most of the flow domain, as can been seen in Figure 9.10a, which shows a fully converged 117x21 grid. Sheared grids cause difficulties for time-marching Euler methods, requiring more restrictive time-step limitations and more numerical smoothing. Table 9.10 presents the Newton iteration history for this case, showing that the Newton procedure appears unaffected by the high grid shearing. After six iterations the solution has converged to machine accuracy. An under-relaxation factor of 0.88 was required on the first iteration, and streamwise node adjustments were performed after the first three Newton iterations. The CPU time per iteration was 1.1 mins.

Figures 9.10b and 9.10c show contours of the Mach number and the stagnation density. Figure 9.11 presents a close-up of the grid at the leading edge stagnation point, showing that good resolution is obtained by varying the mass flux in each streamtube, with the stagnation streamtubes having the least mass flux. Finally, Figure 9.12 shows a comparison between the calculated surface Mach numbers and those measured experimentally. The agreement is good except towards the trailing edge. In [10] it is shown that the inclusion of viscous effects through a coupled integral boundary layer analysis improves the agreement in this region.

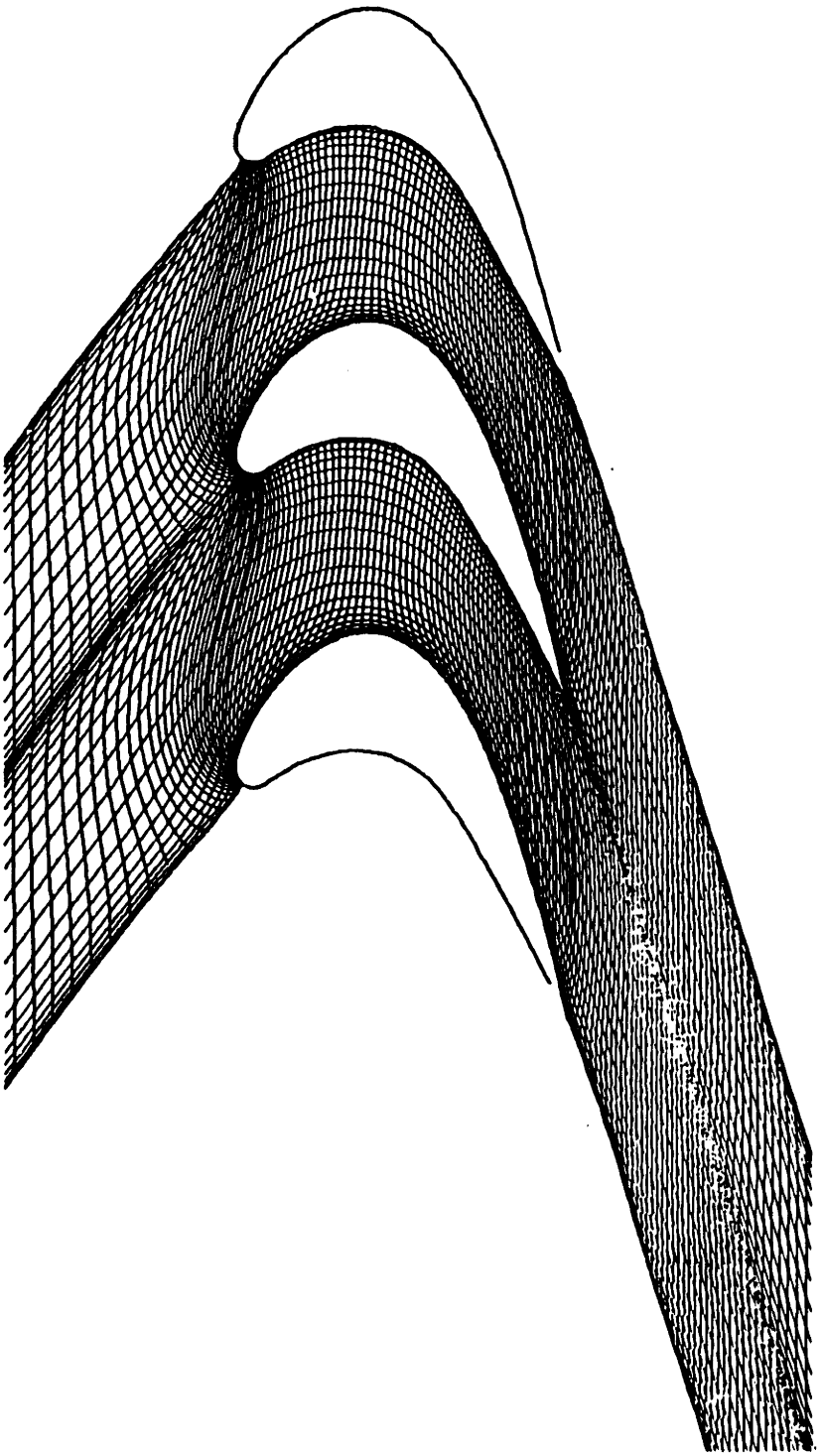


Figure 9.10a: Airfoil and grid geometry for T7 turbine cascade.

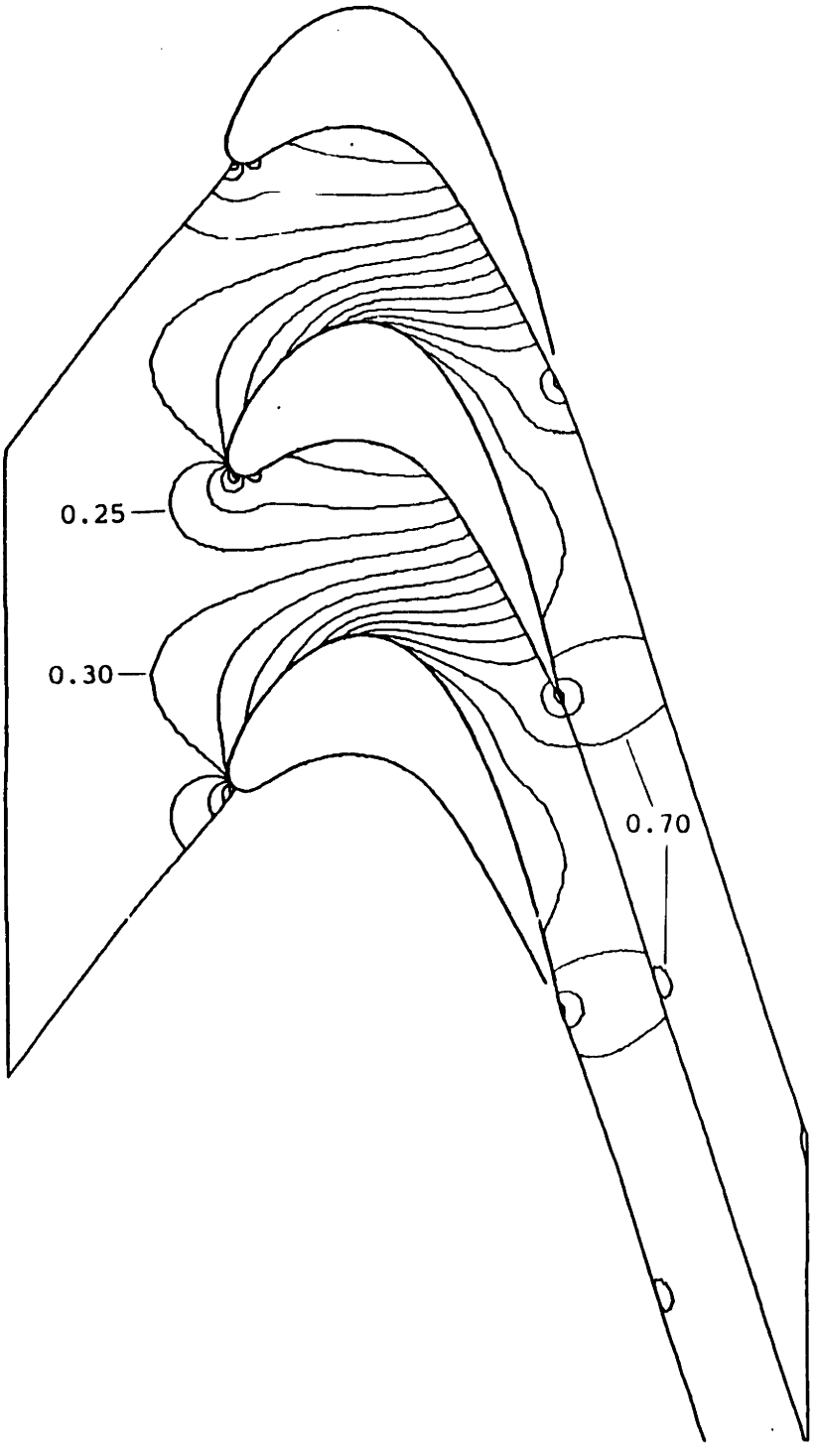


Figure 9.10b: Mach number contours for T7 turbine cascade with increments of 0.05.

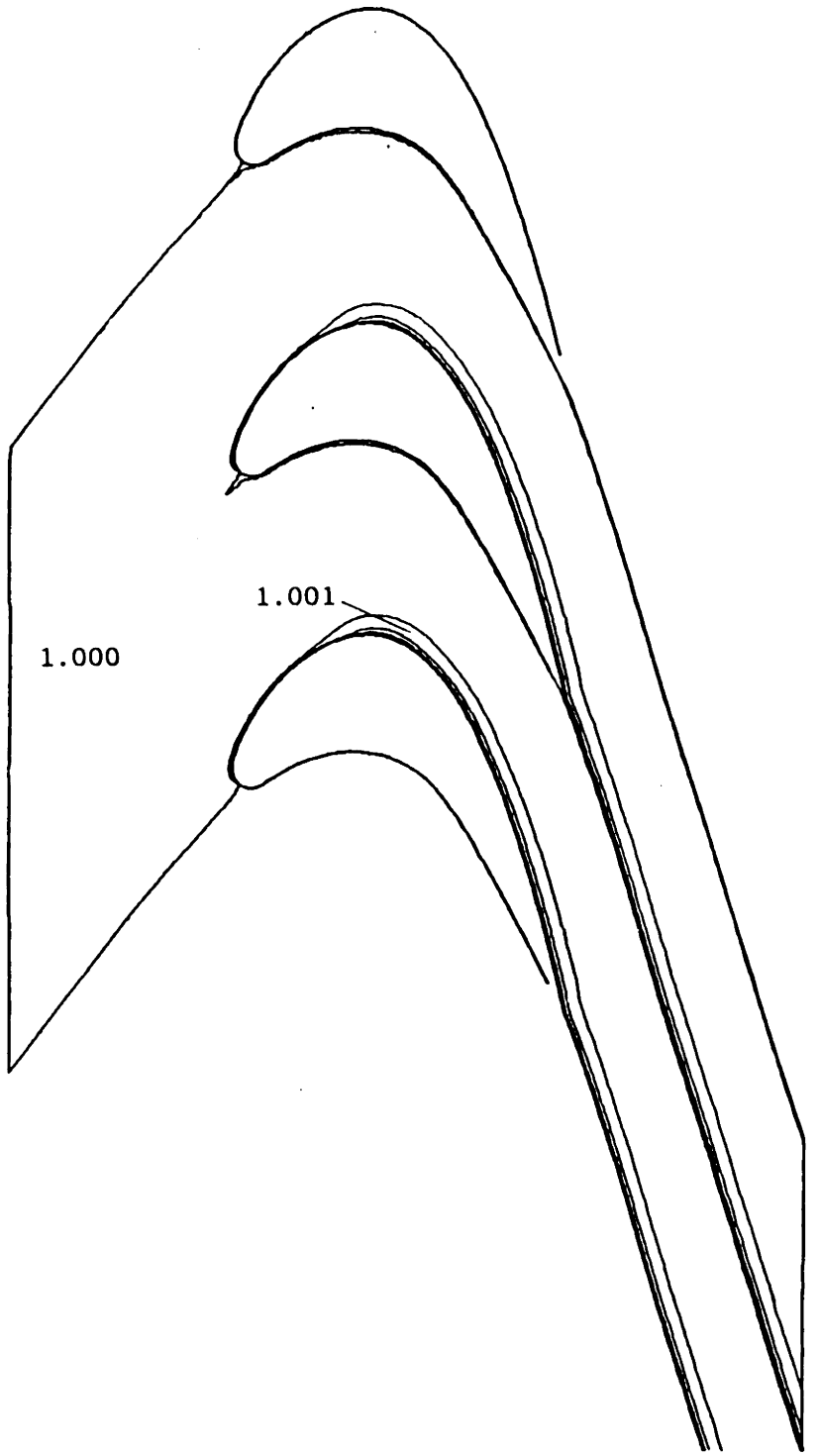


Figure 9.10c: Stagnation density contours for T7 turbine cascade with increments of 0.001.

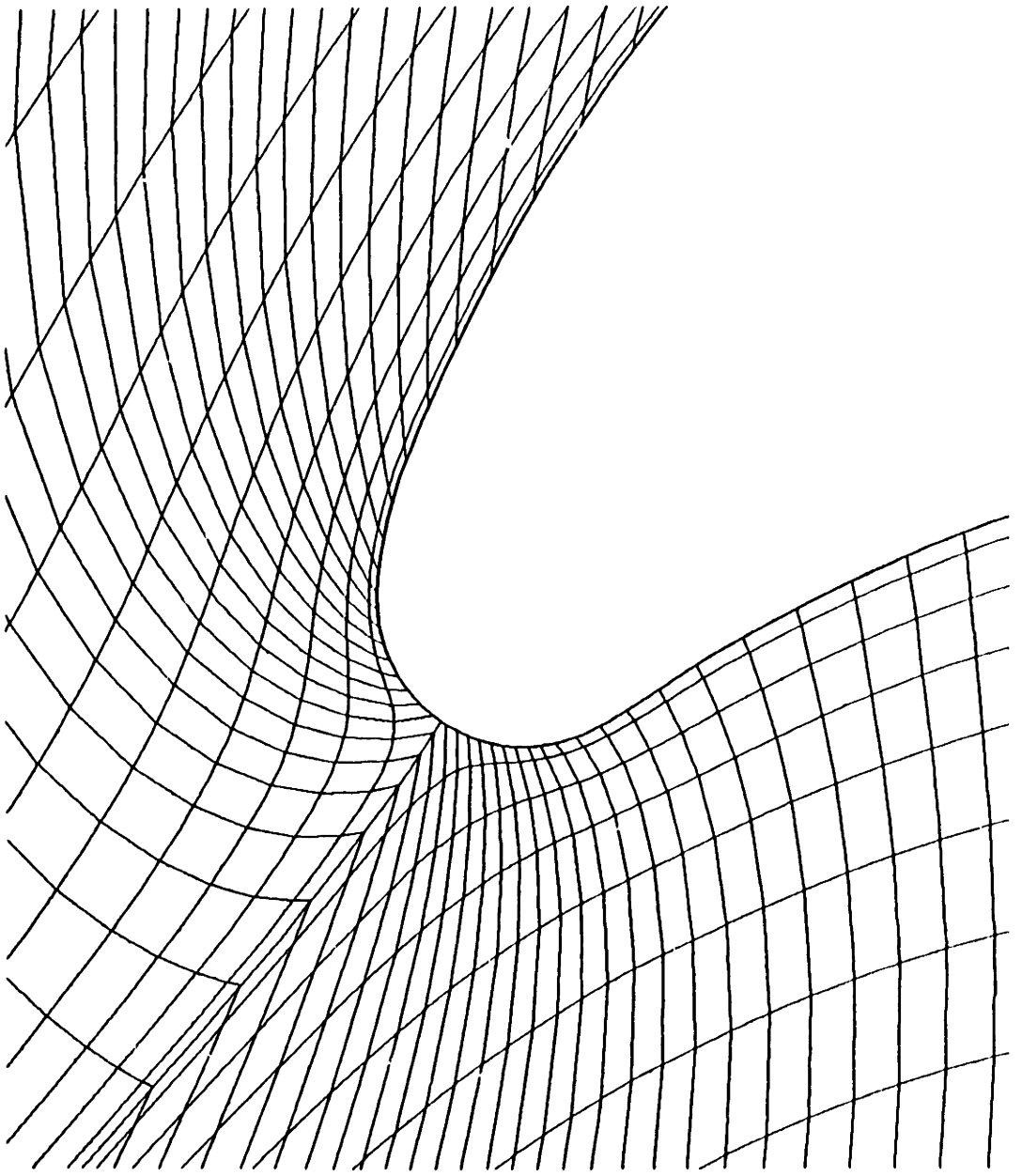


Figure 9.11: Close-up of the grid near the leading edge stagnation point of the T7 turbine cascade.

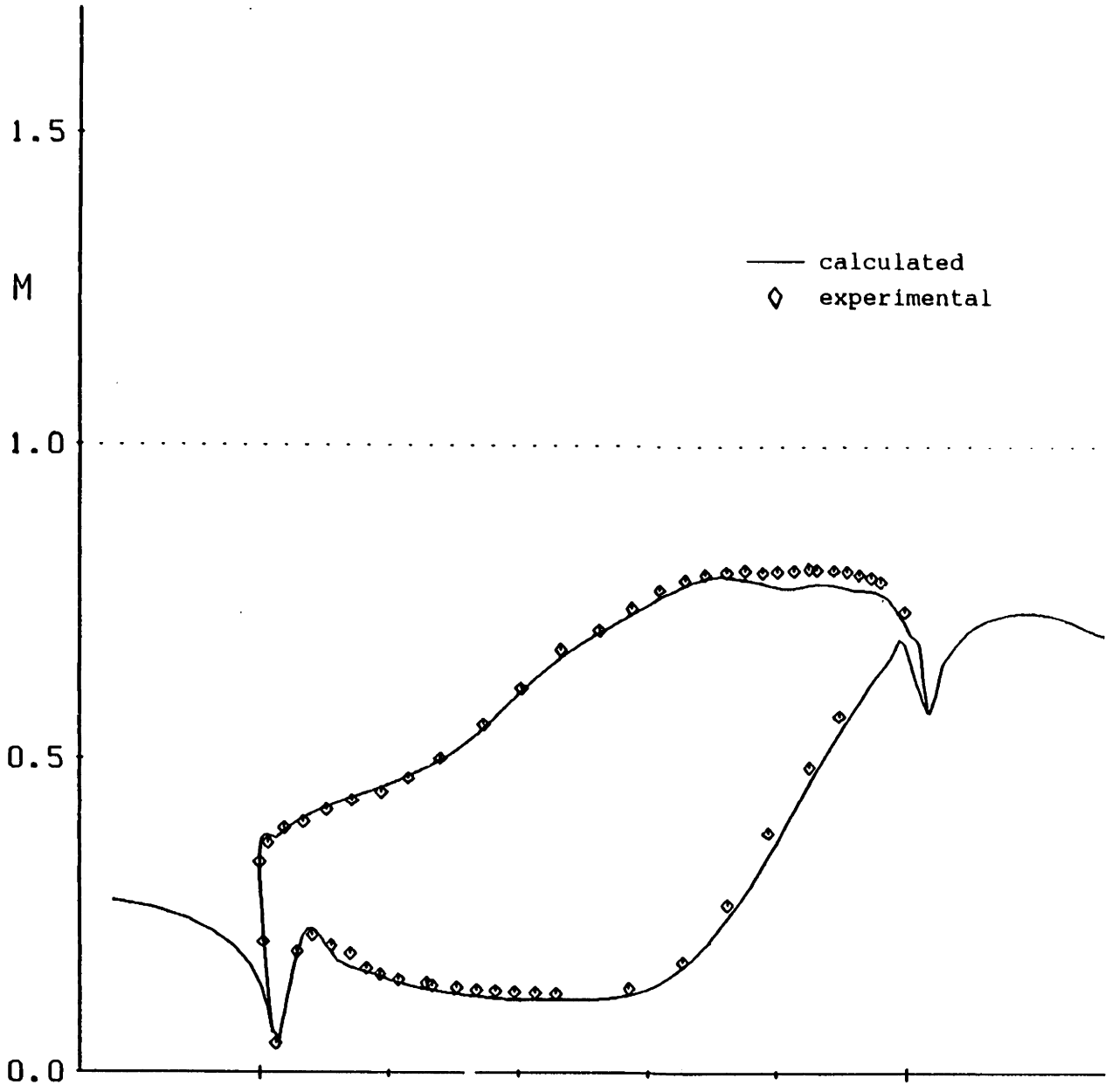


Figure 9.12: Comparison of calculated and experimental surface Mach numbers for T7 turbine cascade.

Table 9.10: Newton iteration history for T7 turbine cascade

Iteration number	δρ/ρ	δn	Comments
1	9.62e-2	1.06e-1	S, RLX=0.88
2	1.79e-2	1.92e-2	S
3	8.96e-4	4.20e-3	S
4	5.16e-4	2.06e-4	
5	3.95e-5	7.12e-6	
6	9.79e-6	1.22e-6	
7	5.53e-6	6.44e-7	
8	8.29e-6	9.90e-7	
9	1.08e-5	1.51e-6	
10	8.79e-6	1.91e-6	

This case is also used to demonstrate the flexibility of the Newton approach. Figure 9.13 shows the variation of the lift coefficient as the inlet flow angle is varied, with the total mass flow and stagnation enthalpy and density being held fixed. Since the upstream velocity and static density are not constant, the lift is non-dimensionalized with respect to the axial chord length, the inlet stagnation density and the inlet stagnation speed of sound. Each of the calculated points on the curve were obtained using only three Newton iterations, using the solution from the previous point as the initial solution. Also for comparison the dotted straight line through the design point has the slope predicted by the design point calculation using the global linear sensitivities, as discussed at the end of Section 8.2. The agreement is excellent, showing the utility of the linear sensitivities.

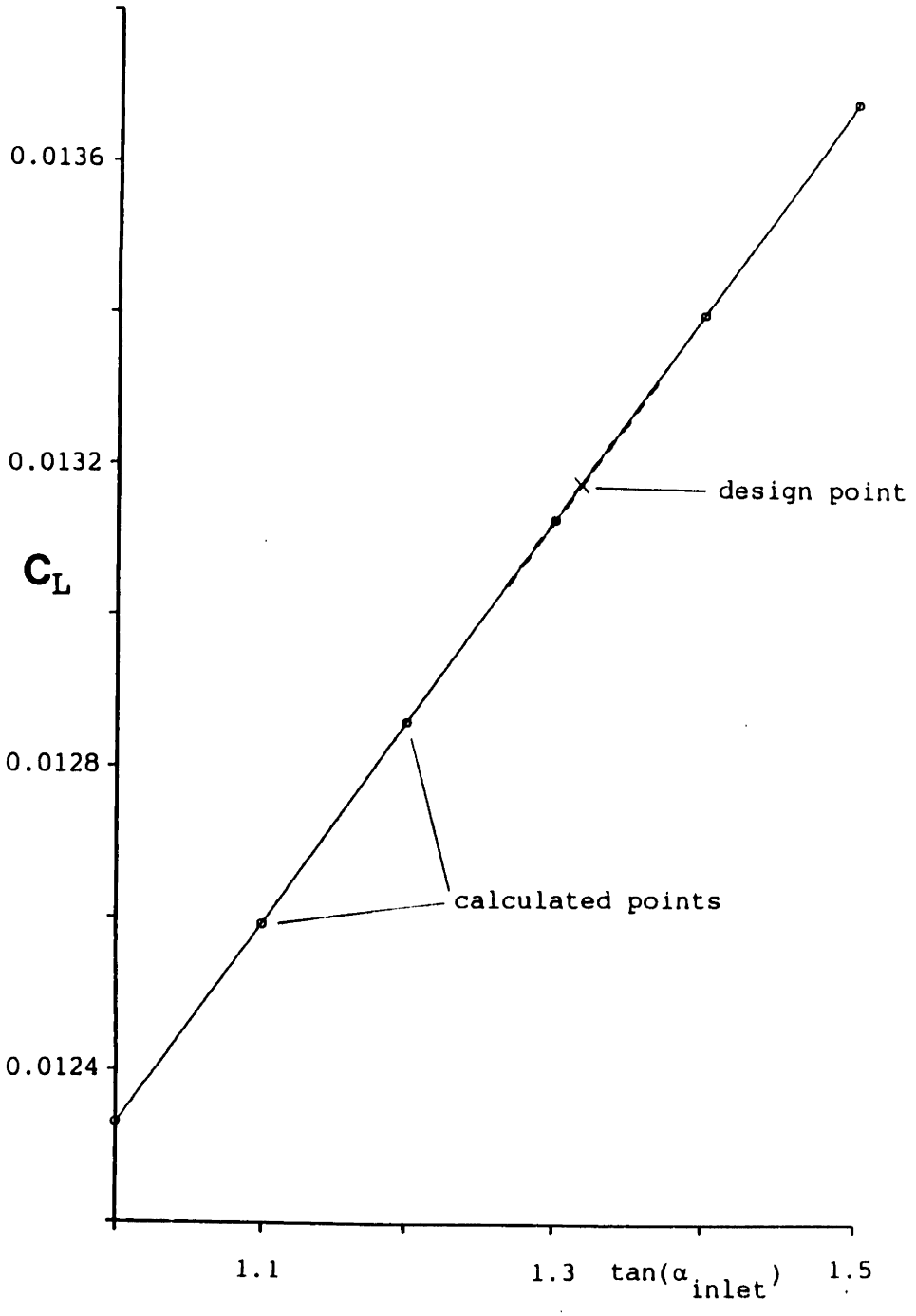


Figure 9.13: Variation of lift with inflow angle for T7 turbine cascade.

9.5 Garabedian compressor cascade

This test case is a supercritical compressor cascade designed by Garabedian using a numerical hodograph method [3], which is considered to be an accurate calculation, against which other numerical methods can be compared. The geometry is shown in Figure 9.14a with a converged 78x23 grid. An interesting feature of the geometry is that Garabedian included the displacement thickness due to a boundary layer in his calculations so the the trailing edge of the airfoil is not closed, and this gap is held fixed in the wake. This was treated in the current formulation by setting the prescribed wake thickness in the initial grid, which is then maintained throughout the calculation, while the wake position is free to move as usual under the condition that the pressures on the two sides of the wake are equal.

One point of interest in this case is the effect of the artificial compressibility, and the second order corrections. Figures 9.14b and 9.14c show the contours of Mach number and stagnation density for a solution obtained with only first order artificial compressibility with μ_{con}=1.0 and M^2_{crit}=0.7. Figure 9.15 presents a comparison between the calculated surface Mach numbers, and those given by Garabedian [3]. The agreement is very good except in the supersonic zone, where slight numerical errors and the formation of a very weak shock produce stagnation density changes of the order of 1% which are convected downstream, as can be seen in Figure 9.14c. Figures 9.16a,b,c show the grid, and contours of Mach number and stagnation density changes, for a solution obtained with second order artificial compressibility corrections, as detailed in Section 3.5. The stagnation density changes have been greatly reduced, and the agreement with Garabedian's surface Mach number distribution, shown in Figure 9.17, is much better. The only slight differences are at the points at which the sonic line joins the surface, near which the hodograph method has some difficulty, so that the disagreement is as likely due to the hodograph method as it is due to the

present method.

Figure 9.18 shows a close-up of the grid and Mach number contours at the leading edge. Despite the good grid resolution, the Mach number changes by up to 0.07 between streamwise stations due to the very rapid flow expansion on the suction surface side of the leading edge, but this is better than the resolution achieved by Garabedian, and the second order accuracy keeps the stagnation density errors small.

The increase in accuracy using the second order density corrections for the artificial compressibility, is achieved at the cost of rate of convergence of the Newton iteration procedure. Tables 9.11 and 9.12 show the iteration histories for the two cases. The case with first order artificial compressibility converges to machine accuracy in six iterations, while the one with the second order corrections requires an additional nine iterations, during which the residuals decay by a factor of approximately 0.5 at each iteration. This non-quadratic terminal convergence rate is due to the approximations in the Newton linearization introduced in Section 4.3, in particular the neglect of changes in ρ_{-1} and μ_c. The removal of these approximations would require modifications to the direct solution method for the Newton equations which would greatly increase the computational cost of each Newton iteration, and so would not produce any overall savings in the computational cost of the solution. The CPU time per iteration in these calculations was 50 secs.

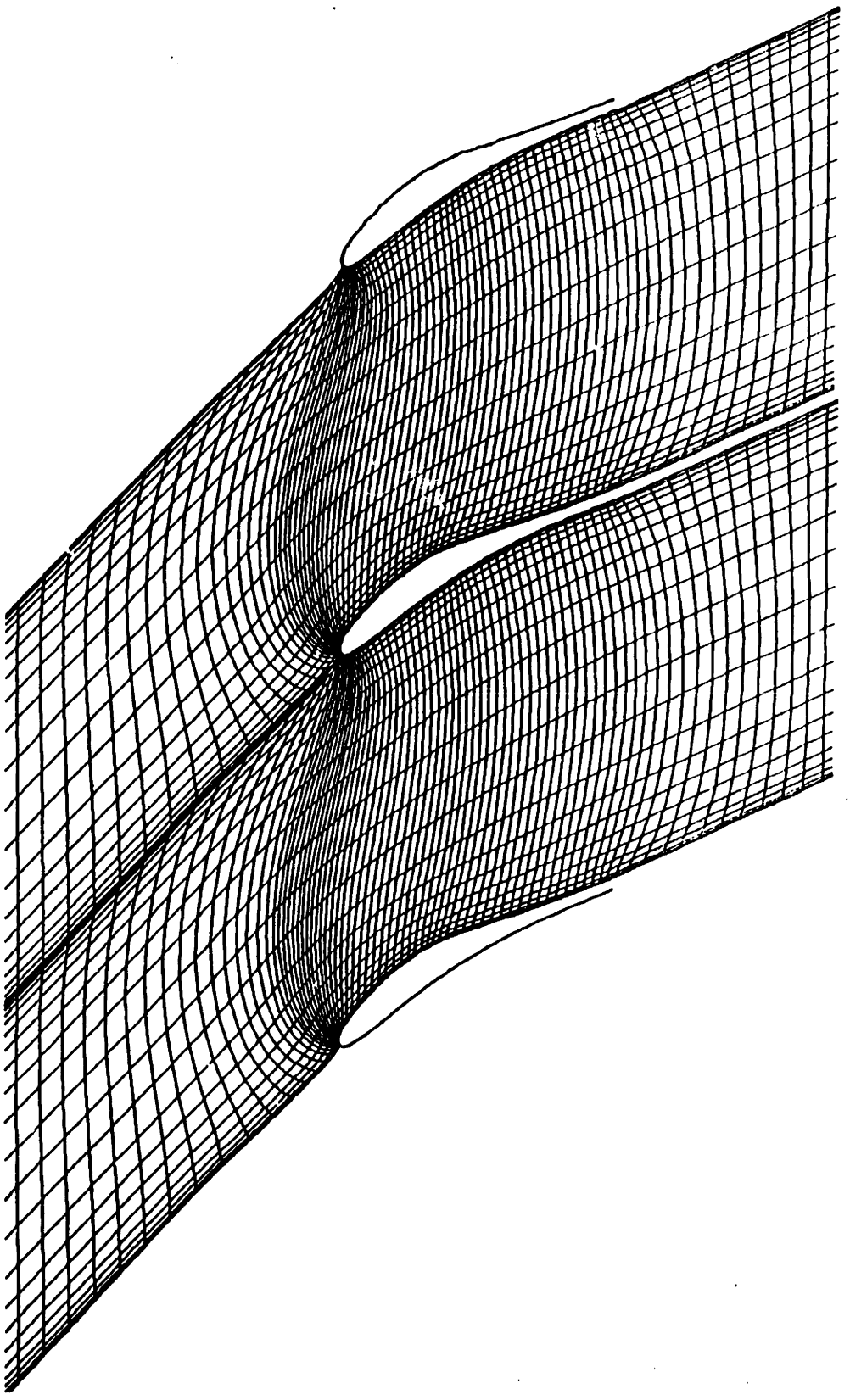


Figure 9.14a: Airfoil and grid geometry for Garabedian cascade, using first order artificial compressibility.

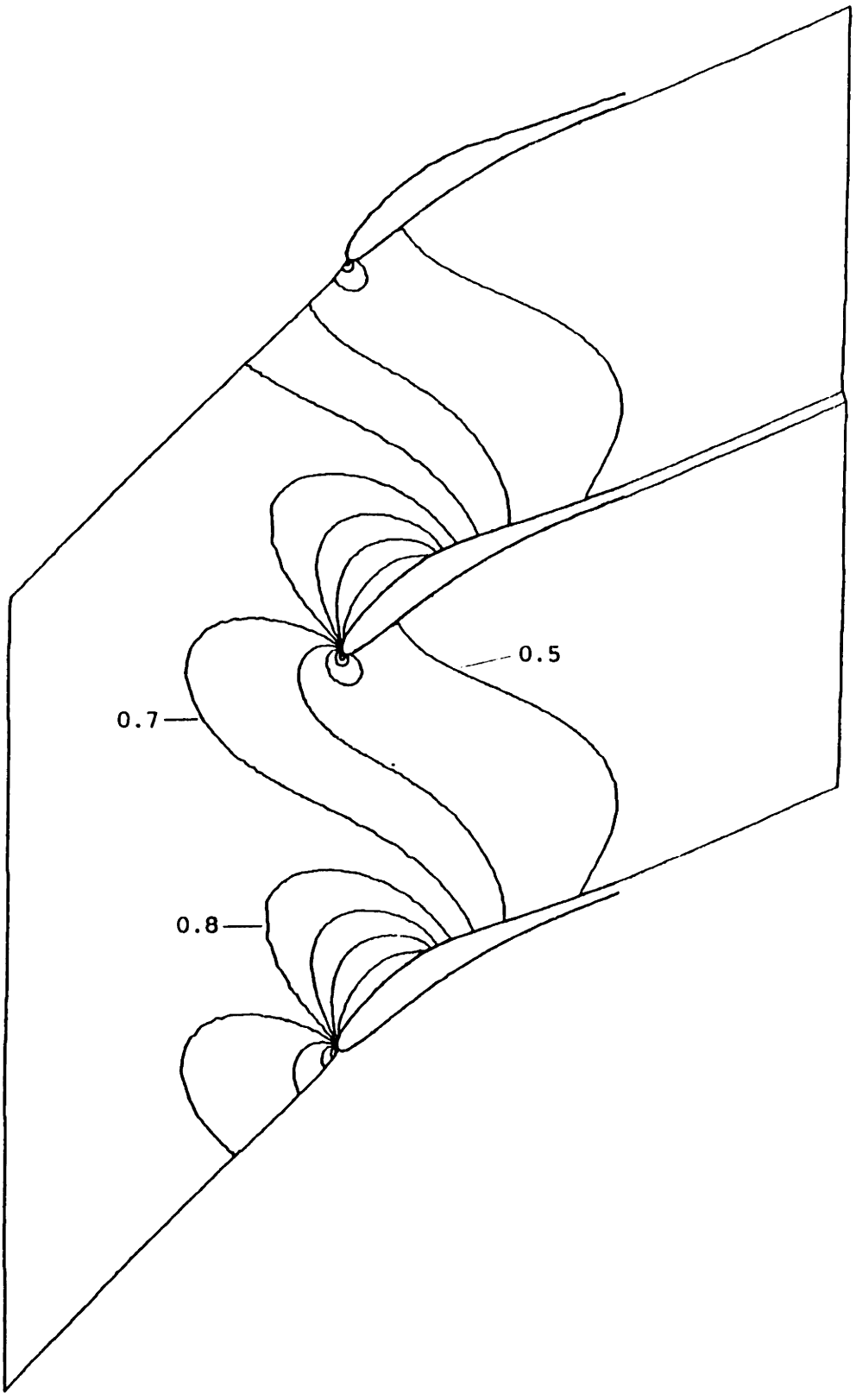


Figure 9.14b: Mach number contours for Garabedian cascade, using first order artificial compressibility with increments of 0.1.

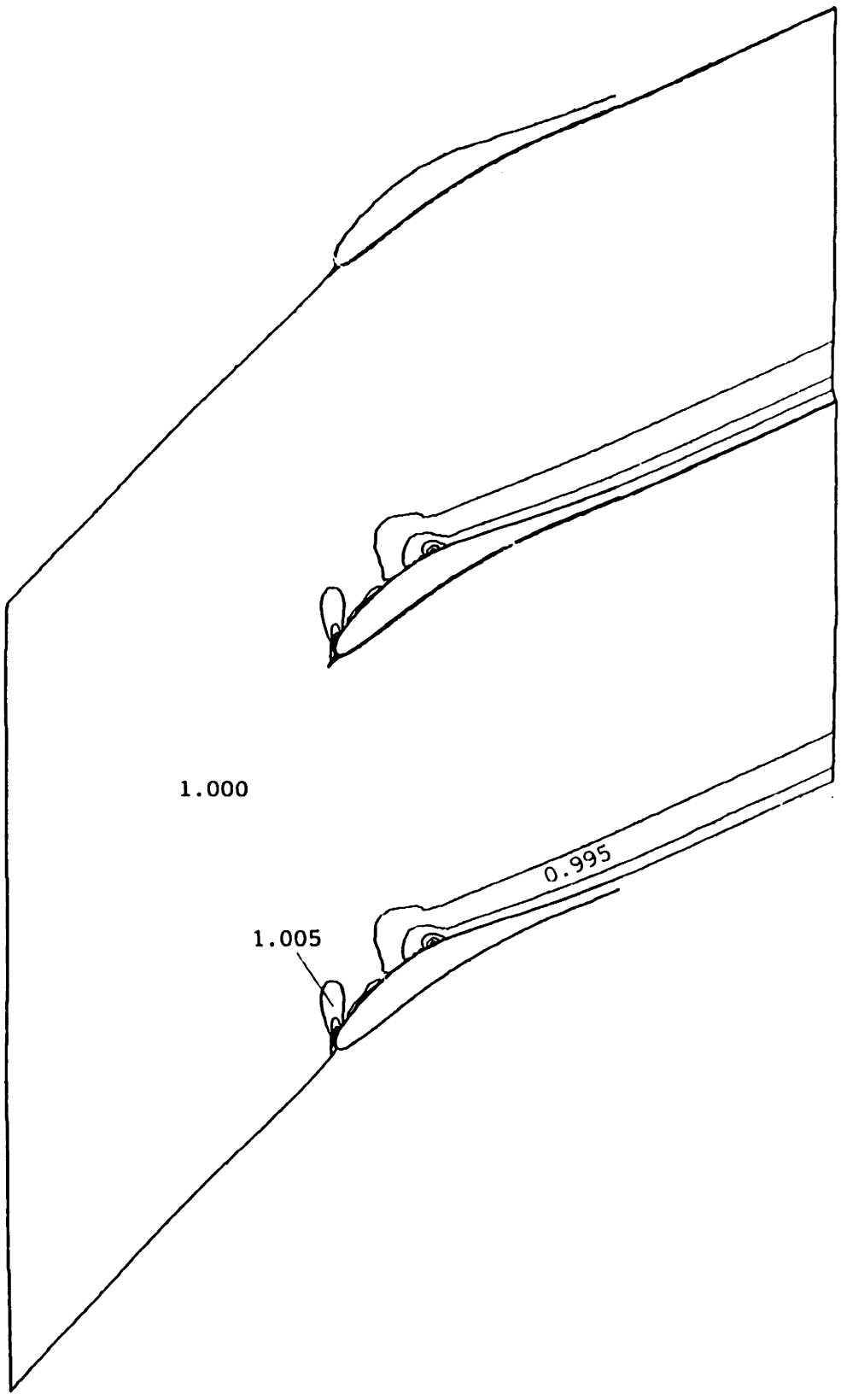


Figure 9.14c: Stagnation density contours for Garabedian cascade, using first order artificial compressibility with increments of 0.005.

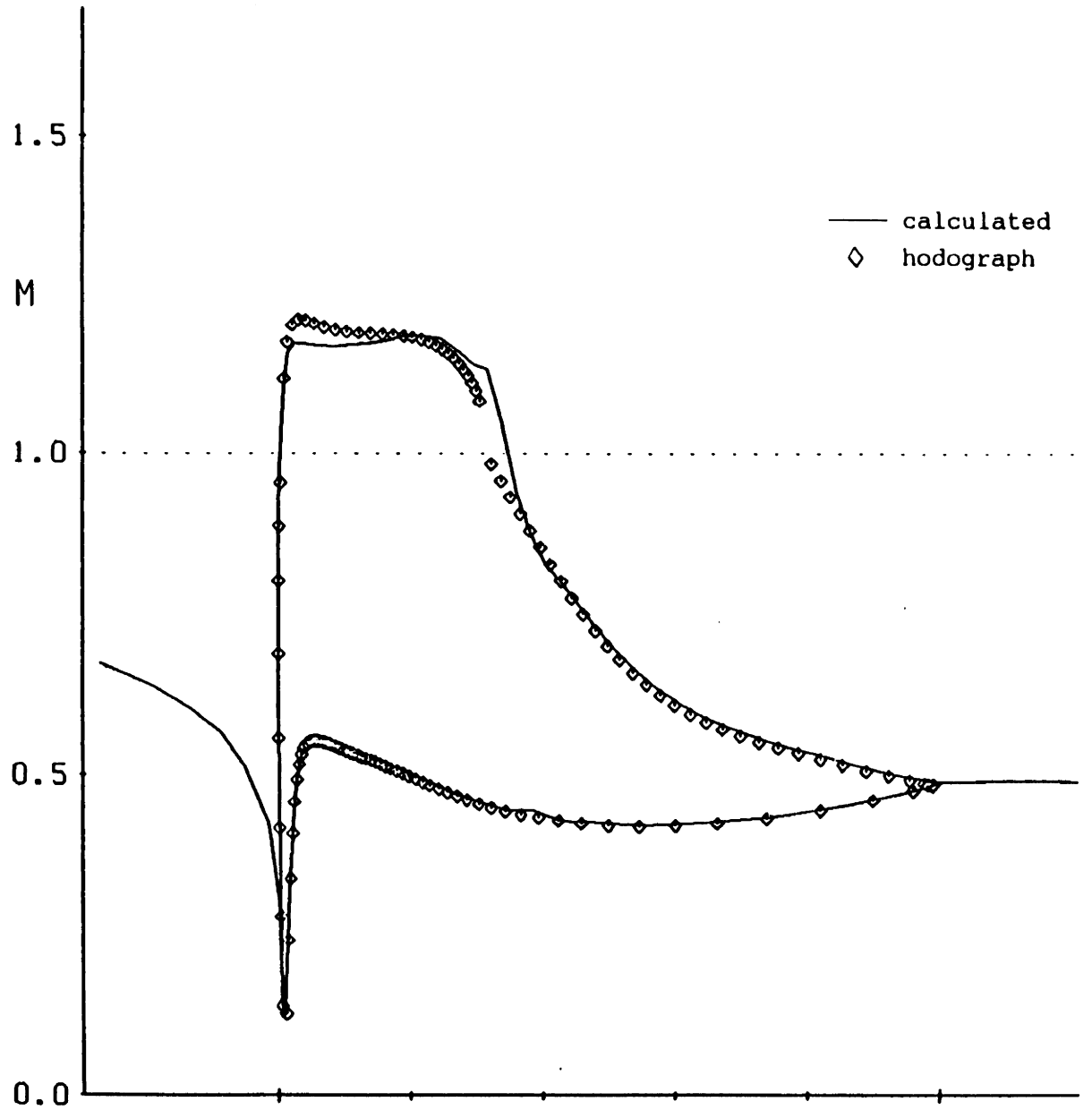


Figure 9.15: Comparison of calculated and hodograph surface Mach numbers for Garabedian cascade, using first order artificial compressibility.

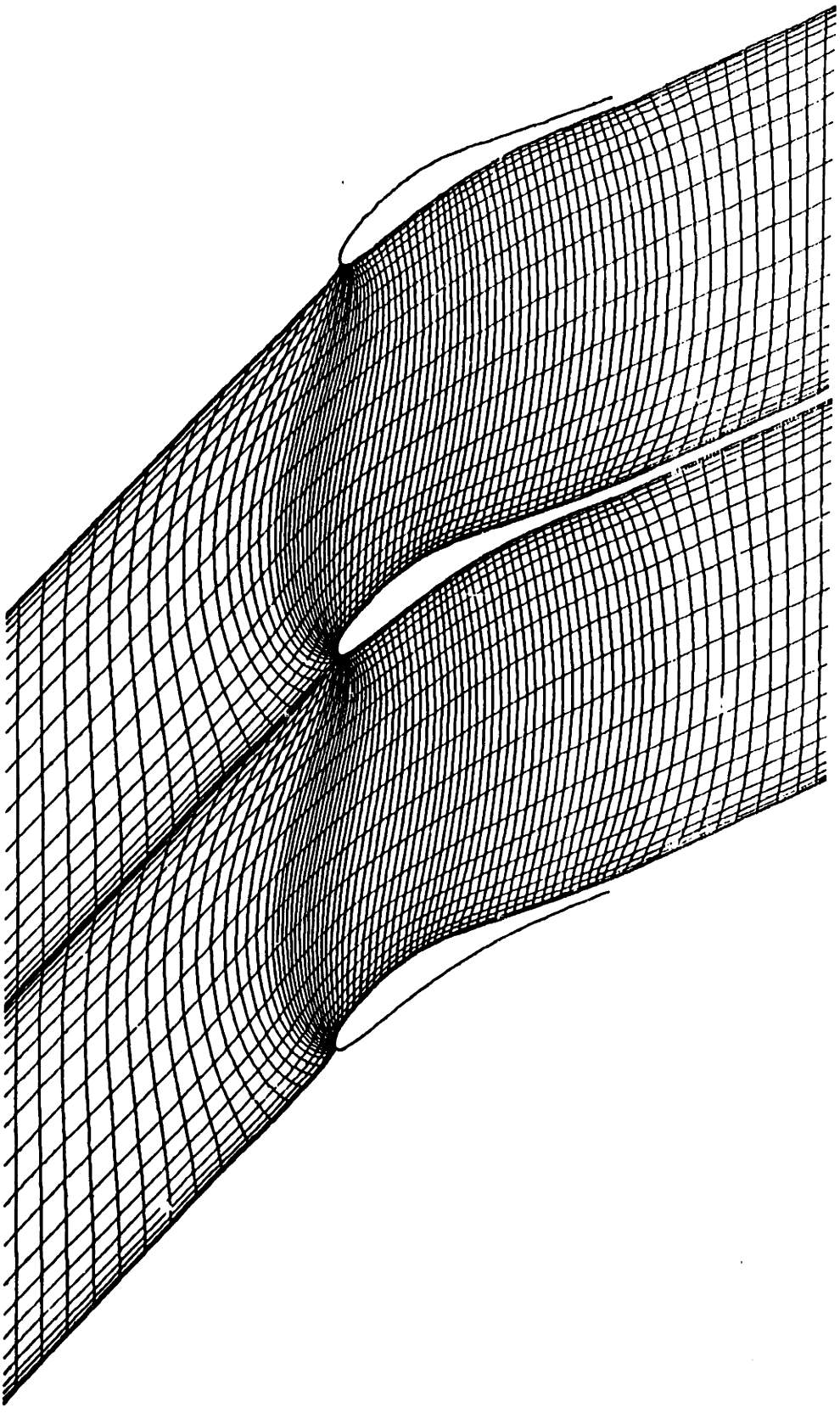


Figure 9.16a: Airfoil and grid geometry for Garabedian cascade, using second order artificial compressibility.

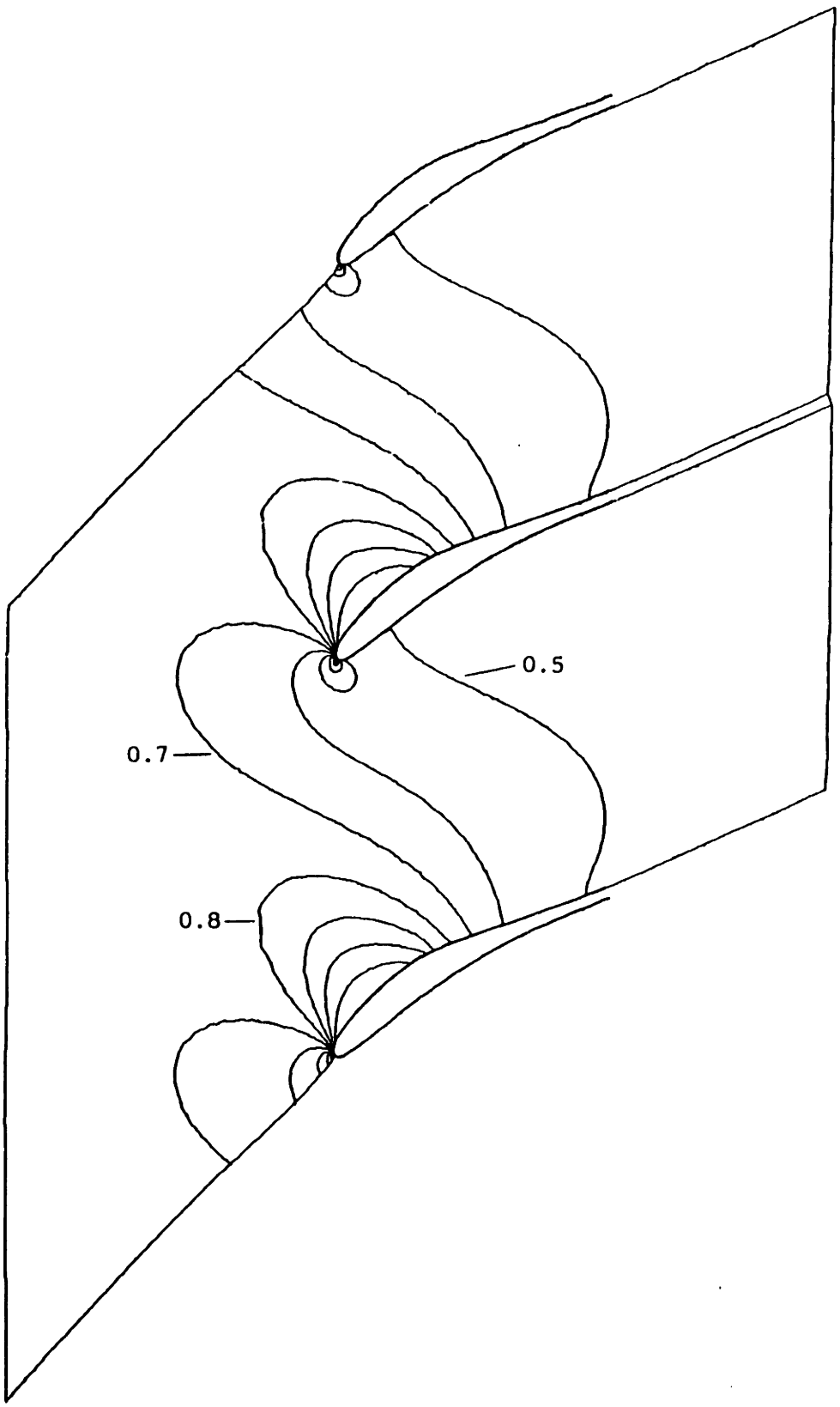


Figure 9.16b: Mach number contours for Garabedian cascade, using second order artificial compressibility with increments of 0.1.

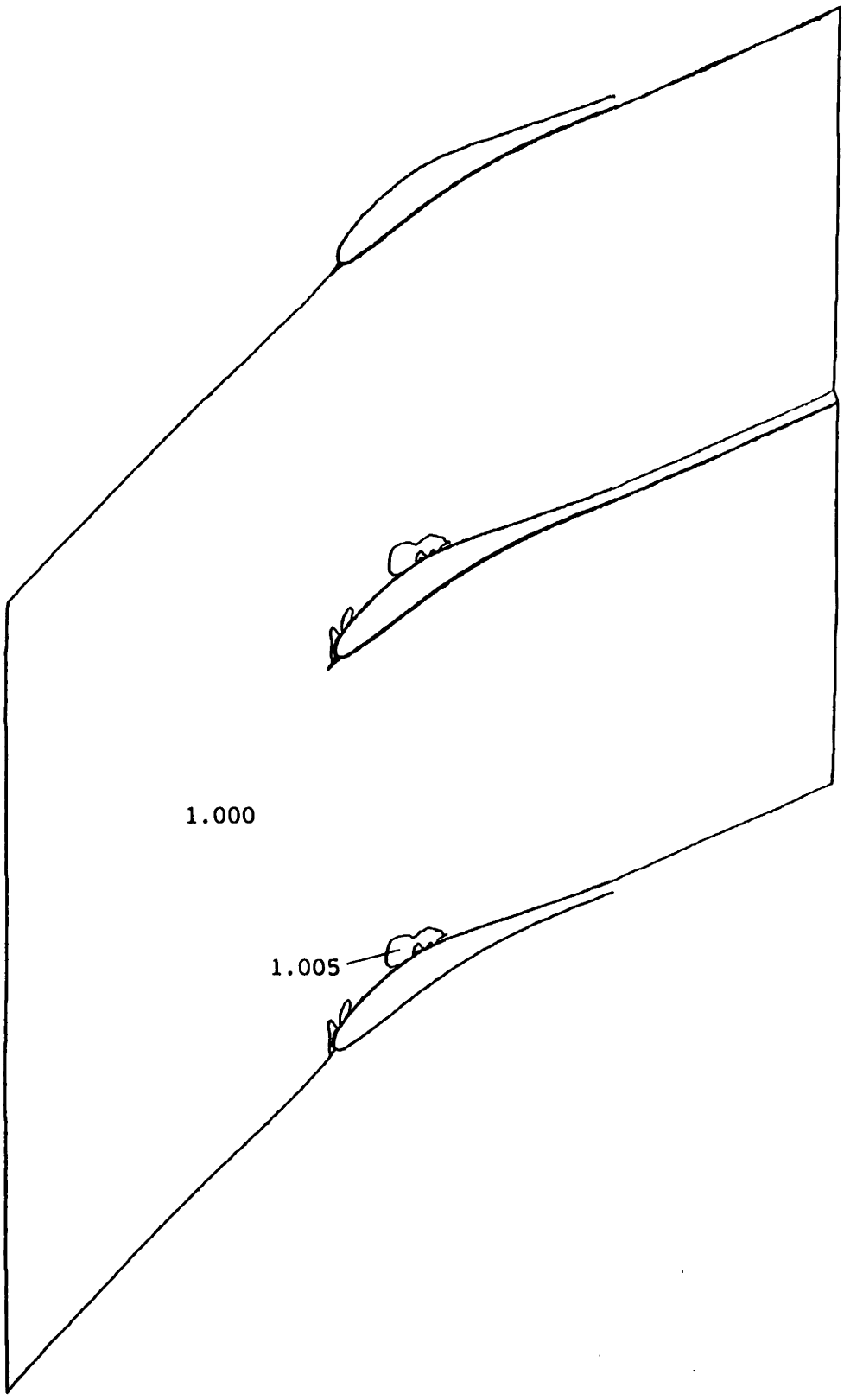


Figure 9.16c: Stagnation density contours for Garabedian cascade, using second order artificial compressibility with increments of 0.005.

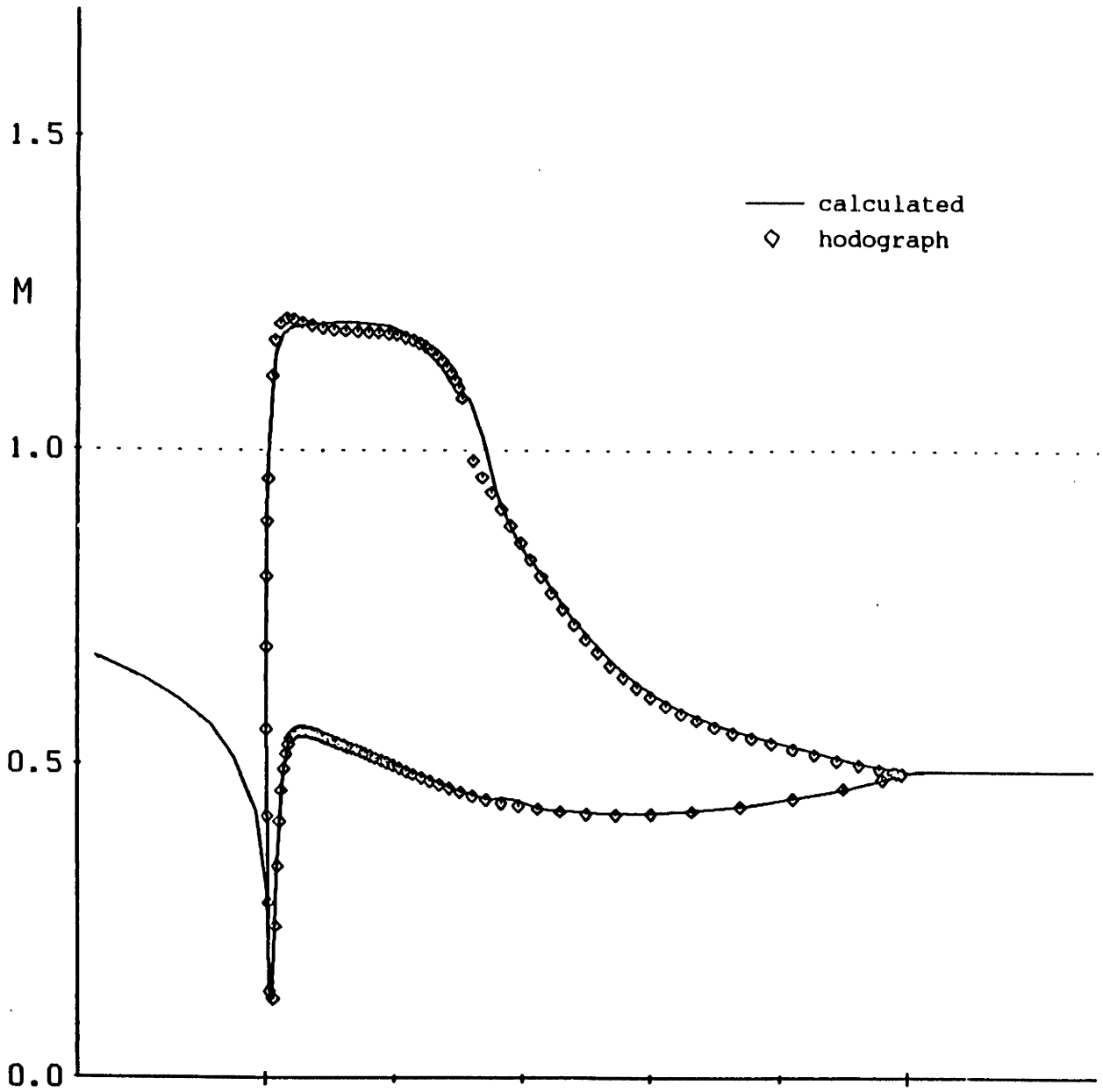


Figure 9.17: Comparison of calculated and hodograph surface Mach numbers for Garabedian cascade, using second order artificial compressibility.

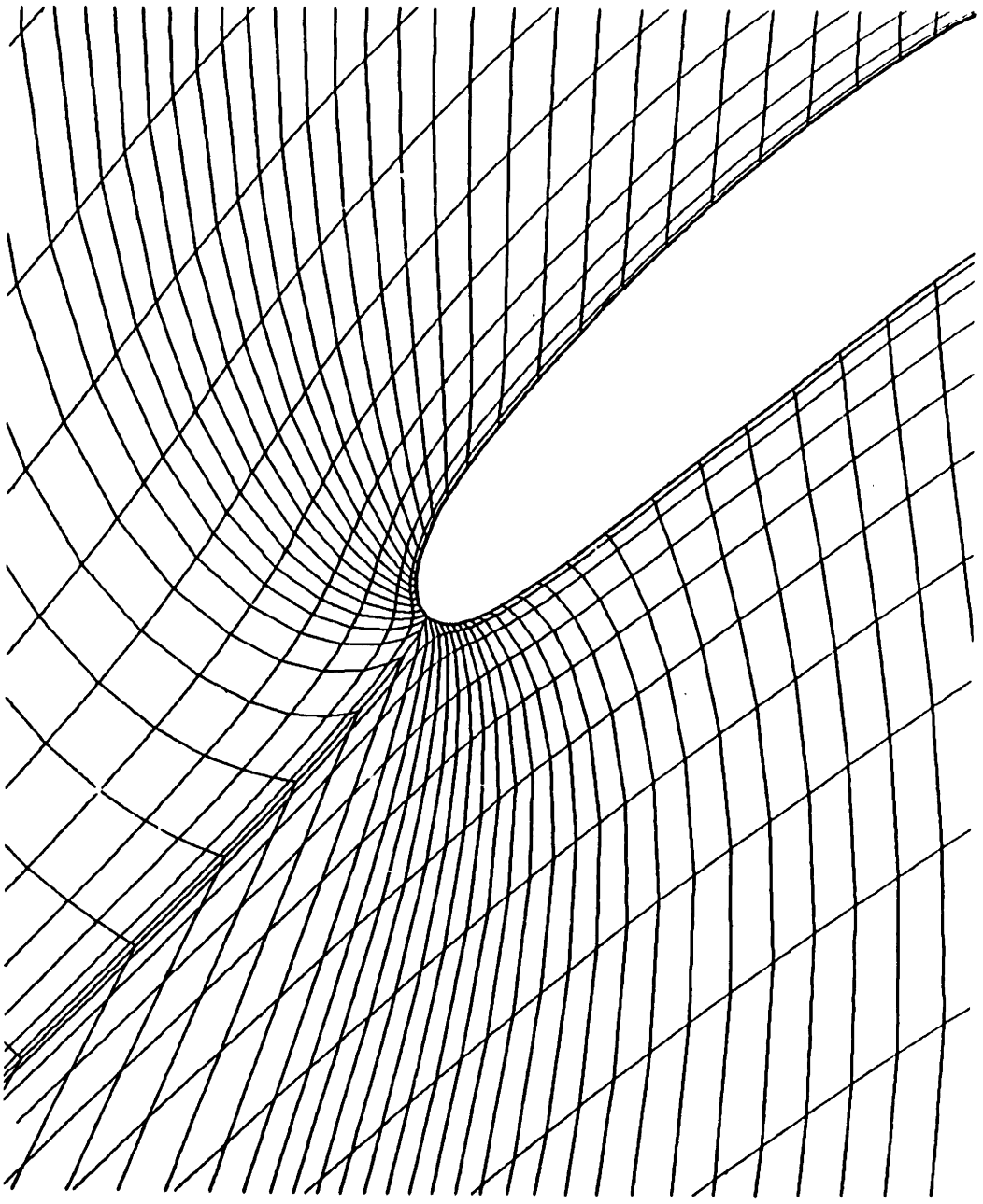


Figure 9.18: Close-up of the grid near the leading edge stagnation point of the Garabedian cascade.

Table 9.11: Newton iteration history for Garabedian cascade with first order artificial compressibility

Iteration number	$\delta\rho/\rho$	δn	Comments
1	1.09e-1	7.43e-3	S, RLX=0.52
2	7.02e-2	5.06e-3	S
3	1.22e-2	3.08e-3	
4	1.46e-3	7.15e-5	
5	4.49e-5	2.44e-6	
6	1.26e-5	5.18e-7	
7	2.95e-6	8.47e-7	
8	2.60e-6	7.79e-7	
9	2.36e-6	4.65e-7	
10	4.97e-6	8.94e-7	

Table 9.12: Newton iteration history for Garabedian cascade with second order artificial compressibility

Iteration number	$\delta\rho/\rho$	δn	Comments
1	1.09e-1	7.43e-3	S, RLX=0.52
2	7.02e-2	5.06e-3	S
3	1.25e-2	3.23e-3	↑
4	7.37e-3	8.17e-4	Supersonic region
5	5.78e-3	4.03e-4	settling down
6	3.36e-3	1.30e-4	
7	1.41e-3	4.57e-5	↓
8	7.93e-4	3.13e-5	↑
9	4.26e-4	1.49e-5	
10	2.25e-4	8.11e-6	Steady convergence with spectral radius ≈0.5
11	1.10e-4	2.80e-6	
12	5.57e-5	1.99e-6	
13	2.56e-5	5.00e-7	
14	1.19e-5	5.17e-7	
15	7.52e-6	6.04e-7	↓
16	3.99e-6	8.92e-7	
17	3.24e-6	6.42e-7	
18	4.86e-6	5.89e-7	
19	3.22e-6	6.12e-7	
20	3.18e-6	6.18e-7	

9.6 NACA 0012 airfoil

This test case is one of a series of cases proposed by the AGARD Fluid Dynamics Panel, Working Group 07 to formulate a set of benchmark solutions, against which new computational methods can be judged for accuracy [32]. The case which is analyzed here is AGARD02, the transonic flow past a NACA 0012 airfoil, with a freestream Mach number of 0.85 and an angle of attack of 1^0. Table 9.13, taken from reference [23], lists lift, drag and moment coefficients of solutions submitted to the Working Group by various contributors. The table also lists the type of grid used, the grid size, and the distance (in chords) of the outer boundary from the airfoil. Unfortunately the results give no clear consensus on the correct solution to the problem, but further developments in the next year or two will hopefully resolve much of the discrepancy between the different solutions. The latest results of Pulliam, listed as #10 in Table 9.13, are probably the most accurate, given the fine grid resolution and the quality of the solutions displayed in reference [23].

Table 9.13: Solutions of AGARD02

Sol. #	type	mesh size	O.B. Dist.	C_L	C_D	C_M
1	C	193 x 29	10/14/10	0.3405	0.0464	-0.0951
2	C	188 x 24	4/6/7	0.3436	0.0541	-0.1093
3	O	158 x 23	16	0.3637	0.0556	-0.1209
5	C	249 x 67	48/96/96	0.3889	0.0590	-0.1378
6	O	192 x 39	50	0.3472	0.0557	-0.1167
8	O	128 x 28	5	0.3300	0.0528	-0.1040
9	O	320 x 64	25	0.3584	0.0580	-0.1228
10	C	560 x 65	24/48/48	0.3938	0.0604	-0.1393

Figure 9.19a shows the airfoil geometry with the converged 113x32 grid, which extends two chord lengths upstream and downstream, and ten chords in the normal direction. Normal grid lines have been clustered in the region of the shocks on the pressure and suction surfaces. Figures 9.19b and 9.19c show contours of Mach number and stagnation density, and Figure 9.20 shows the surface pressure coefficient. The calculated lift, drag and moment coefficients are,

$$C_l=0.3950 \ , \quad C_d=0.0610 \ , \quad C_m=-0.1400$$

which agree very well with Pulliam's results which are,

$$C_l=0.3938 \ , \quad C_d=0.0604 \ , \quad C_m=-0.1393$$

The quality of the agreement with Pulliam may be slightly fortuitous. An estimate of the numerical error in calculating the lift and moment coefficients can be obtained by considering the uncertainty in the position of the shock on the suction surface. The error in the shock position will be at least of the order of the grid spacing, which is 1% of the chord length, and the jump in the pressure coefficient across the shock is approximately 1.0, producing an error estimate of 0.01 for the lift coefficient, and 0.005 for the moment coefficient. Thus the agreement with Pulliam's results is probably better than could be expected.

Table 9.14 lists the iteration history, showing that convergence is achieved in twenty-one iterations. Most of the time is spent in the suction surface shock moving to the correct location. The CPU time per iteration was 3.5 mins.

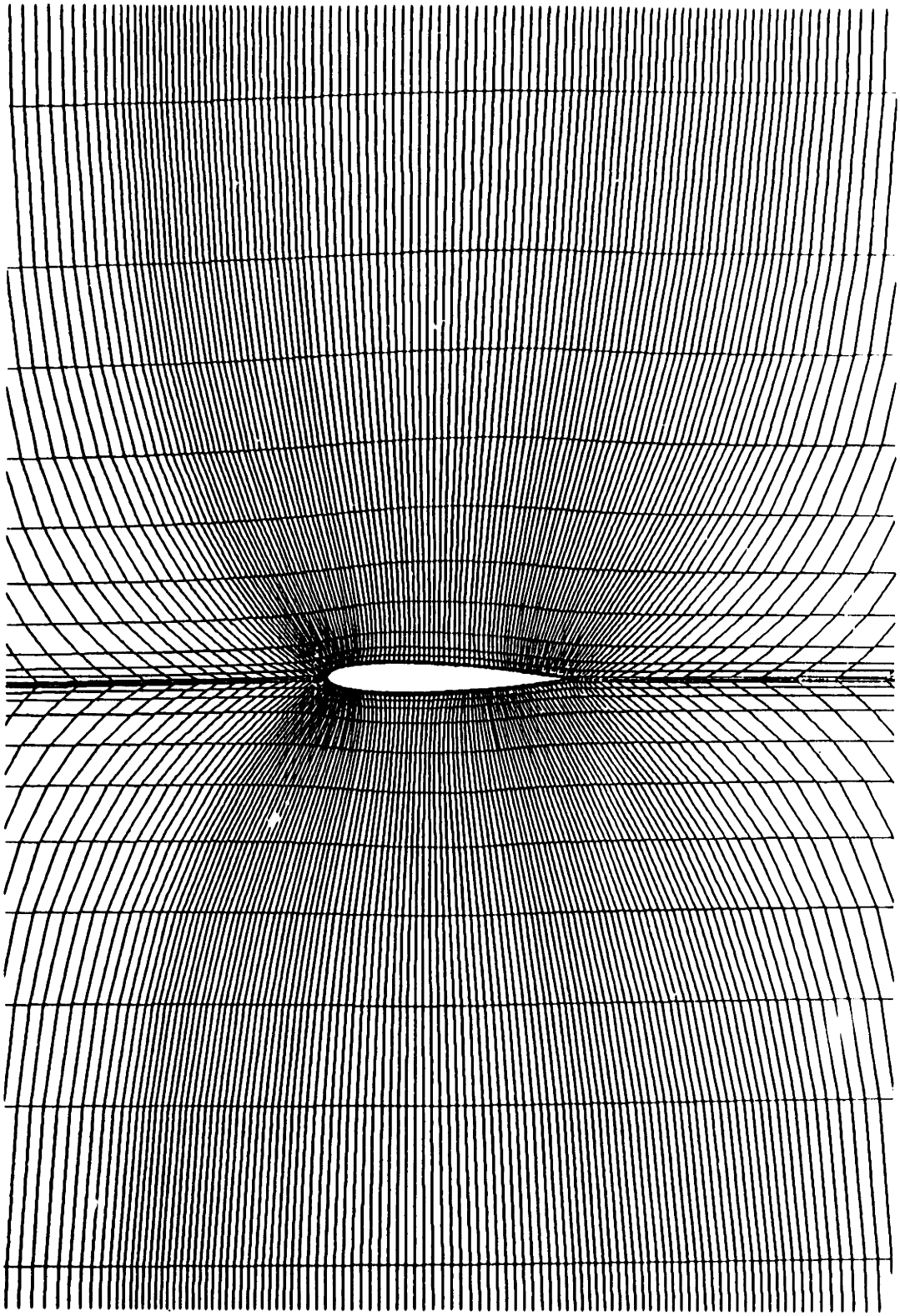


Figure 9.19a: Airfoil and grid geometry for NACA 0012 airfoil.

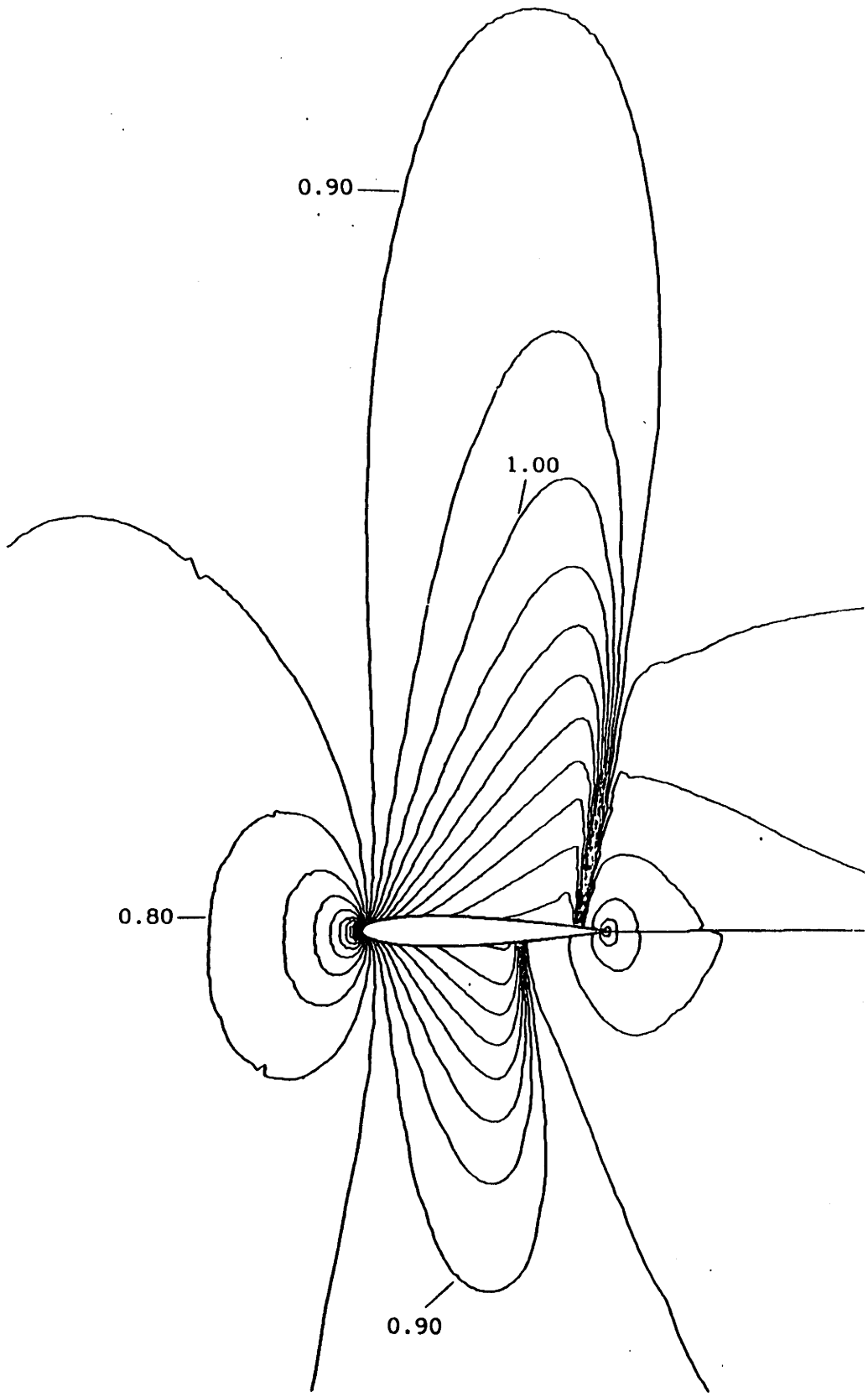


Figure 9.19b: Mach number contours for NACA 0012 airfoil, with increments of 0.1.

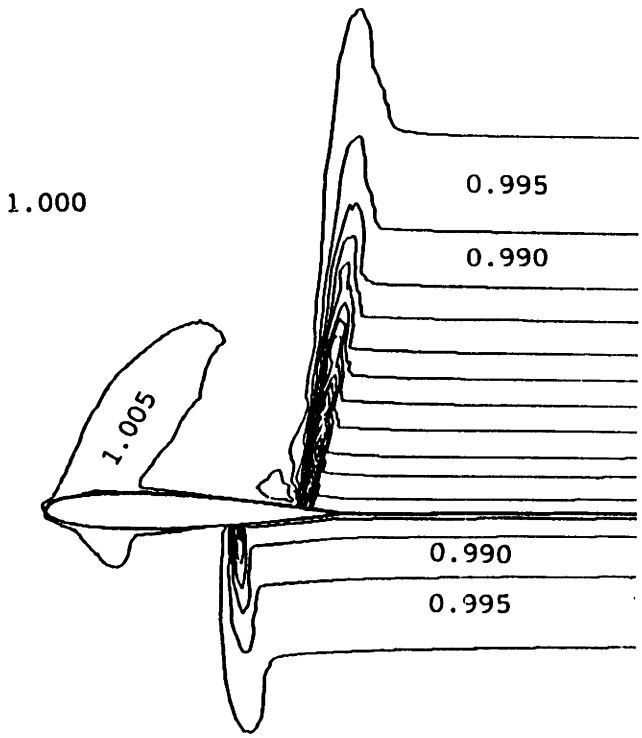


Figure 9.19c: Stagnation density contours for NACA 0012 airfoil, with increments of 0.005.

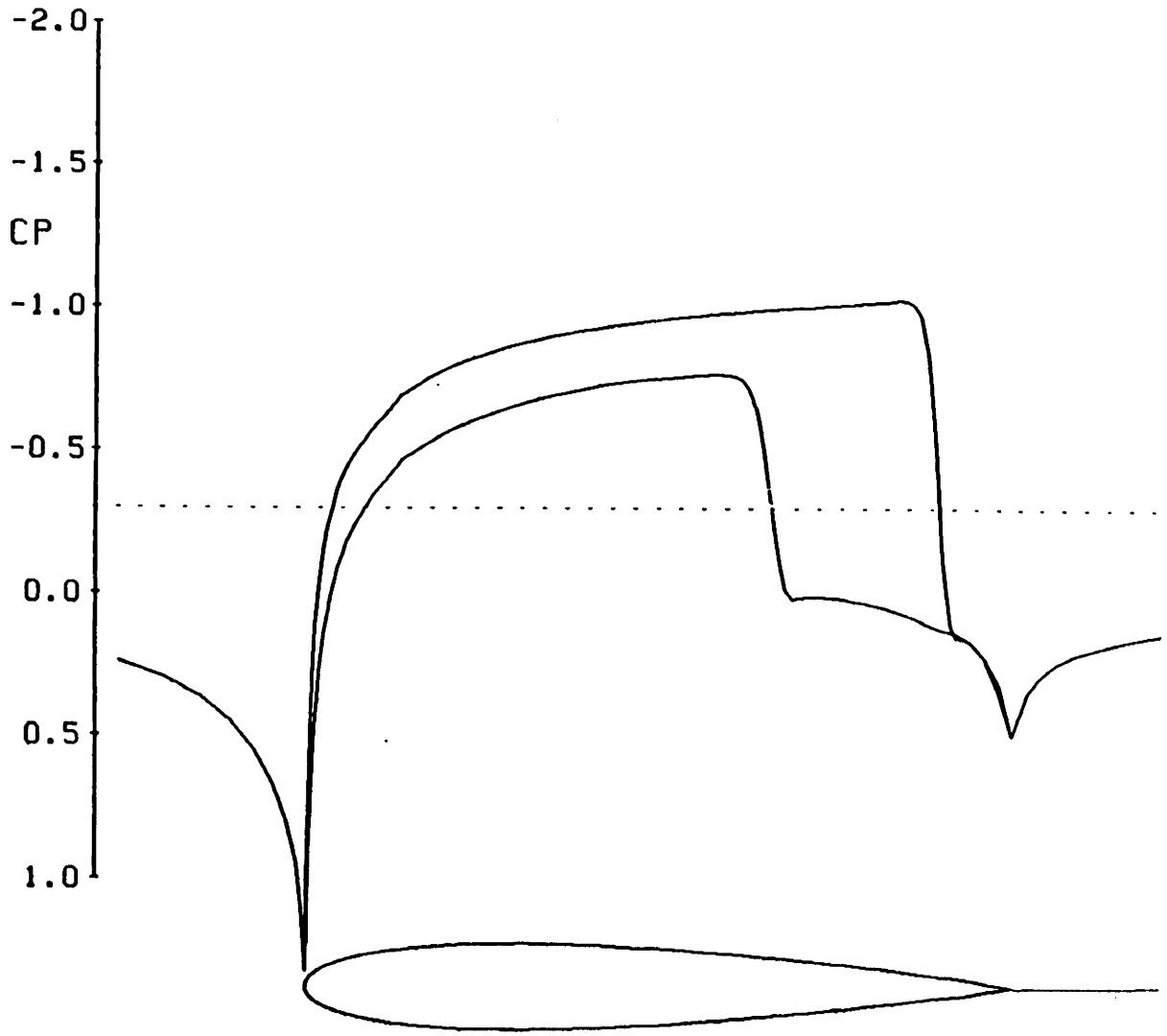


Figure 9.20: Surface pressure coefficients for NACA 0012 airfoil.

Table 9.14: Newton iteration history for NACA 0012 airfoil

Iteration number	δρ/ρ	δn	Comments
1	1.68e-1	3.17e-2	S, RLX=0.75
2	9.60e-2	7.78e-3	S
3	9.84e-2	7.02e-3	RLX=0.49 ↑
4	1.16e-1	6.72e-3	RLX=0.28 : Shock
5	1.37e-1	6.01e-3	RLX=0.20 :
6	1.65e-1	5.98e-3	RLX=0.14 : forming
7	1.88e-1	6.32e-3	RLX=0.13 :
8	1.77e-1	4.98e-3	RLX=0.13 : and moving
9	2.10e-1	6.08e-3	RLX=0.10 :
10	1.52e-1	4.18e-3	RLX=0.15 : downstream
11	1.32e-1	3.83e-3	RLX=0.19 :
12	1.01e-1	3.87e-3	RLX=0.23 : to correct
13	7.17e-2	4.05e-3	RLX=0.34 :
14	5.71e-2	3.25e-3	RLX=0.46 : position
15	4.32e-2	2.15e-3	RLX=0.53 :
16	2.94e-2	1.27e-3	RLX=0.76 ↓
17	1.88e-2	2.67e-4	
18	1.04e-2	3.24e-4	
19	2.89e-3	1.58e-4	
20	3.35e-4	1.92e-5	
21	5.57e-5	8.84e-6	
22	1.97e-5	7.76e-6	
23	1.33e-5	4.26e-6	
24	4.74e-5	5.08e-6	
25	2.74e-5	2.66e-6	

9.7 Two-Dimensional Laval nozzle

The purpose of this test case is to demonstrate the modified direct solver for choked transonic flows. The test geometry is,

Inlet $x=-0.1$

Outlet $x= 1.1$

Upper wall $$y=\begin{cases} 0.2 & x<0 \ , \ x>1 \\ 0.2-0.05\sin^2(\pi x) & 0<x<1 \end{cases}$$

Lower wall $$y=\begin{cases} 0. & x<0 \ , \ x>1 \\ 0.05\sin^2(\pi x) & 0<x<1 \end{cases}$$

Figure 9.21a shows the geometry with a converged 61x11 grid. The flow conditions are,

$h_t=1/(\gamma-1)$, $m_{tot}=0.065$, $\gamma=1.4$

with an average outlet stagnation density of 1.0. Using quasi-one-dimensional Laval nozzle theory the throat area is 0.1, and so the stagnation density at the throat, and hence everywhere upstream of the shock, must be 1.123. Since the average outlet stagnation density is 1.0 this implies that the shock area is 0.126, implying that the shock is at x=0.67, and the fractional change in the outlet stagnation density is 0.110.

Figures 9.21b and 9.21c show contours of the Mach numbers and the stagnation density (normalized by the inlet stagnation density). The dotted lines indicate the analytic position of the shock determined above. Figure 9.22a and 9.22b show the fractional change in stagnation density and the Mach number distribution along one of the center stream-tubes. The results are in good agreement with the quasi-one-dimensional analysis. Table 9.15 lists the Newton iteration history. The CPU time per iteration was 15 secs.

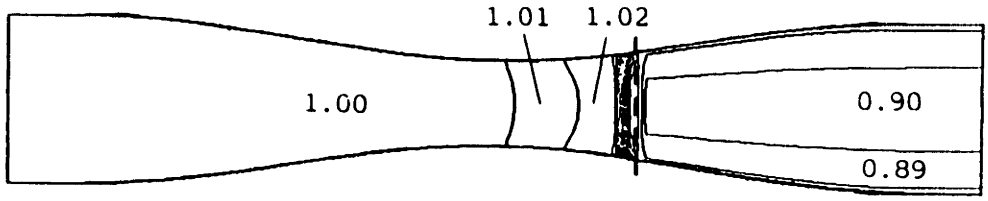


Figure 9.21c: Stagnation density contours for 2-D Laval nozzle flow, with increments of 0.01.

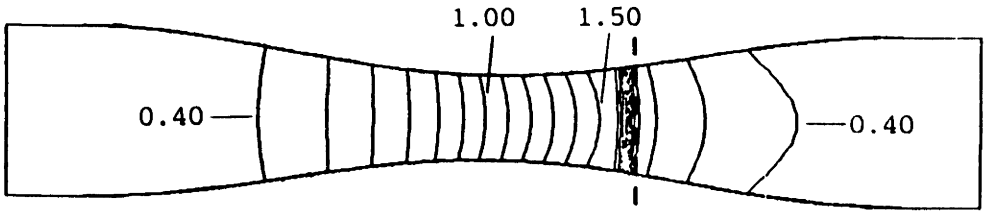


Figure 9.21b: Mach number contours for 2-D Laval nozzle flow, with increments of 0.1.

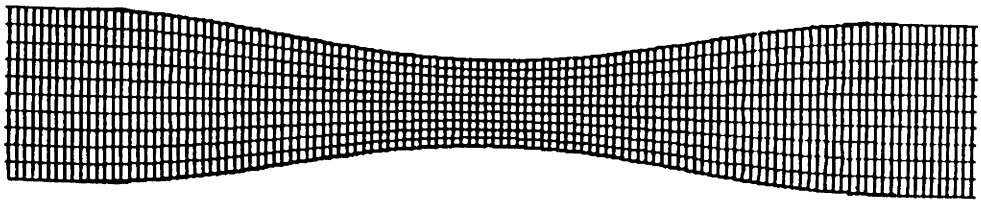


Figure 9.21a: Duct and grid geometry for 2-D Laval nozzle flow.

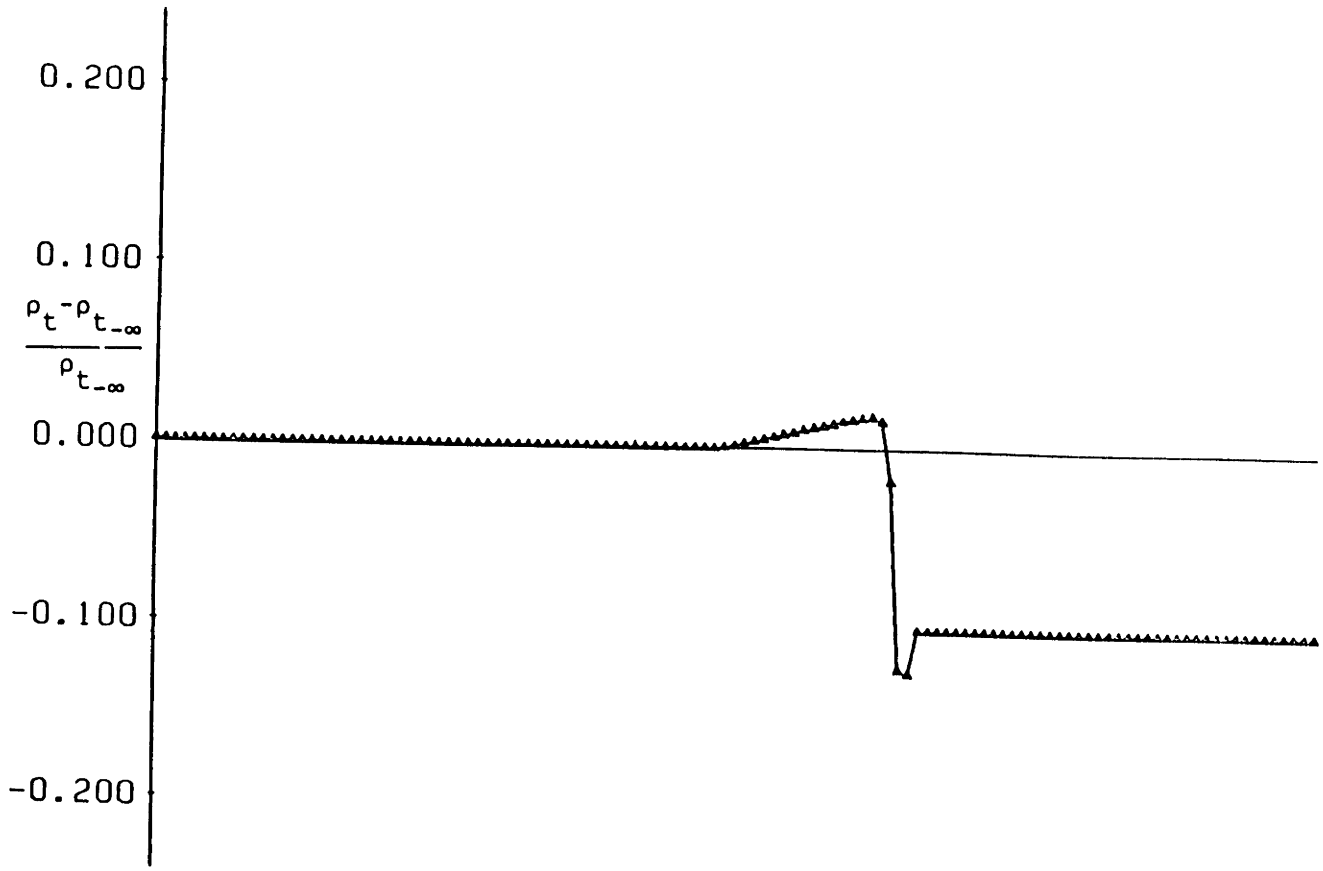


Figure 9.22a: Stagnation density changes on center streamtubes of 2-D Laval nozzle flow.

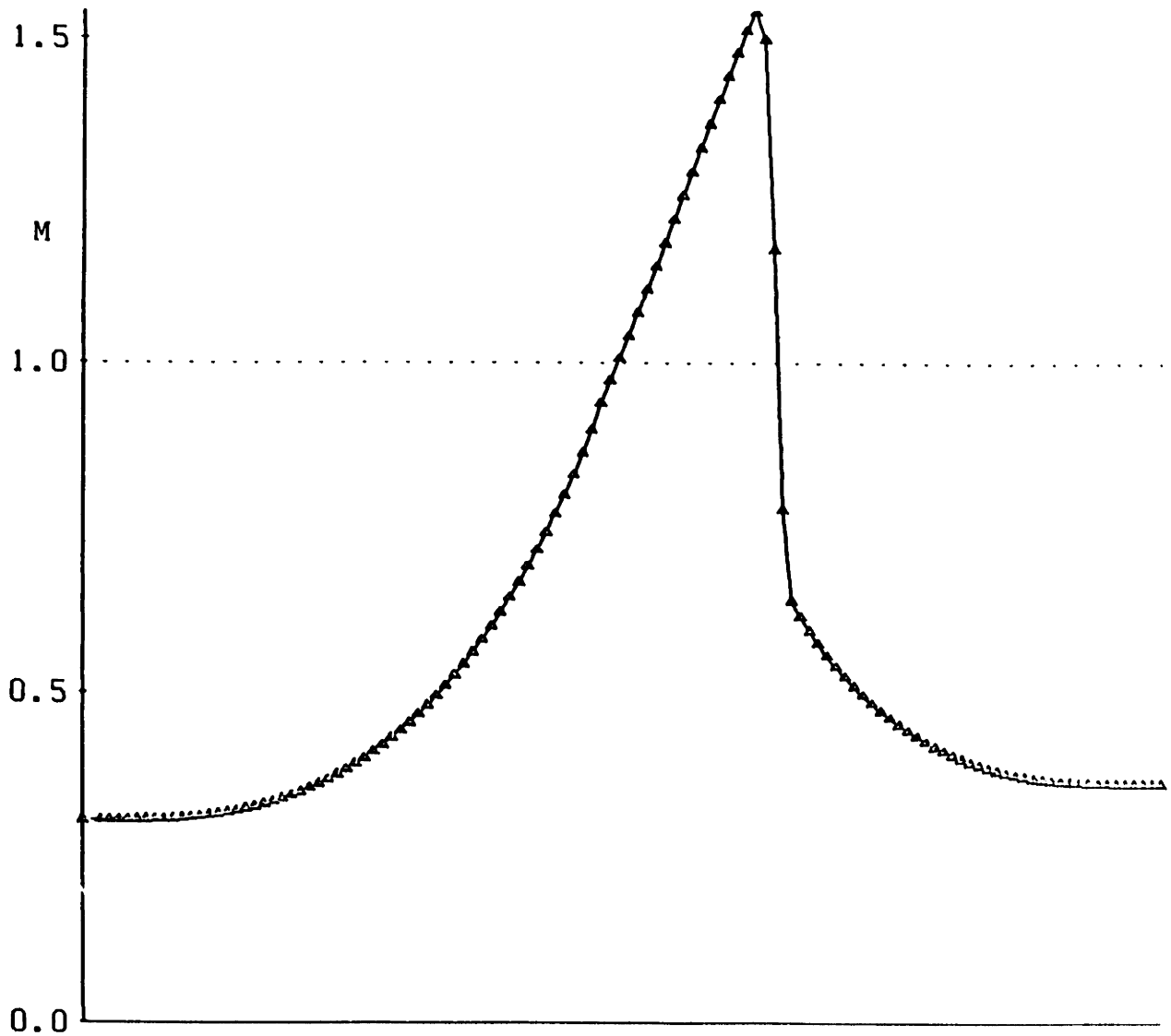


Figure 9.22b: Mach number distribution on center streamtubes of 2-D Laval nozzle flow.

Table 9.15: Newton iteration history for Laval nozzle

Iteration number	$\delta\rho/\rho$	δn	Comments
1	1.36e-1	9.80e-5	
2	1.64e-1	2.68e-4	RLX=0.52
3	1.76e-1	2.00e-4	RLX=0.35
4	1.68e-1	3.39e-4	RLX=0.37
5	1.26e-1	3.24e-4	RLX=0.43
6	8.57e-2	3.61e-4	RLX=0.61
7	5.20e-2	2.26e-4	
8	2.46e-2	3.78e-4	
9	2.39e-3	1.88e-4	
10	2.91e-4	8.16e-6	
11	3.45e-5	6.36e-7	
12	5.35e-6	3.07e-7	
13	3.31e-6	3.51e-7	

10. CONCLUSIONS

10.1 Discretization of Euler Equations

This thesis has presented a novel method of discretizing the steady state transonic Euler equations. The approach used is to apply the integral form of the Euler equations to a four-sided finite volume cell, two sides of which are defined to be streamlines of the flow. This intrinsic grid formulation is different from most current methods for solving the transonic Euler equations, in that it requires the position of the grid nodes to be determined during the calculation procedure. Intrinsic grids have been used with a class of streamline curvature methods, but these methods do not have a conservative formulation because they are based on the differential form of the Euler equations, and so cannot correctly calculate transonic solutions with shocks. The present method however, since it is based on the integral form of the Euler equations, ensures the correct Rankine-Hugoniot shock jump relations, and so will converge to the analytic solution in the limit of infinite grid resolution.

A feature of the thesis is the analysis of the artificial compressibility, which is added to the mass equation to ensure a well-posed solution in the supersonic region. The analysis of linearized perturbations of uniform one-dimensional flow reveals both a minimum level of artificial compressibility required for well-posedness, and an optimum level which ensures sharp shocks. Numerical experiments verify this analysis for a quasi-one-dimensional test case. In addition, a new method of density corrections to obtain second order accuracy is presented and analyzed, and numerical tests demonstrate the resultant decrease in stagnation density errors.

Since the discretization of the Euler equations is new, it has been validated by a series of test cases. The first case, the flow over a $\sin^2(\pi x)$ bump in a duct, verified the second order accuracy of the

discretization for smooth subsonic flow, meaning that the numerical errors are proportional to Δx^2, where Δx is the grid spacing. The second test case is the subsonic flow over an elliptic bump in a duct. One potential problem with an intrinsic grid is a lack of grid resolution at stagnation points, and this test case shows that if one uses streamtubes with equal mass flux there is a loss of accuracy due to the stagnation points, but that if one varies the distribution of the total mass flux between the streamtubes, one can regain the full second order accuracy for subsonic flow. All subsequent calculations of cascade and isolated airfoil flow achieved good grid resolution at leading edge stagnation points by using streamtubes with small mass fluxes. The third test case was incompressible flow over a cascade designed using a conformal transformation method. Excellent agreement with the analytic solution was obtained, verifying the incompressible limit of the discretization and the Kutta condition employed to determine the circulation around each airfoil. The fourth test case was subsonic flow over a turbine cascade with 120^0 turning in flow direction, which tested the ablility of the method to handle grids with extreme shearing. Good agreement was achieved with experimental results. The fifth case was a transonic supercritical cascade designed using a numerical hodograph method. Numerical calculations with the present method demonstrated good agreement using the first order artificial compressibility, and even better agreement using the second order density corrections. The sixth test case is transonic flow over a NACA 0012 isolated airfoil, producing shocks on both the suction and the pressure surfaces. The computed lift, drag and moment coefficients are in very good agreement with the most accurate results in the literature. Finally, the seventh case is choked, transonic flow through a twodimensional Laval nozzle, which verifies the boundary condition formulation for choked flow, and the correct Rankine-Hugoniot shock jump relations.

10.2 Newton Solution Method

A second principal conclusion of this thesis is that Newton's method can be very efficient for solving a large system of nonlinear equations. To a large extent this conclusion is independent of the choice of the particular discretization of the Euler equations employed here, although the current discretization does have the advantage that the resultant linearized system can be reduced to having only two variables per grid node, and so the total number of unknowns is smaller than would be the case for other possible discretizations. The main advantage of Newton's method is that it converges quadratically to the solution of the nonlinear equations, once the approximate solution is close to the true solution. The iteration histories for the subsonic test cases showed convergence to machine accuracy in three to five iterations. The transonic solutions with first order artificial compressibility took longer, with a period of up to ten iterations during which the shock established itself and moved to the correct position, followed by five iterations of near-quadratic convergence to the solution. These additional iterations required for the shock movement are due to the strong inherent nonlinearities associated with a shock, but it is important to note that the algorithm handles this shock movement robustly, aided by the under-relaxation clamp which prevents excessively large changes in either the densities or the position of the leading edge stagnation point. The calculations with the second order density corrections took up to 10 iterations more than the calculations with only the first order artificial compressibility, because of approximations made in the linearization of the nonlinear equations which prevented quadratic convergence and instead led to a terminal convergence in which the residuals decreased by factor 0.5 per iteration.

Another advantage of Newton's method is that it is a very simple, straightforward procedure, at least in principle, and so can be applied to a wide range of problems, for which time-marching or other approaches

may be inappropriate or very inefficient. In practice one must be very careful in performing the linearization and in programming the method, and the final program is more complicated than a time-marching algorithm applied to a simple logically rectangular computational grid. On a more positive note, debugging of the program is eased by the ability to check the linearization of different parts of the equation set independently, and the quadratic terminal convergence rate makes it clear when a part of the program is working correctly, and when it is not. Also, apart from the artificial compressibility required to ensure well-posedness in the supersonic region, no numerical smoothing is required, unlike in time-marching methods in which the choice and implementation of numerical smoothing can critically the stability and accuracy of the method.

An important question to address is whether it is more efficient to solve the linear system of Newton equations by a direct method, or by an iterative method. It can be argued, as in Chapter 7, that for a sufficiently large system of equations an iterative solution method will be more efficient than a direct method based upon a block tri-diagonal algorithm. However, for the test case #4, the incompressible flow past the Gostelow cascade, the total CPU time for the iterative method was 18.7 mins. compared to 7.2 mins. for the direct method. These times were for a grid of 122x23. Now if there is no modification to the iterative procedure then the number of iterations required by the iterative solver increases proportional to $\max(I^2,J^2)$, and the work per iteration scales as IJ, so the total CPU time scales as $\max(I^3J,IJ^3)$. Since the CPU cost of the direct method scales as IJ^3 this implies that the direct method will always be more efficient than the iterative method. If, however, one could accelerate the iterative procedure using a multigrid algorithm, then, as discussed in Chapter 7, the number of iterations would remain fixed as I and J increase. Following this assumption with the numbers above as a baseline, the direct method and the iterative method with multigrid would be equal for a grid in which J=32. This size of grid is more than is necessary to obtain very good results for

cascades and subsonic isolated airfoil flows. A grid of 113x32 was needed for the transonic NACA 0012 isolated airfoil case in order to place the far-field boundary 10 chords away in the normal direction, but in this case a new iterative algorithm would be necessary for the transonic flow and it is unlikely to be as efficient as the subsonic algorithm which was constructed to take advantage of the natural separation of the convective entropy equation from the elliptic pressure equation. Another issue which affects this question is the suitability of the algorithm for vector programming. In the direct method almost all of the CPU time is spent in inverting and multiplying matrices whose size is O(J), and this can very easily be programmed to be done in vector mode on a pipeline machine such as CRAY 1 or a CYBER 205, or on an array processor such as the Floating Point Systems FPS 120B, which is attached to the Perkin-Elmer 3242 on which the current calculations were performed. Using the FPS 120B would decrease the quoted CPU times for the cases in this thesis by approximately a factor of ten. The iterative method could also be vectorized, but with multigrid could require a substantial programming effort, and a full factor ten improvement in speed is unlikely. For these reasons the direct method is considered to be the best method for solving the Newton equations.

10.3 Versatility of Approach

Another important feature of the Newton approach is the inherent versatility of the method. One aspect of this is the global unknowns and global constraints discussed in Chapter 8. In the test cases presented in the Chapter 9, the global unknowns were degrees of freedom in the specification of the far-field boundaries, the normal movement of the inlet and outlet planes for the cascade cases, and the far-field angle of attack and circulation for the isolated airfoil case. The corresponding global constraints were specified inlet flow angle and the trailing edge Kutta condition for the cascade cases, and specified angle of attack and trailing edge Kutta condition for the isolated airfoil case. Time-marching methods can calculate solutions under these conditions, but consider now another useful capability, the ability to specify the lift coefficient of the airfoil instead of the angle of attack. This ability is very useful for comparison with experimental wind tunnel data because the presence of the wind tunnel walls causes an effective change in the angle of attack, which can be estimated only approximately. Thus it is preferable to compare results with a numerical calculation which matches the lift coefficient rather than the corrected angle of attack. To do this with a time-marching method requires a series of calculations at different angles of attack to determine the angle of attack which gives the required lift coefficient. With the Newton method one simply changes the global constraint from specified angle of attack to specifed lift coefficient, and the Newton procedure automatically solves the new problem. Another example of a useful capability is to be able to vary the gap/chord ratio in a cascade to achieve the same flow turning with a different specified amount of loading on each individual blade. This could be accomplished in the current method by introducing a stretching factor in the blade-to-blade direction as an additional global unknown, with the specifed lift coefficient for the blade being an additional global constraint. This option would require substantial additional programming since each of the Euler equations

for all of the cells would have to be linearized with respect to variations in this new global unknown, but this could be done in a straightforward manner and would provide a capability which could only be achieved using a time-marching by a costly series of calculations for different gap/chord ratios.

This discussion would not be complete without reference to the Ph.D. thesis research of Mark Drela, who has extended the method in this thesis to solve two extremely important, and difficult, classes of problem. The first problem is the solution of strongly coupled viscous/inviscid interactions, in which the viscous boundary layer solution is calculated using an integral boundary layer method. The details are available in reference [11], but an important point for the current discussion is that the boundary layer equations, and the relations which couple the boundary layer soltuion to the outer inviscid solution, are simply treated as additional nonlinear equations, and the Newton procedure is then applied to the entire set of equations. This use of the conceptual simplicity of Newton's procedure leads to a method which is robust and more efficient than current methods for coupling time-marching algorithms for the Euler equations to finite difference boundary layer calculations, using strong interaction coupling laws. This also supports the opinion that the direct method is the preferable method for solving the Newton equations, since it is very unclear how one would iteratively solve the resultant set of Newton equations, and even if it were possible it would lose the essential simplicity of the Newton approach. The second class of problems which is solved by Drela is the inverse airfoil problem. The usual analysis problem is to determine the flow and surface pressure distribution for an airfoil of given geometry. The inverse airfoil problem is to determine the flow and the airfoil geometry which corresponds to a specified surface pressure distribution. Again the details are complicated and are presented in reference [11], but there are two points to be noted here. The first is that the ablility to handle a changing airfoil geometry is

inherent in the intrinsic coordinate system which is used for the discretization of the Euler equations. The second is that extensive use is required of global unknowns and constraints to satisfy a number of compatibility requirements, which are handled easily by the Newton procedure, but could not be treated by time-marching methods.

REFERENCES

[1] Anderson, D., Tannehill, J., Pletcher, R., Computational Fluid Mechanics and Heat Transfer, McGraw-Hill, New York, 1984.

[2] Batchelor, G.K., An Introduction to Fluid Dynamics, Cambridge University Press, Cambridge, 1967.

[3] Bauer, F., Garabedian, P.R., Korn, D., Jameson, A., Supercritical Wing Sections II, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, 1975.

[4] Beam, R., Warming, R.F., "An Implicit Finite Difference Algorithm for Hyperbolic Systems in Conservation Law Form," Jour. of Comp. Physics, Vol.22., pp.87-110, September 1976.

[5] Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," Math. of Comp., Vol.31, No.138, pp.333-390, April 1977.

[6] Cebeci, T., Bradshaw, P., Momentum Transfer in Boundary Layers, McGraw-Hill, New York, 1977.

[7] Childs, R.E., Pulliam, T.H., "A Newton Multigrid Method for the Euler Equations" AIAA84-0430, presented at AIAA 22nd Aerospace Sciences Meeting, January 9-12, 1984.

[8] Dahlquist, G., Bjork, A., Anderson, N., Numerical Methods, Prentice-Hall Series in Automatic Computation, New Jersey, 1974.

[9] Drela, M., Giles, M., Thompkins, W.T., "Conservative Streamtube Solution of Steady-State Euler Equations," AIAA-84-1643, presented at AIAA 17th Fluid Dynamics, Plasma Dynamics and Lasers Conference, June 25-27, 1984.

[10] Drela, M., Giles, M., Thompkins, W.T., "Newton Solution of Coupled Euler and Boundary Layer Equations," presented at the Third Symposium on Numerical and Physical Aspects of Aerodynamic Flows, Long Beach, California, January 1985.

[11] Drela, M., "Two-Dimensional Aerodynamic Design and Analysis Using the Euler Equations," Ph.D. thesis, Dept. of Aeronautics and Astronautics, MIT, September 1985.

[12] Eberle, A., "A Method of Finite Volumes for the Calculation of Transonic Potential Flow About Wings Using the Minimum Pressure Integral," translation of MBB-Report UFE 1407(0), February 1978.

[13] Giles, M., "Solution of 1-D Euler Equations using a Box Method,"

MIT CFD Laboratory Report No. CFDL-TR-84-1, February 1984.

[14] Giles, M., Drela, M., Thompkins, W.T., "Newton Solution of Direct and Inverse Transonic Euler Equations," AIAA-85-1530-CP, to be presented at the AIAA 7th Computational Fluid Dynamics Conference, at Cincinnati, Ohio, July 15-17, 1985.

[15] Gostelow, J.P., Cascade Aerodynamics, Pergamon Press, Oxford, 1984.

[16] Hafez, M., South, J., Murman, E., "Artificial Compressibility Methods for Numerical Solutions of Transonic Full Potential Equation," AIAA-7811-84R, AIAA Journal, Vol.17, pp.838-844, August 1979.

[17] Harten, A., "The Artificial Compression Method for Computation of Shocks and and Contact Discontinuities," ICASE Report No. 77-2, February 1977.

[18] Jespersen, D.C., "Recent Developments in Multigrid Methods for the Steady Euler Equations," Lecture Series on Computational Fluid Dynamics at Von Karman Institute for Fluid Dynamics, March 12-16, 1984.

[19] Liepmann, H.W., Roshko, A., Elements of Gasdynamics, Wiley, New York, 1957.

[20] Mulder, W.A., van Leer, B., "Implicit Upwind Methods for the Euler Equations," AIAA-83-1930, presented at the AIAA 6th Computational Fluid Dynamics Conference, at Danvers, Massachusetts, July 13-15, 1983.

[21] Murman, E.M., Cole, J.D., "Calculation of Plane Steady Transonic Flow," AIAA Journal, Vol.9, pp.114-121, January 1971.

[22] Novak, R.A., "Streamline Curvature Computing Procedures for Fluid-Flow Problems," ASME Paper No.66-WA/GT-3, presented at the Winter Annual Meeting of the American Society of Mechanical Engineers, November 1966.

[23] Pulliam, T.H., Barton, J.B., "Euler Computations of AGARD Working Group 07 Airfoil Test Cases," AIAA-85-0018, presented at the AIAA 23rd Aerospace Sciences Meeting, at Reno, Nevada, January 14-17, 1985.

[24] Salas, M.D., Jameson, A., Melnik, R.E., "A Comparative Study of the Nonuniqueness Problem of the Potential Equation," AIAA-83-1888, presented at the AIAA 6th Computational Fluid Dynamics Conference, at Danvers, Massachusetts, July 13-15, 1983.

[25] Shapiro, A.H., Compressible Fluid Flow, Volume I, Wiley, New York, 1953.

[26] Thompson, J.F., Thames, F.C., Mastin, C.W., "Automatic Numerical Generation of Body-Fitted Curvilinear Coordinate System for Field Containing Any Number of Arbitrary Two-Dimensional Bodies," Jour. of Comp. Physics, Vol.15, pp.299-319, 1974.

[27] Thompkins, W.T., Tong, S.S., Bush, R.H., Usab, W.J., Norton, R.J., "Solution Procedures for Accurate Numerical Simulations of Flow in Turbomachinery Cascades," AIAA-83-0257, presented at the AIAA 21st Aerospace Sciences Meeting, January 10-13, 1983.

[28] Wornom, S.F., "Application of Two-point Difference Schemes to the Conservative Euler Equations for One-dimensional Flows," NASA TM-83262, May 1982.

[29] Wornom, S.F., "Implicit Conservative Characteristic Modeling Schemes for the Euler Equations - A New Approach," AIAA-83-1939, presented at the AIAA 6th Computational Fluid Dynamics Conference, July 13-15, 1983, at Danvers, Massachusetts.

[30] Wornom, S.F., Hafez, M.M., "Calculation of Quasi-One-Dimensional Flows with Shocks,", AIAA-84-1245, presented at the AIAA/ASME/SAE 20th Joint Propulsion Conference, June 11-13, 1984, at Cincinnati, Ohio.

[31] Wu, C.-H., "Matrix Solution of Compressible Flow on S1 Surface Through a Turbomachine Blade Row With Splitter Vanes or Tandem Blades," ASME Paper No.83-GT-10, presented at the 28th International Gas Turbine Conference and Exhibit, March 27-31, 1983.

[32] Yoshihara, H. (ed.), "Test Cases for Steady Transonic or Supersonic Flows," AGARD FDP WG-07, Report of the AGARD Fluid Dynamics Panel, Working Group 07, January 1983.

Appendix: Program Listing

This program listing is included primarily as an archival record of the program used to obtain the results presented in Chapter 9. This will not be the final version of the program, which is still under development. In addition the program includes all of the routines used currently by M. Drela to solve flow problems with a coupled integral boundary layer analysis and with inverse or mixed boundary conditions, as detailed in [11]. These are too tightly bound into the program structure to be easily removed, so it was decided to present the entire program that was used.

ISET is the program which initializes everything, the grid, the density distribution, the boundary layer variables etc.

```
      PROGRAM ISET
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
      PIE = 4.0*ATAN(1.0)
      GAM = 1.4
C
      CIRC = 0.
      ALFA = 0.
      REYN = 0.
      INITBL = 0
      ICOUNT = 0
      MSF = 0.98
      MUCON = 1.0
C
      XCENT = 0.25
      YCENT = 0.
      BETSQ = 1.0
C
      GM1 = GAM - 1.0
      GP1 = GAM + 1.0
C
      HINL = 1.0/GM1
      RSTOUT = 1.0
C
      TRF = 0.875                ; turbulent temperature recovery factor
      HVIS = 0.35*HINL           ; Sutherland's constant
C
      WRITE(5,*) 'Set up for BL coupling ?   N'
      READ(5,1000) ANS
 1000 FORMAT(A1)
      COUPLE = (ANS.EQ.'Y')
C
      IF(COUPLE) THEN
    5  WRITE(5,*) 'Enter approximate chord Reynolds number:'
       READ(5,*,ERR=5) REC
      ENDIF
C
C
C---- Read in blade coordinates and grid distribution arrays
      CALL READIN
C
C---- Normalize and spline blade
      CALL NORMIT
      CALL SPLNIT
```

```
C
C---- Set up nodes on grid outline and in the interior
      CALL OUTLIN
      DO 10 I=1, II
        CALL NORLIN(I)
   10 CONTINUE
C
C---- Store outer airfoil boundary
      IF(JJJ.NE.0) THEN
C
       DO 20 J = 1, JJ
         YINL(J) = Y(1,J)
         YOUT(J) = Y(II,J)
   20  CONTINUE
C
       DO 30 I = 1, II
         YTOP(I) = Y(I,JJJ)
         YBOT(I) = Y(I,JJJ+1)
   30  CONTINUE
C
      ENDIF
C
C---- Reset blade coordinates to match grid nodes and re-spline
      CALL RESPLI
C
C---- Fix up entire grid
      WRITE(5,*) ' '
      WRITE(5,*) 'Generating grid ...'
      CALL ELLIP(IX,JX,II,JJ,JJJ,X,Y,YPOS,XPOS)
C
C---- Initialize state vector
      DO 50 J=1, JJ-1
        MFRACT(J) = YPOS(J+1) - YPOS(J)
  50  CONTINUE
C
      DO 60 J=1, JJ-1
        H(J) = HINL
        RSTINL(J) = RSTOUT
   60 CONTINUE
C
      DO 70 J=1, JJ
        DO 701 I=1, II
          R(I,J) = RSTOUT
          PI(I,J) = 0.98*HINL*RSTOUT*GM1/GAM
          MU(I,J) = 0.
          MUC(I,J) = 0.
  701   CONTINUE
   70 CONTINUE
C
```

```
      DO 80 J=1, JJ-1
        IF(J.NE.JJJ) CALL QCALC(J)
   80 CONTINUE
C
C---- initialize BL parameters assuming no coupling
      DO 90 I=1, II
        THET(I,1) = 0.
        THET(I,2) = 0.
        DISP(I,1) = 0.
        DISP(I,2) = 0.
        TAU(I,1) = 0.
        TAU(I,2) = 0.
   90 CONTINUE
C
      IF(COUPLE) THEN
       CALL BLINIT
       CALL OFFSET
       WRITE(5,*) ' '
       WRITE(5,*) 'Adjusting grid ...'
       CALL ELLIP(IX,JX,II,JJ,JJJ,X,Y,YPOS,XPOS)
      ENDIF
C
C---- Write everything to disk
      CALL OUTPUT
C
      STOP
      END ; ISET



      SUBROUTINE READIN
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
C---- Read in blade data and find leading edge index
CCC   OPEN(UNIT=3,FILE='BLADE.DAT',STATUS='OLD')
      WRITE(5,*) ' '
      WRITE(5,*) 'Reading in BLADE.xxx ...'
      READ(3,*) SPINL, SPOUT, CHINL, CHOUT, CHWID
C
      IBLE = 0
      READ(3,*) XB(1), YB(1)
      XMIN = XB(1)
      XMAX = XB(1)
      YMIN = YB(1)
      DO 10 IB = 2, 12345
        READ(3,*,END=11) XB(IB),YB(IB)
        IF(XB(IB).EQ.XB(IB-1) .AND. YB(IB).EQ.YB(IB-1)) IBLE = IB-1
        IF(IB.GT.IBX) STOP 'ISET:  IIB > IBX'
```

```
        XMAX = AMAX1(XMAX,XB(IB))
        IF(XMIN.GT.XB(IB)) THEN
         XMIN = XB(IB)
         YMIN = YB(IB)
        ENDIF
  10  CONTINUE
  11  IIB = IB - 1
CCC   CLOSE(UNIT=3)
C
C---- Read in grid distribution arrays
C     OPEN(UNIT=4,FILE='SPOS.DAT',STATUS='OLD')
C
      WRITE(5,*) 'Reading in SPOS.xxx ...'
      DO 20 K=1, 12345
        READ(4,*) SINL(K)
        IF(SINL(K).EQ.999.) GO TO 21
  20  CONTINUE
  21  NINL = K-1
C
      DO 30 K=1, 12345
        READ(4,*) SOUT(K)
        IF(SOUT(K).EQ.999.) GO TO 31
  30  CONTINUE
  31  NOUT = K-1
C
      DO 40 K=1, 12345
        READ(4,*) SG(K,1)
        IF(SG(K,1).EQ.999.) GO TO 41
  40  CONTINUE
  41  NGS = K-1
C
      DO 50 K=1, 12345
        READ(4,*) SG(K,2)
        IF(SG(K,2).EQ.999.) GO TO 51
  50  CONTINUE
  51  NGP = K-1
C
      JJJ = 0
      DO 60 K=1, 12345
        READ(4,*) YPOS(K)
        IF(YPOS(K).EQ.YPOS(K-1)) JJJ = K-1
        IF(YPOS(K).EQ.999.) GO TO 61
  60  CONTINUE
  61  JJ = K-1
C
      PITCH = 0.
      IF(JJJ.EQ.0) PITCH=CHWID
C
CCC   CLOSE(UNIT=4)
```

```
C
      WRITE(5,*) ' '
      IF(IBLE.EQ.0) WRITE(5,*) 'Blunt leading edge'
      IF(IBLE.GT.0) WRITE(5,*) 'Sharp leading edge'
C
C---- Check for compatibility
      NBLD = MIN0(NGS, NGP)
      ND = NGS-NGP
      IF(ND.NE.0) WRITE(5,*) 'Incompatible blade distribution arrays'
      IF(ND.GT.0) WRITE(5,*) 'Suction surface array reduced by ', ND
      IF(ND.LT.0) WRITE(5,*) 'Pressure surface array reduced by ',-ND
C
      RETURN
      END ; READIN



      SUBROUTINE NORMIT
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
C---- Normalize grid distribution arrays
      DO 71 K=2, NINL
  71    SINL(K) = (SINL(K)-SINL(1)) / (SINL(NINL)-SINL(1))
      DO 72 K=2, NOUT
  72    SOUT(K) = (SOUT(K)-SOUT(1)) / (SOUT(NOUT)-SOUT(1))
      DO 73 K=2, NBLD
        SG(K,1) = (SG(K,1)-SG(1,1)) / (SG(NBLD,1)-SG(1,1))
  73    SG(K,2) = (SG(K,2)-SG(1,2)) / (SG(NBLD,2)-SG(1,2))
      DO 75 K=2, JJ
  75    YPOS(K) = (YPOS(K)-YPOS(1)) / (YPOS(JJ)-YPOS(1))
      SINL(1) = 0.
      SOUT(1) = 0.
      SG(1,1) = 0.
      SG(1,2) = 0.
      YPOS(1) = 0.
C
      II = NINL + NBLD + NOUT - 2
      ILE = NINL
      ITE = II-NOUT+1
C
      IF(II.GT.IX) STOP 'IX is too small'
      IF(JJ.GT.JX) STOP 'JX is too small'
      IF(IIB.GT.IBX) STOP 'IBX is too small'
C
C---- Normalize blade and calculate surface arc length array
      SB(1) = 0.
      DO 80 IB = 1, IIB
        XB(IB) = (XB(IB)-XMIN) / (XMAX-XMIN)
```

```
        YB(IB) = (YB(IB)-YMIN) / (XMAX-XMIN)
        IF(IB.EQ.1) GOTO 80
        SB(IB) = SB(IB-1) +
     &       SQRT( (XB(IB)-XB(IB-1))**2 + (YB(IB)-YB(IB-1))**2 )
  80  CONTINUE
      PITCH = PITCH/(XMAX-XMIN)
C
      RETURN
      END ; NORMIT


      SUBROUTINE SPLNIT
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
C---- Spline blade surface(s)
      IF(IBLE.EQ.0) THEN
       CALL SPLINE(XB,XPB,SB,IIB)
       CALL SPLINE(YB,YPB,SB,IIB)
      ELSE
       CALL SPLINE(XB,XPB,SB,IBLE)
       CALL SPLINE(YB,YPB,SB,IBLE)
       IP = IBLE+1
       CALL SPLINE(XB(IP),XPB(IP),SB(IP),IIB-IBLE)
       CALL SPLINE(YB(IP),YPB(IP),SB(IP),IIB-IBLE)
      ENDIF
C
C---- Find leading edge position SBLE
      IF(IBLE.EQ.0) THEN
       DO 82 IB=2, IIB
         DP1 = XPB(IB-1) + SPINL*YPB(IB-1)
         DP2 = XPB(IB)   + SPINL*YPB(IB)
         IF(DP1.LT.0.0 .AND. DP2.GE.0.0) GO TO 83
  82   CONTINUE
       STOP 'Leading edge not found'
  83   DSB = SB(IB) - SB(IB-1)
       SBLE = SB(IB-1) + DSB*DP1/(DP1-DP2)
       XLE = SEVAL(SBLE,XB,XPB,SB,IIB)
       YLE = SEVAL(SBLE,YB,YPB,SB,IIB)
      ELSE
       XLE = XB(IBLE)
       YLE = YB(IBLE)
       SBLE = SB(IBLE)
      ENDIF
      SBLOLD = SBLE
C
      RETURN
      END ; SPLNIT
```

```
      SUBROUTINE OUTLIN
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
C---- Super-marvy manually-triggered, slope calculating feature (!)
      IF(SPINL.EQ.999. .AND. IBLE.EQ.0)
     & STOP 'Must specify inlet slope with blunt leading edge'
      IF(SPINL.EQ.999.)
     & SPINL = 0.5*((YB(IBLE)-YB(IBLE-1)) / (XB(IBLE)-XB(IBLE-1))
     &            + (YB(IBLE)-YB(IBLE+1)) / (XB(IBLE)-XB(IBLE+1)) )
      IF(SPOUT.EQ.999.)
     & SPOUT = 0.5*((YB(2  )-YB(1    )) / (XB(2  )-XB(1    ))
     &            + (YB(IIB)-YB(IIB+1)) / (XB(IIB)-XB(IIB+1)) )
C
C---- check if TE is open
      DXTE = XB(1)-XB(IIB)
      DYTE = YB(1)-YB(IIB)
      DSTE = SQRT(DXTE**2+DYTE**2)
      FLAP = .FALSE.
      IF(DSTE.GT.1.E-4) THEN
       WRITE(5,*) ' '
       WRITE(5,*) 'Trailing edge is open.  Include floating flap ?  Y'
       READ(5,1000) ANS
 1000  FORMAT(A1)
       FLAP = (ANS.NE.'N')
      ENDIF
C
      WRITE(5,*) ' '
      WRITE(5,*) 'II   = ',II,  '   JJ   = ',JJ  ,'   JJJ  = ',JJJ
      WRITE(5,*) 'NINL = ',NINL,'   NBLD = ',NBLD,'   NOUT = ',NOUT
      WRITE(5,*) ' '
      WRITE(5,*) 'SPINL  = ',SPINL,'   SPOUT  = ',SPOUT
      WRITE(5,*) ' '
C
C---- Set inlet stagnation streamline
      DO 90 K=1, NINL
        X1 = XLE + CHINL * (SINL(K) - 1.0)
        Y1 = YLE + (X1-XLE) * SPINL
        X(K,1) = X1
        Y(K,1) = Y1
        X(K,JJ) = X1
        Y(K,JJ) = Y1 + PITCH
  90  CONTINUE
      DYINL = (X(1,1)-XLE)*SPINL
C
      KTE  = 2
      XTE1 = XB(1)
```

```
      YTE1 = YB(1)
      XTE2 = XB(IIB)
      YTE2 = YB(IIB)
C
      IF(FLAP) THEN
       XTE1 = 0.5*(XB(1)+XB(IIB))
       YTE1 = 0.5*(YB(1)+YB(IIB))
       XTE2 = XTE1
       YTE2 = YTE1
       SE = 4.00            ; trailing edge flap length to gap ratio
       TEP1 = (YPB(1) - SPOUT*XPB(1)) / (XPB(1) + SPOUT*YPB(1))
       A1 = 1.5 + SE*TEP1
       B1 =-1.0 - SE*TEP1
       TEP2 = (YPB(IIB) - SPOUT*XPB(IIB)) / (-XPB(IIB) - SPOUT*YPB(IIB))
       A2 = 1.5 + SE*TEP2
       B2 =-1.0 - SE*TEP2
       DO 92 KTE=2, NOUT
         I = II-NOUT+KTE
         Z = SOUT(KTE)*CHOUT*SQRT(1.0+SPOUT**2)
         IF(Z.GT.SE*DSTE) GO TO 94
          ZB = 1.0 - Z/(SE*DSTE)
          T1 = (A1 + B1*ZB)*ZB**2
          T2 = (A2 + B2*ZB)*ZB**2
          X(I,1)  = XTE1 + CHOUT*SOUT(KTE)        + T1*DXTE
          Y(I,1)  = YTE1 + CHOUT*SOUT(KTE)*SPOUT + T1*DYTE
          X(I,JJ) = XTE2 + CHOUT*SOUT(KTE)        - T2*DXTE
          Y(I,JJ) = YTE2 + CHOUT*SOUT(KTE)*SPOUT - T2*DYTE + PITCH
   92  CONTINUE
      ENDIF
C
C---- Set outlet stagnation streamline
   94 DO 95 K=KTE, NOUT
        I = II-NOUT+K
        X(I,1)  = XTE1 + CHOUT*SOUT(K)
        Y(I,1)  = YTE1 + CHOUT*SOUT(K)*SPOUT
        X(I,JJ) = XTE2 + CHOUT*SOUT(K)
        Y(I,JJ) = YTE2 + CHOUT*SOUT(K)*SPOUT + PITCH
  95  CONTINUE
C
C---- Blade surface lengths
      SS = SBLE
      SP = SB(IIB)-SBLE
C
C---- Set points on blade suction surface
      IBLEN = IBLE
      IF(IBLE.EQ.0) IBLEN = IIB
      DO 100 K=1, NBLD
        I = NINL + K - 1
        S = SBLE - SS*SG(K,1)
```

```
        X(I,1) = SEVAL(S,XB,XPB,SB,IBLEN)
        Y(I,1) = SEVAL(S,YB,YPB,SB,IBLEN)
 100  CONTINUE
C
C---- Set points on blade pressure surface
      IBLEN = IIB-IBLE
      IST = IBLE+1
      IF(IBLE.EQ.0) IBLEN = IIB
      IF(IBLE.EQ.0) IST = 1
      DO 110 K=1, NBLD
        I = NINL + K - 1
        S = SBLE + SP*SG(K,2)
        X(I,JJ) = SEVAL(S,XB(IST),XPB(IST),SB(IST),IBLEN)
        Y(I,JJ) = SEVAL(S,YB(IST),YPB(IST),SB(IST),IBLEN) + PITCH
 110  CONTINUE
C
C---- set up normal line metrics
      SLEN = CHINL*SQRT(1.0+SPINL**2) + CHOUT*SQRT(1.0+SPOUT**2)
     &     + 0.5*SB(IIB)
      DXAVG = SLEN/FLOAT(II-1)
      XPOS(1) = 0.0
      DO 400 I=2, II
        DS = SQRT((X(I,1)-X(I-1,1))**2 + (Y(I,1)-Y(I-1,1))**2)
        DXI = SQRT(SQRT(DXAVG**3*DS))
        XPOS(I) = XPOS(I-1) + DXI
 400  CONTINUE
C
      RETURN
      END  ; OUTLIN



      SUBROUTINE RESPLI
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
C---- Recalculate blade spline data from grid
      I = II-NOUT+1
      IB = 1
      XB(IB) = X(I,1)
      YB(IB) = Y(I,1)
      SB(IB) = 0.0
      DO 142 I=II-NOUT, NINL, -1
        IB = IB+1
        XB(IB) = X(I,1)
        YB(IB) = Y(I,1)
        SB(IB) = SB(IB-1)
     &         + SQRT((XB(IB)-XB(IB-1))**2 + (YB(IB)-YB(IB-1))**2)
 142  CONTINUE
```

```
      IF(IBLE.NE.0) THEN
       IBLE = IB
       IB = IB+1
       XB(IB) = XB(IB-1)
       YB(IB) = YB(IB-1)
       SB(IB) = SB(IB-1)
      ENDIF
      DO 144 I=NINL+1, II-NOUT+1
        IB = IB+1
        XB(IB) = X(I,JJ)
        YB(IB) = Y(I,JJ) - PITCH
        SB(IB) = SB(IB-1)
     &         + SQRT((XB(IB)-XB(IB-1))**2 + (YB(IB)-YB(IB-1))**2)
  144 CONTINUE
      IIB = IB
C
C---- Spline blade surface(s)
      IF(IBLE.EQ.0) THEN
       CALL SPLINE(XB,XPB,SB,IIB)
       CALL SPLINE(YB,YPB,SB,IIB)
      ELSE
       CALL SPLINE(XB,XPB,SB,IBLE)
       CALL SPLINE(YB,YPB,SB,IBLE)
       IP = IBLE+1
       CALL SPLINE(XB(IP),XPB(IP),SB(IP),IIB-IBLE)
       CALL SPLINE(YB(IP),YPB(IP),SB(IP),IIB-IBLE)
      ENDIF
      SBLE = SB(NBLD)
C
      RETURN
      END ; RESPLI



      SUBROUTINE NORLIN(IO)
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
      I = IO
C
      IF(JJJ.NE.0) THEN                ;  Airfoil case
       Y1 = Y(1,1)  + CHWID*YPOS(JJJ)
       Y2 = Y(II,1) + CHWID*YPOS(JJJ)
C
       X(I,JJJ) = -CHINL + (1.+CHINL+CHOUT)*FLOAT(I-1)/FLOAT(II-1)
       Y(I,JJJ) = Y1 + (Y2-Y1)*FLOAT(I-1)/FLOAT(II-1)
       X(I,JJJ+1) = X(I,JJJ)
       Y(I,JJJ+1) = Y(I,JJJ) - CHWID
```

```
C
        XLO = X(I,1)
        YLO = Y(I,1)
        DELX = (X(I,JJJ)-X(I,1)) / YPOS(JJJ)
        DELY = (Y(I,JJJ)-Y(I,1)) / YPOS(JJJ)
        DO 10 J =2, JJJ-1
          X(I,J) = XLO + DELX*YPOS(J)
          Y(I,J) = YLO + DELY*YPOS(J)
   10   CONTINUE
C
        XLO = X(I,JJJ+1)
        YLO = Y(I,JJJ+1)
        DELX = (X(I,JJ)-X(I,JJJ+1)) / (YPOS(JJ)-YPOS(JJJ+1))
        DELY = (Y(I,JJ)-Y(I,JJJ+1)) / (YPOS(JJ)-YPOS(JJJ+1))
        DO 20 J = JJJ+2, JJ
          X(I,J) = XLO + DELX*(YPOS(J)-YPOS(JJJ+1))
          Y(I,J) = YLO + DELY*(YPOS(J)-YPOS(JJJ+1))
   20   CONTINUE
C
       ELSE                                ;  Cascade case
        DO 30 J =2, JJ-1
          X(I,J) = X(I,1) + YPOS(J)*(X(I,JJ)-X(I,1))
          Y(I,J) = Y(I,1) + YPOS(J)*(Y(I,JJ)-Y(I,1))
   30   CONTINUE
       ENDIF
C
       RETURN
       END ; NORLIN



       SUBROUTINE BLINIT
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
C---- get leading edge angle and BetaU
       SB1 = SBLE - 0.001
       SB2 = SBLE + 0.001
       DX1 = -DEVAL(SB1,XB,XPB,SB,IIB)
       DY1 = -DEVAL(SB1,YB,YPB,SB,IIB)
       DX2 =  DEVAL(SB2,XB,XPB,SB,IIB)
       DY2 =  DEVAL(SB2,YB,YPB,SB,IIB)
       COST = (DX1*DX2+DY1*DY2)/SQRT((DX1**2+DY1**2)*(DX2**2+DY2**2))
       THETA = ATAN2((1.0-COST**2), COST)
       BULE = THETA / (2.0*PIE - THETA)
       BULE = AMIN1(1.0,BULE)
       BULE = AMAX1(0.0,BULE)
       BDLE = 0.5*(1.0 - BULE)
       WRITE(5,*) ' '
```

```
      WRITE(5,*) 'Leading edge  x/u du/dx = ', BULE
C
C---- calculate blade arc length arrays
      XI(ILE,1) = 0.
      XI(ILE,2) = 0.
      DO 10 IO=ILE+1, II
        IM = IO-1
        XI(IO,1) = XI(IM,1)
     &           + SQRT((X(IO, 1)-X(IM, 1))**2 + (Y(IO, 1)-Y(IM, 1))**2)
        XI(IO,2) = XI(IM,2)
     &           + SQRT((X(IO,JJ)-X(IM,JJ))**2 + (Y(IO,JJ)-Y(IM,JJ))**2)
   10 CONTINUE
C
C---- Calculate Theta and Dstar distributions using
C     a clamped Thwaites' momentum thickness integral
      BUMIN = -0.18
      CALL TDCALC(1)
      CALL TDCALC(2)
      DSTE = DISP(ITE,1) + DISP(ITE,2)
      WRITE(5,*) ' '
      WRITE(5,*) 'Estimated total Dstar/chord at TE is ',DSTE
      WRITE(5,*) 'Is this reasonable ?  Y'
      READ(5,1000) ANS
 1000 FORMAT(A1)
      IF(ANS.EQ.'N') THEN
       WRITE(5,*) ' '
   48  WRITE(5,*) 'Okay then, enter a reasonable TE Dstar/chord: '
       READ(5,*,ERR=48) DSINP
       WRITE(5,*) ' '
       DO 50 ITRY=1, 10
         DDSDB = 0.5*HH*(DTSQDB(1)/THET(ITE,1) + DTSQDB(2)/THET(ITE,2))
         BUMIN = BUMIN + (DSINP - DSTE)/DDSDB
         BUMIN = AMAX1(BUMIN,-.199)
         CALL TDCALC(1)
         CALL TDCALC(2)
         DSTE = DISP(ITE,1) + DISP(ITE,2)
         WRITE(5,*) 'BUmin,  Dstar/chord = ', BUMIN, DSTE
         IF(ABS(1.0-DSTE/DSINP) .LT. 1.0E-3) GO TO 55
   50  CONTINUE
      ENDIF
C
   55 IF(IBLE.EQ.0) THEN
       THET(ILE,1) = 0.5*(THET(ILE+1,1) + THET(ILE+1,2))
       DISP(ILE,1) = 0.5*(DISP(ILE+1,1) + DISP(ILE+1,2))
       THET(ILE,2) = THET(ILE,1)
       DISP(ILE,2) = DISP(ILE,1)
      ENDIF
C
      RETURN
```

```
      END ; BLINIT



      SUBROUTINE TDCALC(ISINP)
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
      IS = ISINP
      HH = 2.4
      HEXIT = 1.1
C
      J = 1
      IF(IS.EQ.2) J = JJ-1

C---- Set edge velocity array
      DO 10 I=ILE+1, II-1
        UEDG(I,IS) = 0.5*(Q(I-1,J) + Q(I,J))
   10 CONTINUE
      UEDG(II,IS) = UEDG(II-1,IS)
C
C---- Set Ue gradient parameter between 1st and 2nd stations past LE
      BNEXT = ALOG( UEDG(ILE+2,IS)/UEDG(ILE+1,IS) )
     &      / ALOG(   XI(ILE+2,IS)/  XI(ILE+1,IS) )
C
C---- Set initial Ue gradient parameter and constant
      BU = (BULE + 2.0*BNEXT) / 3.0
      UCON = UEDG(ILE+1,IS) / XI(ILE+1,IS)**BU
C
C---- Set initial momentum thickness
      TSQ = 0.45/(REC*UCON*(5.0*BU+1.0)) * XI(ILE+1,IS)**BU
      THET(ILE+1,IS) = SQRT(TSQ)
      DISP(ILE+1,IS) = HH*THET(ILE+1,IS)
C
C---- Integrate Theta integral downstream
      DTSQDB(IS) = 0.0
      DO 30 I=ILE+2, ITE
        XLOG = ALOG( XI(I,IS)/XI(I-1,IS) )
        BU = ALOG( UEDG(I,IS)/UEDG(I-1,IS) ) / XLOG
        BU = AMAX1(BU, BUMIN)
        FBM = 5.0*BU + 1.0
        XEXP = (XI(I,IS)/XI(I-1,IS))**FBM
        UINT = (XEXP*XI(I,IS) - XI(I-1,IS)) * UEDG(I-1,IS)**5 / FBM
        DIDB = 0.0
        IF(BU.EQ.BUMIN) DIDB = -5.0*UINT/FBM
     &   + 5.0*XLOG*XEXP*XI(I,IS)*UEDG(I-1,IS)**5 / FBM
        TSQ         = TSQ         + 0.45*UINT/(REC*UEDG(I,IS)**6)
        DTSQDB(IS) = DTSQDB(IS) + 0.45*DIDB/(REC*UEDG(I,IS)**6)
        THET(I,IS) = SQRT(TSQ)
```

```
        DISP(I,IS) = HH*THET(I,IS)
   30 CONTINUE
C
      DO 35 I=ITE+1, II
        THET(I,IS) = THET(ITE,IS)
        DISP(I,IS) = (HH + (HEXIT-HH)*FLOAT(I-ITE)/FLOAT(II-ITE))
     &             * THET(I,IS)
   35 CONTINUE
C
      RETURN
      END ; TDCALC



      SUBROUTINE OFFSET
$INCLUDE STATE.INC
$INCLUDE ISET.INC
      REAL XOLD(2), YOLD(2)
C
C---- LE Dstar
      DISP(ILE,1) = 0.5 * (DISP(ILE+1,1) + DISP(ILE+1,2))
      IF(IBLE.NE.0) DISP(ILE,1) = 0.0
      DISP(ILE,2) = DISP(ILE,1)
C
C---- save TE-1 coordinates for grid displacement from wake
      XOLD(1) = X(ITE-1,1)
      YOLD(1) = Y(ITE-1,1)
      XOLD(2) = X(ITE-1,JJ)
      YOLD(2) = Y(ITE-1,JJ)
C
C---- set grid edge Dstar away along normal from blade surface
      DO 10 IG=1, NBLD-1
        IO = ILE + IG - 1
        SS = SBLE * (1.0 - SG(IG,1))
        XBLD = SEVAL(SS,XB,XPB,SB,IIB)
        YBLD = SEVAL(SS,YB,YPB,SB,IIB)
        DY  = -DEVAL(SS,XB,XPB,SB,IIB)
        DX  =  DEVAL(SS,YB,YPB,SB,IIB)
        X(IO,1 ) = XBLD + DISP(IO,1) * DX/SQRT(DX*DX + DY*DY)
        Y(IO,1 ) = YBLD + DISP(IO,1) * DY/SQRT(DX*DX + DY*DY)
   10 CONTINUE
      DO 20 IG=1, NBLD-1
        IO = ILE + IG - 1
        SP = SBLE + (SB(IIB) - SBLE) * SG(IG,2)
        XBLD = SEVAL(SP,XB,XPB,SB,IIB)
        YBLD = SEVAL(SP,YB,YPB,SB,IIB) + PITCH
        DY  = -DEVAL(SP,XB,XPB,SB,IIB)
        DX  =  DEVAL(SP,YB,YPB,SB,IIB)
        X(IO,JJ) = XBLD + DISP(IO,2) * DX/SQRT(DX*DX + DY*DY)
```

```
        Y(IO,JJ) = YBLD + DISP(IO,2) * DY/SQRT(DX*DX + DY*DY)
   20 CONTINUE
C
C---- set grid edge Dstar away along wake
      J = 1
      DIR = 1.0
      DO 30 IS=1, 2
        DO 310 IO=ITE, II
          IM = IO-1
          IP = IO+1
          IF(IO.EQ.II) IP = II
          DY  =  X(IP,J) - XOLD(IS)
          DX  = -Y(IP,J) + YOLD(IS)
          XOLD(IS) = X(IO,J)
          YOLD(IS) = Y(IO,J)
          X(IO,J) = XOLD(IS) + DIR*DISP(IO,IS)*DX/SQRT(DX*DX+DY*DY)
          Y(IO,J) = YOLD(IS) + DIR*DISP(IO,IS)*DY/SQRT(DX*DX+DY*DY)
  310   CONTINUE
        J = JJ
        DIR = -1.0
   30 CONTINUE
C
      SWAK = CHOUT*SQRT(1.0 + SPOUT**2)
C
C---- fix up exit streamline positions
      CALL NORLIN(II)
C
      RETURN
      END ; OFFSET



      SUBROUTINE QCALC(J)
$INCLUDE STATE.INC
$INCLUDE ISET.INC
C
      JO = J
      JP = JO + 1
C
      AINL = 0.0
C
C---- Sweep along streamtube
      DO 5 IO = 1, II-1
C
        IP = IO+1
C
        DX = 0.5*(X(IP,JP)+X(IP,JO) - X(IO,JP)-X(IO,JO))
        DY = 0.5*(Y(IP,JP)+Y(IP,JO) - Y(IO,JP)-Y(IO,JO))
        DS2INV = 1.0 / SQRT(DX*DX + DY*DY)
```

```
        SX2 = DX*DS2INV
        SY2 = DY*DS2INV
        AX2 = 0.5*(X(IP,JP)+X(IO,JP) - X(IP,JO)-X(IO,JO))
        AY2 = 0.5*(Y(IP,JP)+Y(IO,JP) - Y(IP,JO)-Y(IO,JO))
        AN2 = SX2*AY2 - SY2*AX2
C
        IF(IO.EQ.1) AINL = AN2
C
        R2 = R(IO,JO)
        Q2 = AINL / (AN2*R2)
        Q(IO,JO) = Q2
C
  5   CONTINUE
C
      RETURN
      END ; QCALC




      SUBROUTINE ELLIP(IMAX,JMAX,II,JJ,JJJ,X,Y,YPOS,XPOS)
      DIMENSION X(IMAX,JMAX),Y(IMAX,JMAX),YPOS(JMAX),XPOS(IMAX)
      DIMENSION C(200),D(2,200)
      CHARACTER*1 ANS
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 JO=2, JJ-1
          JM = JO-1
          JP = JO+1
C
          IF(JO.EQ.JJJ) THEN
           DO 2 IO=2, II-1
             X(IO,JO) = X(IO,JM)
```

```
  2         CONTINUE
            GO TO 5
           ELSE IF(JO.EQ.JJJ+1) THEN
            DO 3 IO=2, II-1
              X(IO,JO) = X(IO,JP)
  3         CONTINUE
            GO TO 5
           ENDIF
C
           DO 6 IO=2, II-1
             IM = IO-1
             IP = IO+1
C
             XMM = X(IM,JM)
             XOM = X(IO,JM)
             XPM = X(IP,JM)
             XMO = X(IM,JO)
             XOO = X(IO,JO)
             XPO = X(IP,JO)
             XMP = X(IM,JP)
             XOP = X(IO,JP)
             XPP = X(IP,JP)
             YMM = Y(IM,JM)
             YOM = Y(IO,JM)
             YPM = Y(IP,JM)
             YMO = Y(IM,JO)
             YOO = Y(IO,JO)
             YPO = Y(IP,JO)
             YMP = Y(IM,JP)
             YOP = Y(IO,JP)
             YPP = Y(IP,JP)
C
             DXIM = XPOS(IO)-XPOS(IM)
             DXIP = XPOS(IP)-XPOS(IO)
             DXIAV = 0.5*(DXIM+DXIP)
C
             DETM = YPOS(JO)-YPOS(JM)
             DETP = YPOS(JP)-YPOS(JO)
             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(IO) =  -ALF*CXIP
            IF(IO.EQ.2) B = 0
C
            D(1,IO) = 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,IO) = 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(IM))
            C(IO) = C(IO) * AINV
            D(1,IO) = ( D(1,IO) - B*D(1,IM) ) * AINV
            D(2,IO) = ( D(2,IO) - B*D(2,IM) ) * AINV
C
   6      CONTINUE
C
          D(1,II) = 0.
          D(2,II) = 0.
C
          IFIN = II-1
          DO 8 IBACK=2, IFIN
            IO = II-IBACK+1
            IP = IO+1
            D( 1,IO) = D( 1,IO) - C(IO)*D(1,IP)
            D( 2,IO) = D( 2,IO) - C(IO)*D(2,IP)
            X(IO,JO) = X(IO,JO) + RLX*D(1,IO)
            Y(IO,JO) = Y(IO,JO) + RLX*D(2,IO)
            AD1 = ABS(D(1,IO))
            AD2 = ABS(D(2,IO))
            DMAX = AMAX1(DMAX,AD1,AD2)
   8      CONTINUE
C
   5    CONTINUE
C
        WRITE(5,*) 'Dmax = ', DMAX, RLX
C
        RLX = RLX1
        IF(DMAX.LT.DSET1) RLX = RLX2
        IF(DMAX.LT.DSET2) RLX = RLX3
        IF(DMAX.LT.DSET3) RETURN
```

```
C
    1  CONTINUE
C
       RETURN
       END ; ELLIP
```

ISES is the main program which solves the Euler equations for transonic flow over cascades or isolated airfoils, with or without a coupled integral boundary layer analysis, and with either direct, inverse or mixed boundary conditions.

```
      PROGRAM ISES
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
      CHARACTER*1 ANS
C
      CALL INIT
C     CALL SMOVE
CCC
      CALL DISSIP
CCC
C
  10  WRITE(5,*) ' '
      WRITE(5,*) 'Enter number of iterations'
      READ(6,*) INEWT
      IF(INEWT.EQ.0) GOTO 90
C
      WRITE(5,*) 'MSF = ',MSF,'  MUCON = ',MUCON
      WRITE(5,*) 'Change dissipation ?  N'
      READ(6,1000) ANS
      IF(ANS.EQ.'Y') THEN
  12   WRITE(5,*) 'Enter new MSF, MUCON: '
       READ(6,*,ERR=12) MSF,MUCON
      ENDIF
C
      DO 40 ITER=ICOUNT+1, ICOUNT+ABS(INEWT)
C
        CALL NCALC
        CALL SETUP
        CALL SETBC
        CALL SOLVE
        CALL UPDATE
        CALL DEKINK
        CALL DISSIP
        IF(LSMOVE) CALL SMOVE
C
  40  CONTINUE
      ICOUNT = ICOUNT+ABS(INEWT)
C
      IF(INEWT.GT.0) GOTO 10
C
  90  DO 95 J=1, JJ-1
        CALL PICALC(J)
```

```
   95  CONTINUE
       CALL LCALC
       CALL OUTPUT
C
 1000  FORMAT(A1)
       STOP
       END ; ISES



       SUBROUTINE INIT
$INCLUDE STATE.INC
$INCLUDE ISES.INC
       REAL XTR1(2),XTR2(2)
C
       CALL INPUT
C
       GM1 = GAM - 1.0
       GP1 = GAM + 1.0
       GCON = GM1 / GAM
       MPPCON = 9.0
C
       DO 14 L=1, NGLX
         KGVAR(L) = 0
         KGCON(L) = 0
    14 CONTINUE
C
C---- read in ISES.xxx
       LDESI = .FALSE.
       LMIXI = .FALSE.
       READ(2,*,END=151) KGVAR
   151 READ(2,*,END=152) KGCON
   152 DO 156 L=1, NGLX
         IF(KGVAR(L).EQ.6) LDESI = .TRUE.
         IF(KGVAR(L).EQ.9) LMIXI = .TRUE.
         IF(KGVAR(L).EQ.0 .OR. KGCON(L).EQ.0) GO TO 159
   156 CONTINUE
       STOP 'Too many global variables, increase NGLX'
   159 NGLOB = L-1
       NRHS = L
C
       READ(2,*) MASSIN, LIFTIN, SINLIN, SOUTIN, ALFAIN
       READ(2,*) ISMOM, PCWT
       READ(2,*) REYNIN,XTR1(1),XTR2(1),XTR1(2),XTR2(2)
       READ(2,*) BULE
C
       LCASC = JJJ .EQ. 0
       LAIRF = .NOT. LCASC
```

```
      LVISC = REYNIN .NE. 0.0
      LANAL =  NOT.LDESI
C
      WRITE(5,*) ' '
      IF(LCASC) WRITE(5,*) 'CASCADE  option'
      IF(LAIRF) WRITE(5,*) 'AIRFOIL  option'
      IF(LANAL) WRITE(5,*) 'ANALYSIS mode'
      IF(LDESI) WRITE(5,*) 'DESIGN   mode'
      IF(LMIXI) WRITE(5,*) '     ... with freewall segment'
      IF(LVISC) WRITE(5,*) 'Boundary Layer coupling included'
C
      WRITE(5,*) 'Changing mass  from ',MASS, ' to ',MASSIN
      MASS = MASSIN
      IF(LVISC) WRITE(5,*) 'Changing Re from ',REYN,' to ',REYNIN
      REYN = REYNIN
C
      LNINL = 0
      LNOUT = 0
      LCIRC = 0
      LALFA = 0
      LSBLE = 0
      LPDF0 = 0
      LPDF1 = 0
      LPDFL = 0
      LPDX0 = 0
      LPDX1 = 0
C
C---- Assign righthand side column numbers to global iterates
C
      WRITE(5,*) ' '
      WRITE(5,*) 'Active global variables ...'
C
      DO 20 L=1, NGLOB
        GOTO (201,202,203,204,205,206,207,208,209,210), KGVAR(L)
C
        WRITE(5,*) 'Illegal variable code:', KGVAR(L)
        STOP
C
  201   LNINL = L + 1
        WRITE(5,*) ' 1  DNINL  inlet plane position'
        GO TO 20
  202   LNOUT = L + 1
        WRITE(5,*) ' 2  DNOUT  outlet plane position'
        GO TO 20
  203   LCIRC = L + 1
        WRITE(5,*) ' 3  DCIRC  far-field vortex strength'
        GO TO 20
  204   LALFA = L + 1
        WRITE(5,*) ' 4  DALFA  angle of attack'
```

```
        GO TO 20
  205   LSBLE = L + 1
        WRITE(5,*) ' 5  DSBLE  LE stagnation point'
        GO TO 20
  206   LPDF0 = L + 1
        WRITE(5,*) ' 6  DPDF0  zeroth moment prescribed Pi DOF'
        GO TO 20
  207   LPDF1 = L + 1
        WRITE(5,*) ' 7  DPDF1  first  moment prescribed Pi DOF'
        GO TO 20
  208   LPDFL = L + 1
        WRITE(5,*) ' 8  DPDFL  lift-setting  prescribed Pi DOF'
        GO TO 20
  209   LPDX0 = L + 1
        WRITE(5,*) ' 9  DPDX0  zeroth mixed inverse prescribed Pi DOF'
        GO TO 20
  210   LPDX1 = L + 1
        WRITE(5,*) '10  DPDX1  first  mixed inverse prescribed Pi DOF'
   20 CONTINUE
C
      WRITE(5,*) ' '
      WRITE(5,*) 'Active global constraints ...'
C
      DO 30 L=1, NGLOB
        GOTO (301,302,303,304,305,306,307,308,309,310,311,312), KGCON(L)
C
        WRITE(5,*) 'Illegal constraint code:', KGCON(L)
        STOP
C
  301   WRITE(5,*) ' 1  Drive inlet  slope from ',SPINL,' to ',SINLIN
        GO TO 30
  302   WRITE(5,*) ' 2  Drive outlet slope from ',SPOUT,' to ',SOUTIN
        GO TO 30
  303   WRITE(5,*) ' 3  Set LE Kutta condition'
        GO TO 30
  304   WRITE(5,*) ' 4  Set TE Kutta condition'
        GO TO 30
  305   WRITE(5,*) ' 5  Drive alpha from ',ALFA,' to ',ALFAIN
        GO TO 30
  306   WRITE(5,*) ' 6  Drive lift from ',LIFT,' to ',LIFTIN
        GO TO 30
  307   WRITE(5,*) ' 7  Drive LE gap to zero'
        GO TO 30
  308   WRITE(5,*) ' 8  Drive TE gap to zero'
        GO TO 30
  309   WRITE(5,*) ' 9  Fix LE point'
        GO TO 30
  310   WRITE(5,*) '10  Fix TE point'
        GO TO 30
```

```
  311   WRITE(5,*) '11  Fix left  endpoint of freewall segment'
        GO TO 30
  312   WRITE(5,*) '12  Fix right endpoint of freewall segment'
C
  30  CONTINUE
C
      DO 40 J=1, JJ-1
        M(J) = MASS * MFRACT(J)
  40  CONTINUE
C
      ILE = NINL
      ITE = II-NOUT+1
C
C---- set up stagnation streamline BC type array
      DO 50 I=1, ILE-1
         NBCTYP(I) = 1                              ; periodic
   50 CONTINUE
C
      DO 51 I=ITE+1, II
         NBCTYP(I) = 1                              ; periodic
         IF(LVISC) NBCTYP(I) = 5                    ; wake
   51 CONTINUE
C
      DO 52 I=ILE, ITE
        IF(LANAL)                NBCTYP(I) = 2  ; hardwall
        IF(LDESI)                NBCTYP(I) = 3  ; freewall
        IF(LANAL.AND.LVISC)      NBCTYP(I) = 4  ; boundary layer
   52 CONTINUE
      IF(LANAL) NBCTYP(ILE) = 2
C
      IF(LMIXI) THEN
C
      IX0 = 0
      IX1 = 0
      DO 54 IS=1, 2
        DO 541 I=ILE+1, ITE-1
          IG = I-ILE+1
          IF(PSPEC(IG,IS).NE.0.0 .AND. PSPEC(IG-1,IS).EQ.0.0) IX0 = I
          IF(PSPEC(IG,IS).NE.0.0 .AND. PSPEC(IG+1,IS).EQ.0.0) THEN
           IX1 = I
           GO TO 59
          ENDIF
  541   CONTINUE
   54 CONTINUE
      STOP 'Couldn''t find freewall segment endpoints'
C
   59 ISMIX = IS
      WRITE(5,*) 'Freewall segment endpoints: ',IX0,IX1,'   side ',IS
      IF(IX0.EQ.0 .OR. IX1.EQ.0) STOP 'Invalid endpoint index'
```

```
      IF(IX0.EQ.IX1) STOP 'Can''t have zero-length segment'
      IF(IX0.GT.IX1) STOP 'Endpoint indices out of order'
C
      ENDIF
C
C---- set intermittency factor arrays
      DO 60 IS=1, 2
        J = 1
        IF(IS.EQ.2) J = JJ
        DO 601 I=1, II
          IF(X(I,J).LE.XTR1(IS)) THEN
           GTR(I,IS) = 0.
          ELSE IF(X(I,J).GE.XTR2(IS)) THEN
           GTR(I,IS) = 1.0
          ELSE
           XT = (2.0*X(I,J) - XTR1(IS)-XTR2(IS))/(XTR2(IS)-XTR1(IS))
           GTR(I,IS) = 0.5 + 0.25*(3.0*XT - XT**3)
          ENDIF
  601   CONTINUE
   60 CONTINUE
C
C---- set arc length arrays for BL equations
      SBLD = SBLE
      DO 70 IS=1, 2
        IF(IS.EQ.2) SBLD = SB(IIB) - SBLE
        DO 701 I=ILE, ITE
          IG = I-ILE+1
          XI(I,IS) = SG(IG,IS)*SBLD
  701   CONTINUE
        DO 702 I=ITE+1, II
          IG = I-ITE+1
          XI(I,IS) = SBLD + SOUT(IG)*SWAK
  702   CONTINUE
   70 CONTINUE
C
C---- set Pi DOF shape functions
      PIE = 4.0*ATAN(1.0)
      DO 80 IG=1, NBLD
        FN0(IG,1) = SIN(PIE*SG(IG,1))
        FN0(IG,2) = SIN(PIE*SG(IG,2))
        FN1(IG,1) = SIN(2.0*PIE*SG(IG,1))
        FN1(IG,2) = SIN(2.0*PIE*SG(IG,2))
   80 CONTINUE
C
      RETURN
      END ; INIT
```

```
      SUBROUTINE NCALC
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
C---- set usual Nhat vectors perpendicular to all streamlines
      DO 10 IO=1, II
        IP = IO+1
        IM = IO-1
        IF(IO.EQ.1 ) IM = 1
        IF(IO.EQ.II) IP = II
        DO 101 JO=1, JJ
          IF(IO.EQ.ITE .AND. (JO.EQ.1 .OR. JO.EQ.JJ)) IM=IO
          IF(IO.EQ.ILE .AND. (JO.EQ.1 .OR. JO.EQ.JJ)) IP=IO
          DY =  X(IP,JO) - X(IM,JO)
          DX = -Y(IP,JO) + Y(IM,JO)
          DSINV = 1.0 / SQRT(DX*DX + DY*DY)
          NX(IO,JO) = DX*DSINV
          NY(IO,JO) = DY*DSINV
 101    CONTINUE
  10  CONTINUE
C
      IF(LDESI) THEN
C----- Full inverse case:  rotate Nhats so that axial chord is preserved
       XLE = X(ILE,1)
       XTE = X(ITE,1)
       DXE = XTE - XLE
       DO 20 IO=1, II
         DO 201 JO=1, JJ
           IF(X(IO,JO).LE.XLE) GO TO 201
            XNORM = AMIN1((X(IO,JO)-XLE)/DXE , 1.0)
            DX = NX(IO,JO)*(1.0-XNORM)
            DY = NY(IO,JO)*(1.0-XNORM) + XNORM
            DSINV = 1.0 / SQRT(DX*DX + DY*DY)
            NX(IO,JO) = DX*DSINV
            NY(IO,JO) = DY*DSINV
  201    CONTINUE
   20  CONTINUE
      ENDIF
C
      IF(LANAL) THEN
C----- Analysis or Mixed-Inverse case:  set LE Nhat parallel to blade
C
       DX = DEVAL(SBLE,XB,XPB,SB,IIB)
       DY = DEVAL(SBLE,YB,YPB,SB,IIB)
       DSINV = 1.0 / SQRT(DX*DX + DY*DY)
       NX(ILE,1) = DX*DSINV
       NY(ILE,1) = DY*DSINV
       NX(ILE,JJ) = NX(ILE,1)
```

```
       NY(ILE,JJ) = NY(ILE,1)
      ENDIF
C
      RETURN
      END ; NCALC



      SUBROUTINE DISSIP
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
      DO 1 JO = 1, JJ-1
C
C------ Calculate MU
C
        I1 = II-1
        I2 = 2
        DO 2 IO = 2, II-1
          MSQ1 = Q(IO-1,JO)**2 / (GM1*(H(JO) - 0.5*Q(IO-1,JO)**2))
          MSQ2 = Q(IO  ,JO)**2 / (GM1*(H(JO) - 0.5*Q(IO  ,JO)**2))
          MSQ = AMAX1(MSQ1,MSQ2)
          MU(IO,JO) = 0.
          MUC(IO,JO) = 0.
          IF(MSQ.LT.MSF) GOTO 2
           MU(IO,JO) = MUCON * (MSQ-MSF) / (GP1*MSQ)
           I1 = MIN0(IO,I1)
           I2 = MAX0(IO,I2)
  2     CONTINUE
C
        K2 = I2-I1
CCC
C       K2 = 0
CCC
        IF(K2.LT.2) RETURN
C
C------ Calculate MUC for second order correction
C       MUC - MPPCON*MUC'' = MU
C
        DO 3 K = 1, K2
          BB(K) = -MPPCON
          AA(K) = 2.0*MPPCON + 1.0
          CC(K) = -MPPCON
          DD(K) = MU(I1+K-1,JO)
          IF(MU(I1+K-1,JO).EQ.0.0 .OR. MU(I1+K,JO).EQ.0.0) THEN
           BB(K) = 0.
           AA(K) = 1.0
           CC(K) = 0.
           DD(K) = 0.
```

```
          ENDIF
  3     CONTINUE
C
        DO 4 K = 1, K2-1
          CC(K) = CC(K) / AA(K)
          DD(K) = DD(K) / AA(K)
          AA(K+1) = AA(K+1) - BB(K+1)*CC(K)
          DD(K+1) = DD(K+1) - BB(K+1)*DD(K)
  4     CONTINUE
C
        DD(K2) = DD(K2) / AA(K2)
C
        DO 5 K = K2-1, 1, -1
          DD(K) = DD(K) - CC(K) * DD(K+1)
  5     CONTINUE
C
        DO 6 K = 1, K2
          MUC(I1+K-1,JO) = AMIN1(DD(K),MU(I1+K-1,JO))
  6     CONTINUE
C
  1   CONTINUE

      RETURN
      END ; DISSIP



C
      SUBROUTINE INPUT
$INCLUDE STATE.INC
      DIMENSION STATE(1)
      EQUIVALENCE (X(1,1),STATE(1))
      CALL DISGET(1,STATE,NSTATE)
      RETURN
      END


      SUBROUTINE OUTPUT
$INCLUDE STATE.INC
      DIMENSION STATE(1)
      EQUIVALENCE (X(1,1),STATE(1))
      REWIND 1
      CALL DISPUT(1,STATE,NSTATE)
      RETURN
      END
```

```
      SUBROUTINE DISGET(LU,BUF,IWCNT)
      DIMENSION IB(5)
      INTEGER*2 IB2(10)
      DIMENSION BUF(1)
      EQUIVALENCE (IB,IB2)
C
      ICNT=4*IWCNT
      CALL SYSIO(IB,Y'00000049',LU,BUF,ICNT,0)
      IF(IB2(2) .EQ. 0) RETURN
      WRITE(5,1000) IB2(2),IB2(1)
 1000 FORMAT(1X,'**ERROR** DISK STATUS =',2Z5,' (DISGET)')
      STOP
      END


      SUBROUTINE DISPUT(LU,BUF,IWCNT)
      DIMENSION IB(5)
      INTEGER*2 IB2(10)
      DIMENSION BUF(1)
      EQUIVALENCE (IB,IB2)
C
      ICNT=4*IWCNT
      CALL SYSIO(IB,Y'00000029',LU,BUF,ICNT,0)
      IF(IB2(2) .EQ. 0) RETURN
      WRITE(5,1000) IB2(2),IB2(1)
 1000 FORMAT(1X,'**ERROR** DISK STATUS =',2Z5,' (DISPUT)')
      STOP
      END
```

```
C
      SUBROUTINE PICALC(J)
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
      JO = J
      JP = JO + 1
C
      IO = 1
      IP = 2
C
      DX = 0.5*(X(IP,JP)+X(IP,JO) - X(IO,JP)-X(IO,JO))
      DY = 0.5*(Y(IP,JP)+Y(IP,JO) - Y(IO,JP)-Y(IO,JO))
      DS1INV = 1.0 / SQRT(DX*DX + DY*DY)
      SX1 = DX*DS1INV
```

```
      SY1 = DY*DS1INV
      AX1 = 0.5*(X(IP,JP)+X(IO,JP) - X(IP,JO)-X(IO,JO))
      AY1 = 0.5*(Y(IP,JP)+Y(IO,JP) - Y(IP,JO)-Y(IO,JO))
      AN1 = SX1*AY1 - SY1*AX1
C
      R0 = R(IO,JO)
      R1 = R(IO,JO)
      RS1 = R(IO,JO)
      Q1 = M(JO)/(AN1*RS1)
      P1 = GCON * R1 * (H(JO) - 0.5*Q1*Q1)
      RQQ1 = R1*Q1*Q1
C
      Q(IO,JO) = Q1
C
C---- Sweep along streamtube
      DO 5 IO = 2, II-1
C
        IM = IO-1
        IP = IO+1
C
        DX = 0.5*(X(IP,JP)+X(IP,JO) - X(IO,JP)-X(IO,JO))
        DY = 0.5*(Y(IP,JP)+Y(IP,JO) - Y(IO,JP)-Y(IO,JO))
        DS2INV = 1.0 / SQRT(DX*DX + DY*DY)
        SX2 = DX*DS2INV
        SY2 = DY*DS2INV
        AX2 = 0.5*(X(IP,JP)+X(IO,JP) - X(IP,JO)-X(IO,JO))
        AY2 = 0.5*(Y(IP,JP)+Y(IO,JP) - Y(IP,JO)-Y(IO,JO))
        AN2 = SX2*AY2 - SY2*AX2
        BXM = 0.5*(X(IP,JO)-X(IM,JO))
        BXP = 0.5*(X(IP,JP)-X(IM,JP))
        BYM = 0.5*(Y(IP,JO)-Y(IM,JO))
        BYP = 0.5*(Y(IP,JP)-Y(IM,JP))
        AXA = AX1*AY2 - AX2*AY1
        SXSM = (X(IO,JO)-X(IM,JO)) * (Y(IP,JO)-Y(IO,JO))
     &       - (Y(IO,JO)-Y(IM,JO)) * (X(IP,JO)-X(IO,JO))
        SXSP = (X(IO,JP)-X(IM,JP)) * (Y(IP,JP)-Y(IO,JP))
     &       - (Y(IO,JP)-Y(IM,JP)) * (X(IP,JP)-X(IO,JP))
C
        XS = 0.5*(BXM+BXP)
        YS = 0.5*(BYM+BYP)
        XN = 0.5*(AX1+AX2)
        YN = 0.5*(AY1+AY2)
        SXNINV = 1.0 / (XS*YN - YS*XN)
        G1 = AN1*(SX1*XN+SY1*YN)*SXNINV
        G2 = AN2*(SX2*XN+SY2*YN)*SXNINV
        MODS = SQRT(XS*XS + YS*YS)
        MODN = SQRT(XN*XN + YN*YN)
        SANA = MODS*MODN
        SXN = (XS*YN - YS*XN) / SANA
```

```
        SDN = (XS*XN + YS*YN) / SANA
C
C------ Calculate flow variables
C
        R2 = R(IO,JO)
        MU2 = MU(IO,JO)
        RS2 = R2 - MU2*(R2-R1) + MUC(IO,JO)*(R1-R0)
        Q2 = M(JO) / (AN2*RS2)
        P2 = GCON * R2 * (H(JO) - 0.5*Q2*Q2)
        RQQ2 = R2*Q2*Q2
C
C------ calculate dP/dA correction and current PI's
C
        MSQ = 0.5 * (RQQ1/P1 + RQQ2/P2) / GAM
        PWT = PCWT
        IF(MSQ.GT.1.0) PWT = 0.0
        PCORR = 0.25*PWT*(P1+P2)*GAM*MSQ*(1.0-MSQ)*(SXSM-SXSP)*SXNINV
        PIDIF = RQQ1*G1 - RQQ2*G2 + PCORR*AXA*SXNINV
        PISUM = P1 + P2 + 2.0*PCORR
        PI(IO,JP) = 0.5*(PISUM + PIDIF)
        PI(IO,JO) = 0.5*(PISUM - PIDIF)
        Q(IO,JO) = Q2
C
        SX1 = SX2
        SY1 = SY2
        AX1 = AX2
        AY1 = AY2
        AN1 = AN2
        DS1INV = DS2INV
C
        RS1 = RS2
        MU1 = MU2
        R0 = R1
        R1 = R2
        Q1 = Q2
        P1 = P2
        RQQ1  = RQQ2
C
  5   CONTINUE
C
      PI(1,JP) = PI(2,JP)
      PI(1,JO) = PI(2,JO)
      PI(II,JP) = PI(II-1,JP)
      PI(II,JO) = PI(II-1,JO)
C
      RETURN
      END ; PICALC
```

```
      SUBROUTINE LCALC
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
C**** It is assumed that PICALC was called for streamtubes 1 & JJ-1 ***
C
      PITE = 0.5*(PI(ITE,1) + PI(ITE,JJ))
C
C---- calculate lift, drag, and moment on blade
      LIFT = 0.
      DRAG = 0.
      MOMN = 0.
      VLIFT = 0.
      VDRAG = 0.
      VMOMN = 0.
C
      DO 10 IO=ILE, ITE
        IM = IO-1
        IP = IO+1
C
        BXS = 0.5*(X(IP, 1)-X(IM, 1))
        BXP = 0.5*(X(IP,JJ)-X(IM,JJ))
        BYS = 0.5*(Y(IP, 1)-Y(IM, 1))
        BYP = 0.5*(Y(IP,JJ)-Y(IM,JJ))
C
        PIS = PI(IO, 1) - PITE
        PIP = PI(IO,JJ) - PITE
        XBARS = X(IO, 1) - XCENT
        XBARP = X(IO,JJ) - XCENT
        YBARS = Y(IO, 1) - YCENT
        YBARP = Y(IO,JJ) - YCENT
        LIFT = LIFT + (BXP*PIP - BXS*PIS)
        DRAG = DRAG + (BYS*PIS - BYP*PIP)
        MOMN = MOMN + (BXS*PIS*XBARS - BXP*PIP*XBARP)
     &              + (BYS*PIS*YBARS - BYP*PIP*YBARP)
        VLIFT = VLIFT + BYS*TAU(IO,1) + BYP*TAU(IO,2)
        VDRAG = VDRAG + BXS*TAU(IO,1) + BXP*TAU(IO,2)
   10 CONTINUE
C
C---- airfoil drag calculations
      IF(JJJ.NE.0) THEN
       IM = II-1
       VDRAG = R(IM,   1)*UEDG(IM,1)**2 * THET(IM,1)
     &       + R(IM,JJ-1)*UEDG(IM,2)**2 * THET(IM,2)
C
       PINF = GCON*R(1,1)*(H(1) - 0.5*Q(1,1)**2)
       QINF = Q(1,1)
       DRAG = 0.0
```

```
      DO 20 JO=1, JJ-1
        IF(JO.EQ.JJJ) GO TO 20
         DO 201 I=1, II-1
           MSQ = Q(I,JO)**2 / (GM1*(H(JO) - 0.5*Q(I,JO)**2))
           IF(MSQ.GE.1.0) GO TO 202
  201     CONTINUE
          GO TO 20
  202      RST2 = R(IM,JO) * (1.0 - 0.5*Q(IM,JO)**2/H(JO))**(-1.0/GM1)
           PST2 = GCON*RST2*H(JO)
           Q2INF = SQRT(2.0*H(JO)*(1.0 - (PINF/PST2)**GCON))
           DRAG = DRAG + (QINF-Q2INF)*M(JO)
   20  CONTINUE
      ENDIF
C
C---- calculate mass-averaged inlet and outlet velocities
      MXINL = 0.
      MYINL = 0.
      MXOUT = 0.
      MYOUT = 0.
      DO 40 JO=1, JJ-1
        JP = JO+1
        DX = 0.5*(X(2 ,JP)+X(2 ,JO) - X(1 ,JP)-X(1 ,JO))
        DY = 0.5*(Y(2 ,JP)+Y(2 ,JO) - Y(1 ,JP)-Y(1 ,JO))
        DS1INV = 1.0 / SQRT(DX*DX + DY*DY)
        SX1 = DX*DS1INV
        SY1 = DY*DS1INV
        DX = 0.5*(X(II,JP)+X(II,JO) - X(II-1,JP)-X(II-1,JO))
        DY = 0.5*(Y(II,JP)+Y(II,JO) - Y(II-1,JP)-Y(II-1,JO))
        DS2INV = 1.0 / SQRT(DX*DX + DY*DY)
        SX2 = DX*DS2INV
        SY2 = DY*DS2INV
C
        R1 = R(1,JO)
        Q1 = Q(1,JO)
        P1 = GCON * R1 * (H(JO) - 0.5*Q1*Q1)
        R2 = R(II-1,JO)
        Q2 = Q(II-1,JO)
        P2 = GCON * R2 * (H(JO) - 0.5*Q2*Q2)
        MXINL = MXINL + M(JO)*SX1
        MYINL = MYINL + M(JO)*SY1
        MXOUT = MXOUT + M(JO)*SX2
        MYOUT = MYOUT + M(JO)*SY2
   40 CONTINUE
C
      MINL = SQRT(MXINL**2 + MYINL**2)
      SXINL = MXINL/MINL
      SYINL = MYINL/MINL
      MOUT = SQRT(MXOUT**2 + MYOUT**2)
      SXOUT = MXOUT/MOUT
```

```
      SYOUT = MYOUT/MOUT
C
      SPINL = SYINL/SXINL
      SPOUT = SYOUT/SXOUT
C
      RETURN
      END ; LCALC
```

SETUP is the subroutine that sets the coefficients of the linearized s-momentum and n-momentum equations for the flow in the interior of the computational region.

```
      SUBROUTINE SETUP
$INCLUDE ISES.INC
C
      REAL NX1M, NX1P, NY1M, NY1P
      REAL NX2M, NX2P, NY2M, NY2P
      REAL NX3M, NX3P, NY3M, NY3P
C
      DO 1 I=1, II
        DO 11 J=1, JJ
          Z1(J,I) = 0.
          Z2(J,I) = 0.
          Z3(J,I) = 0.
          Z4(J,I) = 0.
          Z5(J,I) = 0.
          Z6(J,I) = 0.
          Z7(J,I) = 0.
          Z8(J,I) = 0.
          A1(J,I) = 0.
          A2(J,I) = 0.
          A3(J,I) = 0.
          A4(J,I) = 0.
          A5(J,I) = 0.
          A6(J,I) = 0.
          A7(J,I) = 0.
          A8(J,I) = 0.
          B1(J,I) = 0.
          B2(J,I) = 0.
          B3(J,I) = 0.
          B4(J,I) = 0.
          B5(J,I) = 0.
          B6(J,I) = 0.
          B7(J,I) = 0.
          B8(J,I) = 0.
          C1(J,I) = 0.
          C2(J,I) = 0.
          C3(J,I) = 0.
          C6(J,I) = 0.
          C7(J,I) = 0.
   11   CONTINUE
        DO 12 K=1, 2
          ZT(K,I) = 0.
          AT(K,I) = 0.
          BT(K,I) = 0.
          CT(K,I) = 0.
```

```
            DO 121 L=1, 6
              AI(K,L,I) = 0.
              BI(K,L,I) = 0.
  121       CONTINUE
            DO 122 L=1, 6
              AVC(K,L,I) = 0.
              BVC(K,L,I) = 0.
              ZVC(K,L,I) = 0.
              AVT(K,L,I) = 0.
              BVT(K,L,I) = 0.
              ZVT(K,L,I) = 0.
              AVH(K,L,I) = 0.
              BVH(K,L,I) = 0.
              ZVH(K,L,I) = 0.
  122       CONTINUE
            CVC(K,1,I) = 0.
            CVC(K,2,I) = 0.
            CVT(K,1,I) = 0.
            CVT(K,2,I) = 0.
            CVH(K,1,I) = 0.
            CVH(K,2,I) = 0.
   12     CONTINUE
          DO 13 J=1, 2*JJ+5
            DO 131 L=1, NRHS
              DR(J,L,I) = 0.
  131       CONTINUE
   13     CONTINUE
    1   CONTINUE
C
      AVC(1,4,1 ) = 1
      AVT(1,5,1 ) = 1.
      AVH(1,6,1 ) = 1.
      AVC(2,4,1 ) = 1.
      AVT(2,5,1 ) = 1.
      AVH(2,6,1 ) = 1.
      AVC(1,4,II) = 1.
      AVT(1,5,II) = 1.
      AVH(1,6,II) = 1.
      AVC(2,4,II) = 1.
      AVT(2,5,II) = 1.
      AVH(2,6,II) = 1.
C
C---- Loop over all streamtubes
C
      RMS = 0.
      RMAX = 0.
C
      DO 4 JO = 1, JJ-1
        IF(JO.EQ.JJJ) GOTO 4
```

```
C
      JP = JO + 1
      JZ = JO + JJ
C
      IO = 1
      IP = 2
C
      SX1M = X(IP,JO) - X(IO,JO)
      SX1P = X(IP,JP) - X(IO,JP)
      SY1M = Y(IP,JO) - Y(IO,JO)
      SY1P = Y(IP,JP) - Y(IO,JP)
      DX = 0.5*(X(IP,JP)+X(IP,JO) - X(IO,JP)-X(IO,JO))
      DY = 0.5*(Y(IP,JP)+Y(IP,JO) - Y(IO,JP)-Y(IO,JO))
      DS1INV = 1.0 / SQRT(DX*DX + DY*DY)
      SX1 = DX*DS1INV
      SY1 = DY*DS1INV
      AX1 = 0.5*(X(IP,JP)+X(IO,JP) - X(IP,JO)-X(IO,JO))
      AY1 = 0.5*(Y(IP,JP)+Y(IO,JP) - Y(IP,JO)-Y(IO,JO))
      AN1 = SX1*AY1 - SY1*AX1
C
      NX1M = NX(IO,JO)
      NX1P = NX(IO,JP)
      NY1M = NY(IO,JO)
      NY1P = NY(IO,JP)
      NX2M = NX(IP,JO)
      NX2P = NX(IP,JP)
      NY2M = NY(IP,JO)
      NY2P = NY(IP,JP)
C
      DA1N1M = 0.5*(-(SX1*NY1M-SY1*NX1M)
     &              + (AX1*NY1M-AY1*NX1M
     &          + AN1*(SX1*NX1M+SY1*NY1M))*DS1INV)
      DA1N1P = 0.5*( (SX1*NY1P-SY1*NX1P)
     &              + (AX1*NY1P-AY1*NX1P
     &          + AN1*(SX1*NX1P+SY1*NY1P))*DS1INV)
      DA1N2M = 0.5*(-(SX1*NY2M-SY1*NX2M)
     &              - (AX1*NY2M-AY1*NX2M
     &          + AN1*(SX1*NX2M+SY1*NY2M))*DS1INV)
      DA1N2P = 0.5*( (SX1*NY2P-SY1*NX2P)
     &              - (AX1*NY2P-AY1*NX2P
     &          + AN1*(SX1*NX2P+SY1*NY2P))*DS1INV)
C
      R0 = R(IO,JO)
      R1 = R(IO,JO)
      RS1 = R(IO,JO)
      Q1 = M(JO)/(AN1*RS1)
      Q(IO,JO) = Q1
      P1 = GCON * R1 * (H(JO) - 0.5*Q1*Q1)
      RQQ1 = R1*Q1*Q1
```

```
        MSQ1 = RQQ1/(GAM*P1)
        RST1 = R1 * (1.0 - 0.5*Q1*Q1/H(JO))**(-1.0/GM1)
        IF(JO.EQ.1) BETSQ = 1.0 - MSQ1
C
C------ Calculate sensitivities
C
        DQ1DR0 = 0.
        DQ1DR1 = - Q1/R1
        DQ1DA1 = -Q1/AN1
        DP1DQ1 = -GCON*R1*Q1
        DP1DR1 = P1/R1
C
C*****  Inlet entropy  ************************************
C
C       Z = RSTINL(J) - RSTinlet
C
        DZDQ1 =  RST1 * Q1 / ( GM1 * (H(JO)-0.5*Q1*Q1) )
        DZDR1 =  RST1 / R1
        DZDA1 =  DZDQ1*DQ1DA1
C
        B8(JO,IO) = DZDQ1*DQ1DR0
        A8(JO,IO) = DZDQ1*DQ1DR1 + DZDR1
C
        A6(JO,IO) = DZDA1*DA1N1M
        A7(JO,IO) = DZDA1*DA1N1P
        C6(JO,IO) = DZDA1*DA1N2M
        C7(JO,IO) = DZDA1*DA1N2P
C
        DR(JZ,1,IO) = RSTINL(JO) - RST1
C
C------ Set isentropic sensitivities
C
        DR1DA1 = (R1/AN1) * MSQ1/(1.0-MSQ1)
        DR1N1M(JO,IO) = DR1DA1*DA1N1M
        DR1N1P(JO,IO) = DR1DA1*DA1N1P
        DR1N2M(JO,IO) = DR1DA1*DA1N2M
        DR1N2P(JO,IO) = DR1DA1*DA1N2P
C
C------ Sweep along streamtube setting coefficients
C
        DO 5 IO = 2, II-1
C
          IL = IO-2
          IM = IO-1
          IP = IO+1
C
          NX3M = NX(IP,JO)
          NX3P = NX(IP,JP)
          NY3M = NY(IP,JO)
```

```
      NY3P = NY(IP,JP)
C
      SX2M = X(IP,JO) - X(IO,JO)
      SX2P = X(IP,JP) - X(IO,JP)
      SY2M = Y(IP,JO) - Y(IO,JO)
      SY2P = Y(IP,JP) - Y(IO,JP)
      DX = 0.5*(X(IP,JP)+X(IP,JO) - X(IO,JP)-X(IO,JO))
      DY = 0.5*(Y(IP,JP)+Y(IP,JO) - Y(IO,JP)-Y(IO,JO))
      DS2INV = 1.0 / SQRT(DX*DX + DY*DY)
      SX2 = DX*DS2INV
      SY2 = DY*DS2INV
      AX2 = 0.5*(X(IP,JP)+X(IO,JP) - X(IP,JO)-X(IO,JO))
      AY2 = 0.5*(Y(IP,JP)+Y(IO,JP) - Y(IP,JO)-Y(IO,JO))
      AN2 = SX2*AY2   SY2*AX2
      BXM = 0.5*(X(IP,JO)-X(IM,JO))
      BXP = 0.5*(X(IP,JP)-X(IM,JP))
      BYM = 0.5*(Y(IP,JO)-Y(IM,JO))
      BYP = 0.5*(Y(IP,JP)-Y(IM,JP))
      AXA = AX1*AY2 - AX2*AY1
      BXB = BXM*BYP - BXP*BYM
      SXSM = (SX1M*SY2M - SY1M*SX2M)
      SXSP = (SX1P*SY2P - SY1P*SX2P)
C
      XS = 0.5*(BXM+BXP)
      YS = 0.5*(BYM+BYP)
      XN = 0.5*(AX1+AX2)
      YN = 0.5*(AY1+AY2)
      SXNINV = 1.0 / (XS*YN - YS*XN)
      F1 = AN1*(SX1*XS+SY1*YS)*SXNINV
      F2 = AN2*(SX2*XS+SY2*YS)*SXNINV
      G1 = AN1*(SX1*XN+SY1*YN)*SXNINV
      G2 = AN2*(SX2*XN+SY2*YN)*SXNINV
      D1 = AN1*(SX1*YS-SY1*XS)*SXNINV*DS1INV*0.5
      D2 = AN2*(SX2*YS-SY2*XS)*SXNINV*DS2INV*0.5
      E1 = AN1*(SX1*YN-SY1*XN)*SXNINV*DS1INV*0.5
      E2 = AN2*(SX2*YN-SY2*XN)*SXNINV*DS2INV*0.5
C
      DA2N2M = 0.5*(-(SX2*NY2M-SY2*NX2M)
     &              + (AX2*NY2M-AY2*NX2M
     &          + AN2*(SX2*NX2M+SY2*NY2M))*DS2INV)
      DA2N2P = 0.5*( (SX2*NY2P-SY2*NX2P)
     &              + (AX2*NY2P-AY2*NX2P
     &          + AN2*(SX2*NX2P+SY2*NY2P))*DS2INV)
      DA2N3M = 0.5*(-(SX2*NY3M-SY2*NX3M)
     &              - (AX2*NY3M-AY2*NX3M
     &          + AN2*(SX2*NX3M+SY2*NY3M))*DS2INV)
      DA2N3P = 0.5*( (SX2*NY3P-SY2*NX3P)
     &              - (AX2*NY3P-AY2*NX3P
     &          + AN2*(SX2*NX3P+SY2*NY3P))*DS2INV)
```

```
C
C-------- Calculate flow variables
C
          R2 = R(IO,JO)
          RS2 = R2 - MU(IO,JO)*(R2-R1) + MUC(IO,JO)*(R1-R0)
          Q2 = M(JO) / (AN2*RS2)
          P2 = GCON * R2 * (H(JO) - 0.5*Q2*Q2)
          RQQ2 = R2*Q2*Q2
          MSQ2 = RQQ2/(GAM*P2)
          MU2 = 0.0
          MSQMAX = AMAX1(MSQ1,MSQ2)
          IF(MSQMAX.GT.MSF) MU2 = MUCON * (MSQMAX-MSF) / (GP1*MSQMAX)
          RS2 = R2 - MU2*(R2-R1) + MUC(IO,JO)*(R1-R0)
          Q2 = M(JO) / (AN2*RS2)
          P2 = GCON * R2 * (H(JO) - 0.5*Q2*Q2)
          RQQ2 = R2*Q2*Q2
          MSQ2 = RQQ2/(GAM*P2)
          RST2 = R2 * (1.0 - 0.5*Q2*Q2/H(JO))**(-1.0/GM1)
C
C-------- calculate dP/dA correction and current PI's
C
          MSQ = 0.5 * (MSQ1 + MSQ2)
          PWT = PCWT
          IF(MSQ.GT.1.0) PWT = 0.0
          PCORR = 0.25*PWT*(P1+P2)*GAM*MSQ*(1.0-MSQ)*(SXSM-SXSP)*SXNINV
C         PCORR = PWT*(P1-P2)
          PIDIF = RQQ1*G1 - RQQ2*G2     + PCORR*AXA*SXNINV
          PISUM = P1 + P2 + 2.0*PCORR
          PIP = 0.5*(PISUM + PIDIF)
          PIM = 0.5*(PISUM - PIDIF)
C
          MU(IO,JO) = MU2
          Q(IO,JO) = Q2
          PI(IO,JO) = PIM
          PI(IO,JP) = PIP
C
C-------- Calculate P and Q sensitivities
C
          DM2DR1 = 0.
          DM2DR2 = 0.
          IF(MU2.GT.0.0 .AND. R2.LE.R1)
     &       DM2DR2 = -MUCON*MSF*(2.0+GM1*MSQ2)/(GP1*MSQ2*MSQ2*R2)
          IF(MU2.GT.0.0 .AND. R2.GT.R1)
     &       DM2DR1 = -MUCON*MSF*(2.0+GM1*MSQ1)/(GP1*MSQ1*MSQ1*R1)
          DQ2DR0 = 0.0   ; Q2/RS2 * MUC(IO,JO)
          DQ2DR1 = -Q2/RS2 * (MU2 + MUC(IO,JO) - (R2-R1)*DM2DR1)
          DQ2DR2 = -Q2/RS2 * ((1.-MU2) - (R2-R1)*DM2DR2)
          DQ2DA2 = -Q2/AN2
          DP2DQ2 = -GCON*R2*Q2
```

```
          DP2DR2 = P2/R2
C
C-------- Calculate PCORR sensitivities
C
          DPCDSS = 0.25*PWT*(P1+P2)*GAM*MSQ*(1.0-MSQ)*SXNINV
          DPCDSN = 0.25*PWT*(P1+P2)*GAM*MSQ*(1.0-MSQ)*(SXSM-SXSP)
          DPCDPP = 0.25*PWT*GAM*MSQ*(1.0-MSQ)*(SXSM-SXSP)*SXNINV
          DPCDMA = 0.25*PWT*(P1+P2)*GAM*(1.0-2.*MSQ)*(SXSM-SXSP)*SXNINV
          DPCDQ1 = DPCDPP*DP1DQ1 + DPCDMA*0.5*MSQ1*(2.0 + GM1*MSQ1)/Q1
          DPCDQ2 = DPCDPP*DP2DQ2 + DPCDMA*0.5*MSQ2*(2.0 + GM1*MSQ2)/Q2
          DPCDR0 = DPCDQ2*DQ2DR0 + DPCDQ1*DQ1DR0
          DPCDR1 = DPCDQ2*DQ2DR1 + DPCDQ1*DQ1DR1 + DPCDPP*DP1DR1
          DPCDR2 = DPCDQ2*DQ2DR2                 + DPCDPP*DP2DR2
C         DPCDSS = 0.
C         DPCDSN = 0.
C         DPCDQ1 = PWT*DP1DQ1
C         DPCDQ2 =-PWT*DP2DQ2
C         DPCDR0 = DPCDQ2*DQ2DR0 + DPCDQ1*DQ1DR0
C         DPCDR1 = DPCDQ2*DQ2DR1 + DPCDQ1*DQ1DR1 + PWT*DP1DR1
C         DPCDR2 = DPCDQ2*DQ2DR2                 - PWT*DP2DR2
C
          QDS = 0.25*DPCDSN*SXNINV**2
C
          DPCN1M = -(NX1M*SY2M - NY1M*SX2M) * DPCDSS
     &      + ( NX1M*YN - NY1M*XN - NX1M*YS + NY1M*XS)*QDS
     &      + DPCDQ1*DQ1DA1*DA1N1M
          DPCN1P =  (NX1P*SY2P - NY1P*SX2P) * DPCDSS
     &      + ( NX1P*YN - NY1P*XN + NX1P*YS - NY1P*XS)*QDS
     &      + DPCDQ1*DQ1DA1*DA1N1P
          DPCN2M =  (NX2M*SY2M - NY2M*SX2M) * DPCDSS
     &             +(NX2M*SY1M - NY2M*SX1M) * DPCDSS
     &      + (-NX2M*YS + NY2M*XS)*2.0*QDS
     &      + DPCDQ1*DQ1DA1*DA1N2M + DPCDQ2*DQ2DA2*DA2N2M
          DPCN2P = -(NX2P*SY2P - NY2P*SX2P) * DPCDSS
     &             -(NX2P*SY1P - NY2P*SX1P) * DPCDSS
     &      + ( NX2P*YS - NY2P*XS)*2.0*QDS
     &      + DPCDQ1*DQ1DA1*DA1N2P + DPCDQ2*DQ2DA2*DA2N2P
          DPCN3M = -(NX3M*SY1M - NY3M*SX1M) * DPCDSS
     &      + (-NX3M*YN + NY3M*XN - NX3M*YS + NY3M*XS)*QDS
     &      + DPCDQ2*DQ2DA2*DA2N3M
          DPCN3P =  (NX3P*SY1P - NY3P*SX1P) * DPCDSS
     &      + (-NX3P*YN + NY3P*XN + NX3P*YS - NY3P*XS)*QDS
     &      + DPCDQ2*DQ2DA2*DA2N3P
C
C-------- Set isentropic sensitivities
C
          DR2DA2 = (R2/AN2) * MSQ2/(1.0-MSQ2)
          DR1N1M(JO,IO) = DR2DA2*DA2N2M
          DR1N1P(JO,IO) = DR2DA2*DA2N2P
```

```
          DR1N2M(JO,IO) = DR2DA2*DA2N3M
          DR1N2P(JO,IO) = DR2DA2*DA2N3P
C
         IF(ISMOM.EQ.1) THEN
C*******  S Momentum coefficients  *************************
C
C         Z = P1 - P2 + R1*Q1*Q1*F1 - R2*Q2*Q2*F2     + PCORR*BXB*SXNINV
C
          DZDPC =  BXB*SXNINV
          DZDQ1 =  DP1DQ1 + 2.*R1*Q1*F1
          DZDQ2 = -DP2DQ2 - 2.*R2*Q2*F2
          DZDR1 =  DP1DR1 + Q1*Q1*F1
          DZDR2 = -DP2DR2 - Q2*Q2*F2
          DZDA1 =  DZDQ1*DQ1DA1 + RQQ1*F1/AN1
          DZDA2 =  DZDQ2*DQ2DA2 - RQQ2*F2/AN2
C
          Z8(JO,IO) = DZDQ2*DQ2DR0 + DZDQ1*DQ1DR0          + DZDPC*DPCDR0
          B8(JO,IO) = DZDQ2*DQ2DR1 + DZDQ1*DQ1DR1 + DZDR1 + DZDPC*DPCDR1
          A8(JO,IO) = DZDQ2*DQ2DR2                + DZDR2 + DZDPC*DPCDR2
C
          B6(JO,IO) = DZDA1*DA1N1M                                    ; n1-
     &       + 0.25*SXNINV*(-P1+P2-PIP+PIM)*(XS*NY1M-YS*NX1M)
     &       - RQQ1*D1 * (SX1*NY1M-SY1*NX1M)
     &       + DZDPC*DPCN1M
          B7(JO,IO) = DZDA1*DA1N1P                                    ; n1+
     &       + 0.25*SXNINV*(+P1-P2-PIP+PIM)*(XS*NY1P-YS*NX1P)
     &       - RQQ1*D1 * (SX1*NY1P-SY1*NX1P)
     &       + DZDPC*DPCN1P
          A6(JO,IO) = DZDA1*DA1N2M + DZDA2*DA2N2M                     ; n2-
     &       + 0.25*SXNINV*(-P1+P2-P1+P2  )*(XS*NY2M-YS*NX2M)
     &       + RQQ1*D1 * (SX1*NY2M-SY1*NX2M)
     &       + RQQ2*D2 * (SX2*NY2M-SY2*NX2M)
     &       + DZDPC*DPCN2M
          A7(JO,IO) = DZDA1*DA1N2P + DZDA2*DA2N2P                     ; n2+
     &       + 0.25*SXNINV*(+P1-P2+P1-P2  )*(XS*NY2P-YS*NX2P)
     &       + RQQ1*D1 * (SX1*NY2P-SY1*NX2P)
     &       + RQQ2*D2 * (SX2*NY2P-SY2*NX2P)
     &       + DZDPC*DPCN2P
          C6(JO,IO) = DZDA2*DA2N3M                                    ; n3-
     &       + 0.25*SXNINV*(-P1+P2+PIP-PIM)*(XS*NY3M-YS*NX3M)
     &       - RQQ2*D2 * (SX2*NY3M-SY2*NX3M)
     &       + DZDPC*DPCN3M
          C7(JO,IO) = DZDA2*DA2N3P                                    ; n3+
     &       + 0.25*SXNINV*(+P1-P2+PIP-PIM)*(XS*NY3P-YS*NX3P)
     &       - RQQ2*D2 * (SX2*NY3P-SY2*NX3P)
     &       + DZDPC*DPCN3P
          DR(JZ,1,IO) = P2 - P1 + RQQ2*F2 - RQQ1*F1   - PCORR*BXB*SXNINV
C
         ELSE
```

```
C*******  Entropy continuity coefficients  ****************
C
C         Z = RST1 - RST2
C
          DZDQ1 =  RST1 * R1*Q1/(GAM*P1)
          DZDQ2 = -RST2 * R2*Q2/(GAM*P2)
          DZDR1 =  RST1/R1
          DZDR2 = -RST2/R2
          DZDA1 =  DZDQ1*DQ1DA1
          DZDA2 =  DZDQ2*DQ2DA2
C
          Z8(JO,IO) = DZDQ1*DQ1DR0
          B8(JO,IO) = DZDQ1*DQ1DR1 + DZDQ2*DQ2DR1 + DZDR1
          A8(JO,IO) = DZDQ2*DQ2DR2 + DZDR2
C
          B6(JO,IO) = DZDA1*DA1N1M                          ; n1-
          B7(JO,IO) = DZDA1*DA1N1P                          ; n1+
          A6(JO,IO) = DZDA1*DA1N2M + DZDA2*DA2N2M           ; n2-
          A7(JO,IO) = DZDA1*DA1N2P + DZDA2*DA2N2P           ; n2+
          C6(JO,IO) = DZDA2*DA2N3M                          ; n3-
          C7(JO,IO) = DZDA2*DA2N3P                          ; n3+
C
          DR(JZ,i,IO) = RST2 - RST1
         ENDIF
C
C*******  N Momentum,  part 1  ******************************
C
C         Z = R1*Q1*Q1*G1 - R2*Q2*Q2*G2  + PCORR*AXA*SXNINV  <- part 1
C
C             +/- (P1 + P2 + 2.0*PCORR)                     <- part 2
C
C                    +              -
C           =  2.0*Pi    /   -2.0*Pi
C
          DZDPC = AXA*SXNINV
          DZDQ1 =  2.*R1*Q1*G1
          DZDQ2 = -2.*R2*Q2*G2
          DZDR1 =  Q1*Q1*G1
          DZDR2 = -Q2*Q2*G2
          DZDA1 =  DZDQ1*DQ1DA1 + RQQ1*G1/AN1
          DZDA2 =  DZDQ2*DQ2DA2 - RQQ2*G2/AN2
C
          Z5(JO,IO) =              DZDQ2*DQ2DR0 + DZDQ1*DQ1DR0
     &              + DZDPC*DPCDR0
          Z4(JP,IO) =              DZDQ2*DQ2DR0 + DZDQ1*DQ1DR0
     &              + DZDPC*DPCDR0
          B5(JO,IO) = B5(JO,IO) + DZDQ2*DQ2DR1 + DZDQ1*DQ1DR1 + DZDR1
     &              + DZDPC*DPCDR1
          B4(JP,IO) = B4(JP,IO) + DZDQ2*DQ2DR1 + DZDQ1*DQ1DR1 + DZDR1
```

```
     &               + DZDPC*DPCDR1
          A5(JO,IO) = A5(JO,IO) + DZDQ2*DQ2DR2                  + DZDR2
     &               + DZDPC*DPCDR2
          A4(JP,IO) = A4(JP,IO) + DZDQ2*DQ2DR2                  + DZDR2
     &               + DZDPC*DPCDR2
C
          TMP = DZDA1*DA1N1M                                 + DZDPC*DPCN1M
     &        + 0.25*SXNINV*(-P1+P2-PIP+PIM)*(XN*NY1M-YN*NX1M)
     &        - RQQ1*E1 * (SX1*NY1M-SY1*NX1M)
          B2(JO,IO) = B2(JO,IO) + TMP
          B1(JP,IO) = B1(JP,IO) + TMP
          TMP = DZDA1*DA1N1P                                 + DZDPC*DPCN1P
     &        + 0.25*SXNINV*(+P1-P2-PIP+PIM)*(XN*NY1P-YN*NX1P)
     &        - RQQ1*E1 * (SX1*NY1P-SY1*NX1P)
          B3(JO,IO) = B3(JO,IO) + TMP
          B2(JP,IO) = B2(JP,IO) + TMP
          TMP = DZDA1*DA1N2M + DZDA2*DA2N2M                  + DZDPC*DPCN2M
     &        + 0.25*SXNINV*(-P1+P2-P1+P2  )*(XN*NY2M-YN*NX2M)
     &        + RQQ1*E1 * (SX1*NY2M-SY1*NX2M)
     &        + RQQ2*E2 * (SX2*NY2M-SY2*NX2M)
          A2(JO,IO) = A2(JO,IO) + TMP
          A1(JP,IO) = A1(JP,IO) + TMP
          TMP = DZDA1*DA1N2P + DZDA2*DA2N2P                  + DZDPC*DPCN2P
     &        + 0.25*SXNINV*(+P1-P2+P1-P2  )*(XN*NY2P-YN*NX2P)
     &        + RQQ1*E1 * (SX1*NY2P-SY1*NX2P)
     &        + RQQ2*E2 * (SX2*NY2P-SY2*NX2P)
          A3(JO,IO) = A3(JO,IO) + TMP
          A2(JP,IO) = A2(JP,IO) + TMP
          TMP = DZDA2*DA2N3M                                 + DZDPC*DPCN3M
     &        + 0.25*SXNINV*(-P1+P2+PIP-PIM)*(XN*NY3M-YN*NX3M)
     &        - RQQ2*E2 * (SX2*NY3M-SY2*NX3M)
          C2(JO,IO) = C2(JO,IO) + TMP
          C1(JP,IO) = C1(JP,IO) + TMP
          TMP = DZDA2*DA2N3P                                 + DZDPC*DPCN3P
     &        + 0.25*SXNINV*(+P1-P2+PIP-PIM)*(XN*NY3P-YN*NX3P)
     &        - RQQ2*E2 * (SX2*NY3P-SY2*NX3P)
          C3(JO,IO) = C3(JO,IO) + TMP
          C2(JP,IO) = C2(JP,IO) + TMP
C
C*******  N Momentum, part 2  ******************************
C
          DZDQ1 =  DP1DQ1
          DZDQ2 =  DP2DQ2
          DZDR0 =                    2.0*DPCDR0
          DZDR1 =  DP1DR1          + 2.0*DPCDR1
          DZDR2 =  DP2DR2          + 2.0*DPCDR2
          DZDA1 =  DZDQ1*DQ1DA1
          DZDA2 =  DZDQ2*DQ2DA2
C
```

```
          Z5(JO,IO) = Z5(JO,IO) - DZDQ2*DQ2DR0 - DZDQ1*DQ1DR0 - DZDR0
          Z4(JP,IO) = Z4(JP,IO) + DZDQ2*DQ2DR0 + DZDQ1*DQ1DR0 + DZDR0
          B5(JO,IO) = B5(JO,IO) - DZDQ2*DQ2DR1 - DZDQ1*DQ1DR1 - DZDR1
          B4(JP,IO) = B4(JP,IO) + DZDQ2*DQ2DR1 + DZDQ1*DQ1DR1 + DZDR1
          A5(JO,IO) = A5(JO,IO) - DZDQ2*DQ2DR2                - DZDR2
          A4(JP,IO) = A4(JP,IO) + DZDQ2*DQ2DR2                + DZDR2
C
          TMP = DZDA1*DA1N1M                    + 2.0*DPCN1M
          B2(JO,IO) = B2(JO,IO) - TMP
          B1(JP,IO) = B1(JP,IO) + TMP
          TMP = DZDA1*DA1N1P                    + 2.0*DPCN1P
          B3(JO,IO) = B3(JO,IO) - TMP
          B2(JP,IO) = B2(JP,IO) + TMP
          TMP = DZDA1*DA1N2M + DZDA2*DA2N2M     + 2.0*DPCN2M
          A2(JO,IO) = A2(JO,IO) - TMP
          A1(JP,IO) = A1(JP,IO) + TMP
          TMP = DZDA1*DA1N2P + DZDA2*DA2N2P     + 2.0*DPCN2P
          A3(JO,IO) = A3(JO,IO) - TMP
          A2(JP,IO) = A2(JP,IO) + TMP
          TMP = DZDA2*DA2N3M                    + 2.0*DPCN3M
          C2(JO,IO) = C2(JO,IO) - TMP
          C1(JP,IO) = C1(JP,IO) + TMP
          TMP = DZDA2*DA2N3P                    + 2.0*DPCN3P
          C3(JO,IO) = C3(JO,IO) - TMP
          C2(JP,IO) = C2(JP,IO) + TMP
C
          DR(JO,1,IO) = DR(JO,1,IO) + 2.0*PIM
          DR(JP,1,IO) = DR(JP,1,IO) - 2.0*PIP
C
C-------- set up Ue equation and/or BL equations
          IF(JO.EQ.1 .OR. JO.EQ.JJ-1) THEN
C
           IS = 1
           IF(JO.EQ.JJ-1) IS = 2
           KC = 2*JJ + 3*(IS-1)
C
           IF(ITER.EQ.1) UEDG(IO,IS) = 0.5*(Q1+Q2)
           RES = (Q1+Q2) - 2.0*UEDG(IO,IS)
C
           BVC(IS,4,IO) = 0.
           AVC(IS,4,IO) = -2.0
           BVC(IS,5,IO) = 0.
           AVC(IS,5,IO) = 0.
           BVC(IS,6,IO) = 0.
           AVC(IS,6,IO) = 0.
           DR(KC,1,IO) = -RES
C
           IF(LVISC .AND. IO.GT.ILE) THEN
            CALL SETBL(IO,JO)
```

```
            ELSE
              AVT(IS,5,IO) = 1.0
              AVH(IS,6,IO) = 1.0
            ENDIF
C
          ENDIF
C
          SX1M = SX2M
          SX1P = SX2P
          SY1M = SY2M
          SY1P = SY2P
          SX1 = SX2
          SY1 = SY2
          AX1 = AX2
          AY1 = AY2
          AN1 = AN2
          DS1INV = DS2INV
          NX1M = NX2M
          NX1P = NX2P
          NX2M = NX3M
          NX2P = NX3P
          NY1M = NY2M
          NY1P = NY2P
          NY2M = NY3M
          NY2P = NY3P
C
          RS1 = RS2
          MU1 = MU2
          R0 = R1
          R1 = R2
          Q1 = Q2
          P1 = P2
          RQQ1 = RQQ2
          MSQ1 = MSQ2
          RST1 = RST2
          DQ1DR0 = DQ2DR1
          DQ1DR1 = DQ2DR2
          DQ1DA1 = DQ2DA2
          DP1DQ1 = DP2DQ2
          DP1DR1 = DP2DR2
          DA1N1M = DA2N2M
          DA1N1P = DA2N2P
          DA1N2M = DA2N3M
          DA1N2P = DA2N3P
C
  5     CONTINUE
C
C------ set exit pressure
C       Z8(JO,II) = MFRACT(JO)*DP1DQ1*DQ1DR0
```

```
C       B6(JO,II) = MFRACT(JO)*DP1DQ1*DQ1DA1*DA1N1M
C       B7(JO,II) = MFRACT(JO)*DP1DQ1*DQ1DA1*DA1N1P
C       B8(JO,II) = MFRACT(JO)*(DP1DQ1*DQ1DR1+DP1DR1)
C       A6(JO,II) = MFRACT(JO)*DP1DQ1*DQ1DA1*DA1N2M
C       A7(JO,II) = MFRACT(JO)*DP1DQ1*DQ1DA1*LA1N2P
C       DR(JZ,1,II) = MFRACT(JO)*(PEXIN-P1)
C
C------ set exit stagnation density
C       Z = RST1
C
        DZDQ1 =  RST1 * R1*Q1/(GAM*P1)
        DZDR1 =  RST1/R1
        DZDA1 =  DZDQ1*DQ1DA1
C
        Z8(JO,II) = MFRACT(JO)*DZDQ1*DQ1DR0
        B8(JO,II) = MFRACT(JO)*(DZDQ1*DQ1DR1 + DZDR1)
        B6(JO,II) = MFRACT(JO)*DZDA1*DA1N1M                          ; n1-
        B7(JO,II) = MFRACT(JO)*DZDA1*DA1N1P                          ; n1+
        A6(JO,II) = MFRACT(JO)*DZDA1*DA1N2M                          ; n2-
        A7(JO,II) = MFRACT(JO)*DZDA1*DA1N2P                          ; n2+
C
        DR(JZ,1,II) = MFRACT(JO)*(RSTOUT - RST1)
C
C------ dummy variable coefficient
        A8(JO,II) = 1.0
C
  4   CONTINUE
C
      RETURN
      END ; SETUP
```

SETBC is the subroutine that sets the coefficients of the linearized boundary conditions.

```
      SUBROUTINE SETBC
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
C**** Store blade surface sensitivities ***********
C
      DO 5 IO=1, II
        A1S(IO,1) = A1(1,IO)
        A2S(IO,1) = A2(1,IO)
        A3S(IO,1) = A3(1,IO)
        A4S(IO,1) = A4(1,IO)
        A5S(IO,1) = A5(1,IO)
        B1S(IO,1) = B1(1,IO)
        B2S(IO,1) = B2(1,IO)
        B3S(IO,1) = B3(1,IO)
        B4S(IO,1) = B4(1,IO)
        B5S(IO,1) = B5(1,IO)
        C1S(IO,1) = C1(1,IO)
        C2S(IO,1) = C2(1,IO)
        C3S(IO,1) = C3(1,IO)
        Z4S(IO,1) = Z4(1,IO)
        Z5S(IO,1) = Z5(1,IO)
        A1S(IO,2) = A1(JJ,IO)
        A2S(IO,2) = A2(JJ,IO)
        A3S(IO,2) = A3(JJ,IO)
        A4S(IO,2) = A4(JJ,IO)
        A5S(IO,2) = A5(JJ,IO)
        B1S(IO,2) = B1(JJ,IO)
        B2S(IO,2) = B2(JJ,IO)
        B3S(IO,2) = B3(JJ,IO)
        B4S(IO,2) = B4(JJ,IO)
        B5S(IO,2) = B5(JJ,IO)
        C1S(IO,2) = C1(JJ,IO)
        C2S(IO,2) = C2(JJ,IO)
        C3S(IO,2) = C3(JJ,IO)
        Z4S(IO,2) = Z4(JJ,IO)
        Z5S(IO,2) = Z5(JJ,IO)
    5 CONTINUE
C
C**** Set boundary conditions at each streamwise station ***************
C
      DO 100 IO = 2, II-1
        IM = IO-1
C
        GO TO (10,20,30,40,50), NBCTYP(IO)
```

```
C
C------ Set periodic boundary condition
  10    DELP = 0.0
        Z4(1,IO) = Z4(JJ,IO)
        B1(1,IO) = B1(JJ,IO)
        BT(1,IO) = B2(JJ,IO)
        B4(1,IO) = B4(JJ,IO)
        A1(1,IO) = A1(JJ,IO)
        AT(1,IO) = A2(JJ,IO)
        A4(1,IO) = A4(JJ,IO)
        C1(1,IO) = C1(JJ,IO)
        CT(1,IO) = C2(JJ,IO)
        DR(1,1,IO) = DR(1,1,IO) + DR(JJ,1,IO) + 2.0*DELP
        DO 11 L=2, NRHS
          DR(1,L,IO) = DR(1,L,IO) + DR(JJ,L,IO)
  11    CONTINUE
C
        CALL CLROW(IO,JJ)
        A2(JJ,IO) = -1.0
        AT(2 ,IO) =  1.0
        DO 12 L=1, NRHS
          DR(JJ,L,IO) = 0.
  12    CONTINUE
        GOTO 100
C
C------ Set hard wall condition
  20    CALL CLROW(IO,1)
        CALL CLROW(IO,JJ)
        A2(1,IO) = 1.0
        A2(JJ,IO) = 1.0
        GOTO 100
C
C------ Set Pspec condition
  30    IG = IO-ILE+1
        DR(1 ,1,IO) = DR(1 ,1,IO)
     &      - 2.0*( PSPEC(IG,1)
     &            - PDF0*FN0(IG,1) - PDF1*FN1(IG,1)
     &            - PDFL*FN0(IG,1) )
        DR(JJ,1,IO) = DR(JJ,1,IO)
     &      + 2.0*( PSPEC(IG,2)
     &            - PDF0*FN0(IG,2) - PDF1*FN1(IG,2)
     &            + PDFL*FN0(IG,2) )
        DR(1 ,LPDF0,IO) = -2.0*FN0(IG,1)
        DR(1 ,LPDF1,IO) = -2.0*FN1(IG,1)
        DR(1 ,LPDFL,IO) = -2.0*FN0(IG,1)
        DR(JJ,LPDF0,IO) =  2.0*FN0(IG,2)
        DR(JJ,LPDF1,IO) =  2.0*FN1(IG,2)
        DR(JJ,LPDFL,IO) = -2.0*FN0(IG,2)
        GO TO 100
```

```
C
C------ Set grid edge - Dstar equivalence on wall (direct BL coupling)
  40    CALL CLROW(IO,1)
        A2(1,IO) = 1.0
        BI(1,1,IO) = 0.
        BI(1,2,IO) = 0.
        BI(1,3,IO) = 0.
        AI(1,1,IO) = 0.
        AI(1,2,IO) = 0.
        AI(1,3,IO) =-1.0
C
        CALL CLROW(IO,JJ)
        A2(JJ,IO) = 1.0
        BI(2,4,IO) = 0.
        BI(2,5,IO) = 0.
        BI(2,6,IO) = 0.
        AI(2,4,IO) = 0.
        AI(2,5,IO) = 0.
        AI(2,6,IO) = 1.0
        GO TO 100
C
C------ Set deltaP across wake
   50   DELP = 0.0
        Z4(1,IO) = Z4(JJ,IO)
        B1(1,IO) = B1(JJ,IO)
        BT(1,IO) = B2(JJ,IO)
        B4(1,IO) = B4(JJ,IO)
        A1(1,IO) = A1(JJ,IO)
        AT(1,IO) = A2(JJ,IO)
        A4(1,IO) = A4(JJ,IO)
        C1(1,IO) = C1(JJ,IO)
        CT(1,IO) = C2(JJ,IO)
        DR(1,1,IO) = DR(1,1,IO) + DR(JJ,1,IO) + 2.0*DELP
        DO 51 L=2, NRHS
          DR(1,L,IO) = DR(1,L,IO) + DR(JJ,L,IO)
   51   CONTINUE
C
C------ Set grid gap to Dstar on wake
C       DX = X(IO+1,1) + X(IO+1,JJ) - X(IO-1,1) - X(IO-1,JJ)
C       DY = Y(IO+1,1) + Y(IO+1,JJ) - Y(IO-1,1) - Y(IO-1,JJ)
C       DS = SQRT(DX*DX + DY*DY)
C       DX = -DY/DS
C       DY =  DX/DS
C       DOTP = DX*NX(IO,1) + DY*NY(IO,1)
        CALL CLROW(IO,JJ)
        A2(JJ,IO) = -1.0
        AT(2 ,IO) =  1.0
        BI(2,1,IO) = 0.
        BI(2,2,IO) = 0.
```

```
        BI(2,3,IO) = 0.
        AI(2,1,IO) = 0.
        AI(2,2,IO) = 0.
        AI(2,3,IO) =-1.0
        BI(2,4,IO) = 0.
        BI(2,5,IO) = 0.
        BI(2,6,IO) = 0.
        AI(2,4,IO) = 0.
        AI(2,5,IO) = 0.
        AI(2,6,IO) =-1.0
  100 CONTINUE
C
C**** Leading edge point movement identity ******************************
C
      DR(1 ,LSBLE,ILE) = -1.0
      DR(JJ,LSBLE,ILE) = -1.0
C
C
C**** Mixed-Inverse freewall segment ***********************************
C
      IF(LMIXI) THEN
C
      IS = ISMIX
      IF(IS.EQ.1) JO = 1
      IF(IS.EQ.2) JO = JJ
      PWT = 1.0
      IF(IS.EQ.1) PWT = -1.0
      DO 110 IO=IX0, IX1
        IG = IO-ILE+1
        IGX0 = IX0-ILE+1
        IGX1 = IX1-ILE+1
        A1(JO,IO) = A1S(IO,IS)
        A2(JO,IO) = A2S(IO,IS)
        A3(JO,IO) = A3S(IO,IS)
        A4(JO,IO) = A4S(IO,IS)
        A5(JO,IO) = A5S(IO,IS)
        B1(JO,IO) = B1S(IO,IS)
        B2(JO,IO) = B2S(IO,IS)
        B3(JO,IO) = B3S(IO,IS)
        B4(JO,IO) = B4S(IO,IS)
        B5(JO,IO) = B5S(IO,IS)
        C1(JO,IO) = C1S(IO,IS)
        C2(JO,IO) = C2S(IO,IS)
        C3(JO,IO) = C3S(IO,IS)
        Z4(JO,IO) = Z4S(IO,IS)
        Z5(JO,IO) = Z5S(IO,IS)
C
        FX0 = (SG(IG,IS) - SG(IGX0,IS)) / (SG(IGX1,IS) - SG(IGX0,IS))
        FX1 = (SG(IGX1,IS) - SG(IG,IS)) / (SG(IGX1,IS) - SG(IGX0,IS))
```

```
C
        DR(JO,1,IO) =
     &       2.0*PWT*(PSPEC(IG,IS) + PDX0*FX0 + PDX1*FX1 - PI(IO,JO))
        DR(JO,LPDX0,IO) = -2.0*PWT*FX0
        DR(JO,LPDX1,IO) = -2.0*PWT*FX1
  110 CONTINUE
C
      ENDIF
C
C**** Cascade inlet/outlet fixing conditions ****************************
C
      IF(LCASC) THEN
C
       DO 150 JO = 1, JJ
         IO = 1
         A2(JO,IO) = 1.0
         DR(JO,1,IO) = 0.0
         DR(JO,LNINL,IO) = -1.0
  150  CONTINUE
C
       DO 155 JO = 1, JJ-1
         A2(JO,II) =  1.0
         A3(JO,II) = -1.0
         B2(JO,II) = -1.0
         B3(JO,II) =  1.0
         DR(JO,1,II) = 0.0
  155  CONTINUE
C
       A2(JJ,II) = 1.0
       DR(JJ,LNOUT,II) = -1.0
       RETURN
C
      ENDIF
C
C**** Airfoil far field conditions **************************************
C
      IF(LAIRF) THEN
C
      SOURCE = (Y(II,1) - Y(II,JJ)) / 6.2831853
      JO = JJJ
      JP = JJJ+1
C
      DO 200 IO = 2, II
C
        CALL CLROW(IO,JO)
        BWHY = SQRT(BETSQ)*Y(IO,JO)
        EXXS = X(IO,JO) - XCENT - 0.5
        ANGLE = ATAN2(BWHY,-EXXS)
        RSQ = (Y(IO,JO)-YCENT)**2 + (X(IO,JO)-XCENT)**2 / BETSQ
```

```
          ALOGR = 0.5*ALOG(RSQ)
          RES = Y(IO,JO) - YTOP(IO) - ALFA*(X(IO,JO)-XCENT) - CIRC*ALOGR
     &        - SOURCE*ANGLE
          A2(JO,IO) = 1.0 - CIRC*(Y(IO,JO)-YCENT)/RSQ
          DR(JO,1,IO) = -RES
          DR(JO,LCIRC,IO) = -ALOGR
          DR(JO,LALFA,IO) = -(X(IO,JO)-XCENT)
C
          CALL CLROW(IO,JP)
          BWHY = SQRT(BETSQ)*Y(IO,JP)
          EXXS = X(IO,JP) - XCENT - 0.5
          ANGLE = ATAN2(BWHY,-EXXS)
          RSQ = (Y(IO,JP)-YCENT)**2 + (X(IO,JP)-XCENT)**2 / BETSQ
          ALOGR = 0.5*ALOG(RSQ)
          RES = Y(IO,JP) - YBOT(IO) - ALFA*(X(IO,JP)-XCENT) - CIRC*ALOGR
     &        - SOURCE*ANGLE
          A2(JP,IO) = 1.0 - CIRC*(Y(IO,JP)-YCENT)/RSQ
          DR(JP,1,IO) = -RES
          DR(JP,LCIRC,IO) = -ALOGR
          DR(JP,LALFA,IO) = -(X(IO,JP)-XCENT)
C
  200 CONTINUE
C
        DO 250 JO = 1, JJ
          IO = 1
          BWHY = SQRT(BETSQ)*Y(IO,JO)
          EXXS = X(IO,JO) - XCENT - 0.5
          ANGLE = ATAN2(BWHY,-EXXS)
          RSQ = (Y(IO,JO)-YCENT)**2 + (X(IO,JO)-XCENT)**2 / BETSQ
          ALOGR = 0.5*ALOG(RSQ)
          RES = Y(IO,JO) - YINL(JO) - ALFA*(X(IO,JO)-XCENT) - CIRC*ALOGR
     &        - SOURCE*ANGLE
          A2(JO,IO) = 1.0 - CIRC*(Y(IO,JO)-YCENT)/RSQ
          DR(JO,1,IO) = -RES
          DR(JO,LCIRC,IO) = -ALOGR
          DR(JO,LALFA,IO) = -(X(IO,JO)-XCENT)
  250 CONTINUE
C
        DO 255 JO = 1, JJJ-1
          A2(JO,II) =  1.0
          A3(JO,II) = -1.0
          B2(JO,II) = -1.0
          B3(JO,II) =  1.0
          DR(JO,1,II) = 0.0
  255 CONTINUE
C
        DO 260 JO = JJJ+2, JJ
          A1(JO,II) = -1.0
          A2(JO,II) =  1.0
```

```
        B1(JO,II) =  1.0
        B2(JO,II) = -1.0
        DR(JO,1,II) = 0.0
 260  CONTINUE
C
C---- Set dummy s-mom equation
C
      DO 270 IO = 1, II
        A8(JJJ,IO) = 1.0
  270 CONTINUE
C
      ENDIF
      RETURN
      END ; SETBC


      SUBROUTINE CLROW(I,J)
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
      Z4(J,I) = 0.
      Z5(J,I) = 0.
      B1(J,I) = 0.
      B2(J,I) = 0.
      B3(J,I) = 0.
      B4(J,I) = 0.
      B5(J,I) = 0.
      A1(J,I) = 0.
      A2(J,I) = 0.
      A3(J,I) = 0.
      A4(J,I) = 0.
      A5(J,I) = 0.
      C1(J,I) = 0.
      C2(J,I) = 0.
      C3(J,I) = 0.
      DO 10 L=1, NGLX
        DR(J,L,I) = 0.
   10 CONTINUE
      RETURN
      END ; CLROW
```

SETBL is the subroutine that sets the coefficients for the linearized boundary layer equations, as detailed in M. Drela's Ph.D. thesis [11].

```
      SUBROUTINE SETBL(I,J)
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
      IO = I
      IM = IO-1
      JO = J
      JP = JO+1
      IG = IO-ILE+1
C
      HMU = 0.00
      TRF = 0.875
      FB2 = 0.5
      FB1 = 1.0 - FB2
C
      IF(JO.EQ.1) THEN
       IS = 1
       KT = 2*JJ+1
       KH = 2*JJ+2
      ELSE
       IS = 2
       KT = 2*JJ+4
       KH = 2*JJ+5
      ENDIF
C
      DO 10 L=1, 6
        AVT(IS,L,IO) = 0.
        BVT(IS,L,IO) = 0.
        ZVT(IS,L,IO) = 0.
        AVH(IS,L,IO) = 0.
        BVH(IS,L,IO) = 0.
        ZVH(IS,L,IO) = 0.
   10 CONTINUE
      CVT(IS,1,IO) = 0.
      CVT(IS,2,IO) = 0.
      CVH(IS,1,IO) = 0.
      CVH(IS,2,IO) = 0.
      DR(KT,1,IO) = 0.
      DR(KH,1,IO) = 0.
C
      TAU(IO,IS) = 0.0
C
      WT = GTR(IO,IS)
      WL = 1.0 - WT
```

```
C
      X1 = XI(IM,IS)
      X2 = XI(IO,IS)
C
      U1 = UEDG(IM,IS)
      U2 = UEDG(IO,IS)
      T1 = THET(IM,IS)
      T2 = THET(IO,IS)
      D1 = DISP(IM,IS)
      D2 = DISP(IO,IS)
C
      T0 = THET(IM-1,IS)
      D0 = DISP(IM-1,IS)
C
      M1 = U1*U1 / (GM1*(H(JO)-0.5*U1*U1))
      M2 = U2*U2 / (GM1*(H(JO)-0.5*U2*U2))
C
      TR1 = 1.0 + 0.5*GM1*M1
      TR2 = 1.0 + 0.5*GM1*M2
C
      R1 = RSTOUT*TR1**(-1.0/GM1)
      R2 = RSTOUT*TR2**(-1.0/GM1)
C
      IF(IO.EQ.ILE+1) THEN
C----- similarity station
       WL = 1.0
       WT = 0.
       HMU = 0.
       XLINV = 0.
       BETU = BULE
       BETT = 0.5*(1.0 - BULE)
       X1 = X2
       R1 = R2
       U1 = U2
       M1 = M2
       T1 = T2
       D1 = D2
       T0 = T2
       D0 = D2
      ELSE
C----- any other station
       XLINV = 1.0 / ALOG(X2/X1)
       BETU = ALOG(U2/U1)*XLINV
       BETT = ALOG(T2/T1)*XLINV
      ENDIF
C
      M1_U1 = M1*(2.0 + GM1*M1)/U1
      M2_U2 = M2*(2.0 + GM1*M2)/U2
C
```

```
      R1_U1 = -R1/TR1 * 0.5*M1_U1
      R2_U2 = -R2/TR2 * 0.5*M2_U2
C
      H0 = D0/T0
      H1 = D1/T1
      H2 = D2/T2
      H0_D0 = 1.0/T0
      H1_D1 = 1.0/T1
      H2_D2 = 1.0/T2
      H0_T0 = -H0/T0
      H1_T1 = -H1/T1
      H2_T2 = -H2/T2
C
      HS1 = H(JO) - 0.5*U1*U1
      HS2 = H(JO) - 0.5*U2*U2
      V1 = SQRT((HS1/H(JO))**3) * (H(JO)+HVIS)/(HS1+HVIS) / REYN
      V2 = SQRT((HS2/H(JO))**3) * (H(JO)+HVIS)/(HS2+HVIS) / REYN
      V1_U1 = V1*U1*(1.0/(HS1+HVIS)-1.5/HS1)
      V2_U2 = V2*U2*(1.0/(HS2+HVIS)-1.5/HS2)
C
C     HW1 = HS1 + TRF*(H(JO)-HS1)
C     HW2 = HS2 + TRF*(H(JO)-HS2)
C     VW1 = SQRT((HW1/H(JO))**3) * (H(JO)+HVIS)/(HW1+HVIS) / REYN
C     VW2 = SQRT((HW2/H(JO))**3) * (H(JO)+HVIS)/(HW2+HVIS) / REYN
C     VW1_U1 = VW1*U1*(1.0/(HW1+HVIS)-1.5/HW1) * (1.0-TRF)
C     VW2_U2 = VW2*U2*(1.0/(HW2+HVIS)-1.5/HW2) * (1.0-TRF)
C
      XA = 0.5*(X1+X2)
      RA = 0.5*(R1+R2)
      UA = 0.5*(U1+U2)
      MA = 0.5*(M1+M2)
      TA = 0.5*(T1+T2)
      DA = 0.5*(D1+D2)
      HA = 0.5*(H1+H2)
      VA = 0.5*(V1+V2)
C
      HL1 = (1.0-HMU)*H1 + HMU*H0
      HL2 = (1.0-HMU)*H2 + HMU*H1
      HL1_T0 =       HMU *H0_T0
      HL1_T1 = (1.0-HMU)*H1_T1
      HL2_T1 =       HMU *H1_T1
      HL2_T2 = (1.0-HMU)*H2_T2
      HL1_D0 =       HMU *H0_D0
      HL1_D1 = (1.0-HMU)*H1_D1
      HL2_D1 =       HMU *H1_D1
      HL2_D2 = (1.0-HMU)*H2_D2
C
      HKA = (HA  - 0.29*MA)/(1.0 + 0.113*MA)
      HK1 = (HL1 - 0.29*M1)/(1.0 + 0.113*M1)
```

```
      HK2 = (HL2 - 0.29*M2)/(1.0 + 0.113*M2)
      HKA = AMAX1(HKA,1.02)
      HK1 = AMAX1(HK1,1.02)
      HK2 = AMAX1(HK2,1.02)
      HKA_HA  = 1.0/(1.0 + 0.113*MA)
      HK1_HL1 = 1.0/(1.0 + 0.113*M1)
      HK2_HL2 = 1.0/(1.0 + 0.113*M2)
      HKA_MA = (-.29 - 0.113*HKA) / (1.0 + 0.113*MA)
      HK1_M1 = (-.29 - 0.113*HK1) / (1.0 + 0.113*M1)
      HK2_M2 = (-.29 - 0.113*HK2) / (1.0 + 0.113*M2)
C
      HKA_U1 = HKA_MA*0.5*M1_U1
      HKA_U2 = HKA_MA*0.5*M2_U2
      HKA_T1 = HKA_HA*0.5*H1_T1
      HKA_T2 = HKA_HA*0.5*H2_T2
      HKA_D1 = HKA_HA*0.5*H1_D1
      HKA_D2 = HKA_HA*0.5*H2_D2
C
      HK1_U1 = HK1_M1*M1_U1
      HK2_U2 = HK2_M2*M2_U2
C
      HK1_T0 = HK1_HL1*HL1_T0
      HK1_T1 = HK1_HL1*HL1_T1
      HK2_T1 = HK2_HL2*HL2_T1
      HK2_T2 = HK2_HL2*HL2_T2
C
      HK1_D0 = HK1_HL1*HL1_D0
      HK1_D1 = HK1_HL1*HL1_D1
      HK2_D1 = HK2_HL2*HL2_D1
      HK2_D2 = HK2_HL2*HL2_D2
C
C**** Laminar HE correlation  (from LeBalleur)
      EX1 = EXP(-2.5*(HK1-2.0))
      EX2 = EXP(-2.5*(HK2-2.0))
      HE1 = WL*(0.635 + 0.055*HK1*(HK1+8.0)/(HK1-1.0) - 0.048*EX1)
      IF(HK1.GT.4.0) THEN
       HE1 = 1.514676
       HE1_HK1 = 0.0
      ENDIF
      HE2 = WL*(0.635 + 0.055*HK2*(HK2+8.0)/(HK2-1.0) - 0.048*EX2)
      HE1_HK1 = WL*(0.055*(HK1-4.0)*(HK1+2.0)/(HK1-1.0)**2 + 0.12*EX1)
      HE2_HK2 = WL*(0.055*(HK2-4.0)*(HK2+2.0)/(HK2-1.0)**2 + 0.12*EX2)
      IF(HK2.GT.4.0) THEN
       HE2 = 1.514676
       HE2_HK2 = 0.0
      ENDIF
C
C**** Turbulent HE correlation  (Whitfield)
      IF(HK1.LE.3.4) THEN
```

```
       EX1 = EXP(6.0*(HK1-3.4))
       EX2 = EXP(6.0*(HK2-3.4))
       TMP1 = (HK1-2.75)/1.75
       TMP2 = (HK2-2.75)/1.75
       HE1 = HE1 + WT*(1.5 + TMP1*TMP1/6. - TMP1*TMP1*TMP1/3. + .02*EX1)
       HE2 = HE2 + WT*(1.5 + TMP2*TMP2/6. - TMP2*TMP2*TMP2/3. + .02*EX2)
       HE1_HK1 = HE1_HK1 + WT*((TMP1/3.0 - TMP1*TMP1)/1.75 + 0.12*EX1)
       HE2_HK2 = HE2_HK2 + WT*((TMP2/3.0 - TMP2*TMP2)/1.75 + 0.12*EX2)
      ELSE
       HE1 = HE1 + WT*(1.628 + .0133*(HK1-5.) + .0556*((HK1-5.)/1.6)**3)
       HE2 = HE2 + WT*(1.628 + .0133*(HK2-5.) + .0556*((HK2-5.)/1.6)**3)
       HE1_HK1 = HE1_HK1 + WT*(.0133 + .0556*3.0*((HK1-5.)/1.6)**2/1.6)
       HE2_HK2 = HE2_HK2 + WT*(.0133 + .0556*3.0*((HK2-5.)/1.6)**2/1.6)
      ENDIF
C
      HE1_U1 = HE1_HK1*HK1_U1
      HE2_U2 = HE2_HK2*HK2_U2
      HE1_T0 = HE1_HK1*HK1_T0
      HE1_T1 = HE1_HK1*HK1_T1
      HE2_T1 = HE2_HK2*HK2_T1
      HE2_T2 = HE2_HK2*HK2_T2
      HE1_D0 = HE1_HK1*HK1_D0
      HE1_D1 = HE1_HK1*HK1_D1
      HE2_D1 = HE2_HK2*HK2_D1
      HE2_D2 = HE2_HK2*HK2_D2
C
      RT = RA*UA*TA/VA
C
C**** Laminar Cf correlation      (from Falkner-Skan)
      IF(HKA.LT.7.4) THEN
       TMP = (7.4-HKA)**2.2 / (HKA-1.0)
       CFN    = 0.02844*TMP - 0.134
       CFN_HKA = -.02844*TMP * (2.2/(7.4-HKA) + 1.0/(HKA-1.0))
      ELSE
       TMP = 1.0 - 1.4/(HKA-6.0)
       CFN     = 0.044*TMP**2 - 0.134
       CFN_HKA = 0.088*TMP*1.4/(HKA-6.0)**2
      ENDIF
C
CCC   CF     = WL*CFN/RT
CCC   CF_HKA = WL*CFN_HKA/RT
CCC   CF_RT  = -CF/RT
C
      CF     = WL*CFN
      CF_HKA = WL*CFN_HKA
      CF_RT  = 0.0
C
C**** Turbulent Cf correlation        (Whitfield)
      GRT = ALOG(RT)
```

```
      GRT = AMAX1(GRT,3.0)
      GEX = -1.74 - 0.31*HKA
      CFB = 0.3*EXP(-1.33*HKA) * (GRT/2.3026)**GEX
      CFB_HKA = -1.33*CFB - 0.31*ALOG(GRT/2.3026)*CFB
      CFB_RT = GEX*CFB/(GRT*RT)
C
      CF     = CF     + WT*CFB
      CF_HKA = CF_HKA + WT*CFB_HKA
      CF_RT  = CF_RT  + WT*CFB_RT
C
C     IF(IO.GT.ITE) THEN              ;  set zero shear in wake
C      CF     = 0.
C      CF_HKA = 0.
C      CF_RT  = 0.
C     ENDIF
C
      CF_U1 = CF_HKA*HKA_U1 + CF_RT*.5*RT*(R1_U1/RA + 1.0/UA - V1_U1/VA)
      CF_U2 = CF_HKA*HKA_U2 + CF_RT*.5*RT*(R2_U2/RA + 1.0/UA - V2_U2/VA)
      CF_T1 = CF_HKA*HKA_T1 + CF_RT*.5*RT/TA
      CF_T2 = CF_HKA*HKA_T2 + CF_RT*.5*RT/TA
      CF_D1 = CF_HKA*HKA_D1
      CF_D2 = CF_HKA*HKA_D2
C
CCC   TAU(IO,IS) = 0.5*RA*UA*UA * CF
      TAU(IO,IS) = 0.5*RA*UA*UA * CF/RT
C
C**** Laminar CD correlation          (from Falkner-Skan)
      IF(HKA.LT.4.0) THEN
       TMP = (4.0-HKA)**4.44
       CDN     = 0.004*TMP + 0.156
       CDN_HKA = -.004*TMP*4.44/(4.0-HKA)
      ELSE
       TMP = (HKA-4.0)**1.8
       CDN     = -.0015*TMP + 0.156
       CDN_HKA = -.0015*TMP*1.8/(HKA-4.0)
      ENDIF
C
CCC   CD     = WL*CDN/RT
CCC   CD_HKA = WL*CDN_HKA/RT
CCC   CD_RT  = -CD/RT
C
      CD     = WL*CDN
      CD_HKA = WL*CDN_HKA
      CD_RT  = 0.0
C
C**** Turbulent CD correlation    (Whitfield)
C???????????
C
      CD_U1 = CD_HKA*HKA_U1 + CD_RT*.5*RT*(R1_U1/RA + 1.0/UA - V1_U1/VA)
```

```
      CD_U2 = CD_HKA*HKA_U2 + CD_RT*.5*RT*(R2_U2/RA + 1.0/UA - V2_U2/VA)
      CD_T1 = CD_HKA*HKA_T1 + CD_RT*.5*RT/TA
      CD_T2 = CD_HKA*HKA_T2 + CD_RT*.5*RT/TA
      CD_D1 = CD_HKA*HKA_D1
      CD_D2 = CD_HKA*HKA_D2
C
C---- set up momentum equation
      BTMP = HA + 2.0 - MA
CCC   CTMP = 0.5*XA/TA
      S1 = X1*V1/(R1*U1*T1*T1)
      S2 = X2*V2/(R2*U2*T2*T2)
      CTMP = 0.25*(S1 + S2)
      REZ1 = BETT + BTMP*BETU - CTMP*CF
      Z_S = -0.25*CF
C
      Z_U1 = -CTMP*CF_U1 - BETU*0.5*M1_U1 - XLINV/U1*BTMP
      Z_U2 = -CTMP*CF_U2 - BETU*0.5*M2_U2 + XLINV/U2*BTMP
      Z_T1 = -CTMP*CF_T1 + BETU*0.5*H1_T1 - XLINV/T1  ; + 0.5*CTMP*CF/TA
      Z_T2 = -CTMP*CF_T2 + BETU*0.5*H2_T2 + XLINV/T2  ; + 0.5*CTMP*CF/TA
      Z_D1 = -CTMP*CF_D1 + BETU*0.5*H1_D1
      Z_D2 = -CTMP*CF_D2 + BETU*0.5*H2_D2
C
      Z_U1 = Z_U1 + Z_S*(-S1/U1 - S1/R1*R1_U1 + S1/V1*V1_U1)
      Z_U2 = Z_U2 + Z_S*(-S2/U2 - S2/R2*R2_U2 + S2/V2*V2_U2)
      Z_T1 = Z_T1 + Z_S*(-S1/T1)
      Z_T2 = Z_T2 + Z_S*(-S2/T2)
C
      BVT(IS,4,IO) = Z_U1
      AVT(IS,4,IO) = Z_U2
      BVT(IS,5,IO) = Z_T1
      AVT(IS,5,IO) = Z_T2
      BVT(IS,6,IO) = Z_D1
      AVT(IS,6,IO) = Z_D2
C
      DR(KT,1,IO) = -REZ1
C
C
C---- Set up dissipation equation
C
      BETH = 0.0
      IF(IO.GT.ILE+1) BETH = ALOG(HE2/HE1)*XLINV
C
      HEA = 0.5*(HE1+HE2)
C
      BTMP = 3.0 - MA
      CTMP = (S1+S2)/HEA
      REZ2 = BETH + BETT + BTMP*BETU - CTMP*CD
      Z_HE1 = 0.5*CTMP*CD/HEA - XLINV/HE1
      Z_HE2 = 0.5*CTMP*CD/HEA + XLINV/HE2
```

```
      Z_CD = -CTMP
      Z_MA = -BETU
      Z_TA = 0.0
      Z_S = -CD/HEA
C
      Z_U1 = Z_HE1*HE1_U1 + Z_CD*CD_U1 + Z_MA*0.5*M1_U1 - XLINV/U1*BTMP
      Z_U2 = Z_HE2*HE2_U2 + Z_CD*CD_U2 + Z_MA*0.5*M2_U2 + XLINV/U2*BTMP
      Z_T0 = Z_HE1*HE1_T0
      Z_T1 = Z_HE1*HE1_T1 + Z_CD*CD_T1 + Z_TA*0.5       - XLINV/T1
     &     + Z_HE2*HE2_T1
      Z_T2 = Z_HE2*HE2_T2 + Z_CD*CD_T2 + Z_TA*0.5       + XLINV/T2
      Z_D0 = Z_HE1*HE1_D0
      Z_D1 = Z_HE1*HE1_D1 + Z_CD*CD_D1
     &     + Z_HE2*HE2_D1
      Z_D2 = Z_HE2*HE2_D2 + Z_CD*CD_D2
C
      Z_U1 = Z_U1 + Z_S*(-S1/U1 - S1/R1*R1_U1 + S1/V1*V1_U1)
      Z_U2 = Z_U2 + Z_S*(-S2/U2 - S2/R2*R2_U2 + S2/V2*V2_U2)
      Z_T1 = Z_T1 + Z_S*(-S1/T1)
      Z_T2 = Z_T2 + Z_S*(-S2/T2)
C
      ZVH(IS,4,IO) = 0.
      BVH(IS,4,IO) = Z_U1
      AVH(IS,4,IO) = Z_U2
      ZVH(IS,5,IO) = Z_T0
      BVH(IS,5,IO) = Z_T1
      AVH(IS,5,IO) = Z_T2
      ZVH(IS,6,IO) = Z_D0
      BVH(IS,6,IO) = Z_D1
      AVH(IS,6,IO) = Z_D2
C
      DR(KH,1,IO) = -REZ2
C
      IF(IO.LE.ILE+6) THEN
      WRITE(5,*) ' '
      WRITE(5,*) IO, REZ1, REZ2, HK1, HK2
      WRITE(5,*) HE1, HE2, CF, CD, X1, X2
      WRITE(5,*) BETU, BETT, BETH, S1, S2
C
      ENDIF
C
      IF(IO.EQ.ILE+1) THEN
C----- redefine Jacobian entries for similarity station
       AVT(IS,4,IO) = AVT(IS,4,IO) + BVT(IS,4,IO) + ZVT(IS,4,IO)
       BVT(IS,4,IO) = 0.
       ZVT(IS,4,IO) = 0.
       AVT(IS,5,IO) = AVT(IS,5,IO) + BVT(IS,5,IO) + ZVT(IS,5,IO)
       BVT(IS,5,IO) = 0.
       ZVT(IS,5,IO) = 0.
```

```
 AVT(IS,6,IO) = AVT(IS,6,IO) + BVT(IS,6,IO) + ZVT(IS,6,IO)
 BVT(IS,6,IO) = 0.
 ZVT(IS,6,IO) = 0.
 AVH(IS,4,IO) = AVH(IS,4,IO) + BVH(IS,4,IO) + ZVH(IS,4,IO)
 BVH(IS,4,IO) = 0.
 ZVH(IS,4,IO) = 0.
 AVH(IS,5,IO) = AVH(IS,5,IO) + BVH(IS,5,IO) + ZVH(IS,5,IO)
 BVH(IS,5,IO) = 0.
 ZVH(IS,5,IO) = 0.
 AVH(IS,6,IO) = AVH(IS,6,IO) + BVH(IS,6,IO) + ZVH(IS,6,IO)
 BVH(IS,6,IO) = 0.
 ZVH(IS,6,IO) = 0.
ENDIF
RETURN
END ; SETBL
```

FSOLVE is the subroutine that solves the linear system of Newton equations by the direct block elimination method.

```
      SUBROUTINE SOLVE
$INCLUDE STATE.INC
$INCLUDE ISES.INC
       COMMON/ARRAY/A(JW,JW),B(JW,JW),C(JW,JX,IX),Z(JW,JW)
C
      NBK = 2*JJ+5
      K1 = JJ+1
      K2 = NBK-6
C
CCC** Forward sweep: Elimination of lower block diagonals (B's and Z's).
      DO 1000 I=1, II
        IM = I-1
        IL = I-2
        DO 5 K=1, NBK
          DO 501 L=1, NBK
            A(K,L) = 0.
            B(K,L) = 0.
            Z(K,L) = 0.
  501     CONTINUE
          DO 502 L=1, JJ
            C(K,L,I) = 0.
  502     CONTINUE
    5   CONTINUE
C
C
CCC**** FIRST, fill 'er up
        Z(1 ,K2) = Z4(1,I)
        Z(1 ,K1) = Z5(1,I)
        Z(JJ,K2) = Z4(JJ,I)
        Z(JJ,K1) = Z5(JJ,I)
        B(1 ,K2) = B4(1,I)
        B(1 ,K1) = B5(1,I)
        B(JJ,K2) = B4(JJ,I)
        B(JJ,K1) = B5(JJ,I)
        A(1 ,K2) = A4(1,I)
        A(1 ,K1) = A5(1,I)
        A(JJ,K2) = A4(JJ,I)
        A(JJ,K1) = A5(JJ,I)
        DO 111 J=2, JJ-1
          Z(J,K1+J-2) = Z4(J,I)
          Z(J,K1+J-1) = Z5(J,I)
          B(J,K1+J-2) = B4(J,I)
          B(J,K1+J-1) = B5(J,I)
          A(J,K1+J-2) = A4(J,I)
          A(J,K1+J-1) = A5(J,I)
```

```
  111   CONTINUE
        DO 112 J=1, JJ-1
          K = K1+J-1
          Z(K,K) = Z8(J,I)
          B(K,K) = B8(J,I)
          A(K,K) = A8(J,I)
  112   CONTINUE
        Z(K2+1,K1) = ZVC(1,3,I)
        Z(K2+2,K1) = ZVT(1,3,I)
        Z(K2+3,K1) = ZVH(1,3,I)
        Z(K2+4,K2) = ZVC(2,3,I)
        Z(K2+5,K2) = ZVT(2,3,I)
        Z(K2+6,K2) = ZVH(2,3,I)
        B(K2+1,K1) = BVC(1,3,I)
        B(K2+2,K1) = BVT(1,3,I)
        B(K2+3,K1) = BVH(1,3,I)
        B(K2+4,K2) = BVC(2,3,I)
        B(K2+5,K2) = BVT(2,3,I)
        B(K2+6,K2) = BVH(2,3,I)
        A(K2+1,K1) = AVC(1,3,I)
        A(K2+2,K1) = AVT(1,3,I)
        A(K2+3,K1) = AVH(1,3,I)
        A(K2+4,K2) = AVC(2,3,I)
        A(K2+5,K2) = AVT(2,3,I)
        A(K2+6,K2) = AVH(2,3,I)
C
C------ Fill left parts of ? block
        B(1 ,1)    = B2(1 ,I)                  ; <-P
        B(1 ,2)    = B3(1 ,I)                  ; <-P
        B(1 ,JJ-1) = B1(1 ,I)                  ; <-P
        B(1 ,JJ)   = BT(1 ,I)                  ; <-P
        B(JJ,2)    = B3(JJ,I)                  ; <-P
        B(JJ,JJ-1) = B1(JJ,I)                  ; <-P
        B(JJ,JJ)   = B2(JJ,I)                  ; <-P
        B(JJ,1)    = BT(2 ,I)                  ; <-P
        A(1 ,1)    = A2(1 ,I)                  ; <-P
        A(1 ,2)    = A3(1 ,I)                  ; <-P
        A(1 ,JJ-1) = A1(1 ,I)                  ; <-P
        A(1 ,JJ)   = AT(1 ,I)                  ; <-P
        A(JJ,2)    = A3(JJ,I)                  ; <-P
        A(JJ,JJ-1) = A1(JJ,I)                  ; <-P
        A(JJ,JJ)   = A2(JJ,I)                  ; <-P
        A(JJ,1)    = AT(2 ,I)                  ; <-P
        C(1 ,1   ,I) = C2(1 ,I)                  ; <-P
        C(1 ,2   ,I) = C3(1 ,I)                  ; <-P
        C(1 ,JJ-1,I) = C1(1 ,I)                  ; <-P
        C(1 ,JJ  ,I) = CT(1 ,I)                  ; <-P
        C(JJ,2   ,I) = C3(JJ,I)                  ; <-P
        C(JJ,JJ-1,I) = C1(JJ,I)                  ; <-P
```

```
      C(JJ,JJ  ,I) = C2(JJ,I)                    ; <-P
      C(JJ,1   ,I) = CT(2 ,I)                    ; <-P
      DO 131 J=2, JJ-1
        B(J,J-1) = B1(J,I)
        B(J,J  ) = B2(J,I)
        B(J,J+1) = B3(J,I)
        A(J,J-1) = A1(J,I)
        A(J,J  ) = A2(J,I)
        A(J,J+1) = A3(J,I)
        C(J,J-1,I) = C1(J,I)
        C(J,J  ,I) = C2(J,I)
        C(J,J+1,I) = C3(J,I)
131   CONTINUE
      DO 132 J=1, JJ-1
        K = J+JJ
        B(K,J  ) = B6(J,I)
        B(K,J+1) = B7(J,I)
        A(K,J  ) = A6(J,I)
        A(K,J+1) = A7(J,I)
        C(K,J  ,I) = C6(J,I)
        C(K,J+1,I) = C7(J,I)
132   CONTINUE
      B(K2+1,1)    = BVC(1,1,I)
      B(K2+2,1)    = BVT(1,1,I)
      B(K2+3,1)    = BVH(1,1,I)
      B(K2+1,2)    = BVC(1,2,I)
      B(K2+2,2)    = BVT(1,2,I)
      B(K2+3,2)    = BVH(1,2,I)
      B(K2+4,JJ-1) = BVC(2,1,I)
      B(K2+5,JJ-1) = BVT(2,1,I)
      B(K2+6,JJ-1) = BVH(2,1,I)
      B(K2+4,JJ)   = BVC(2,2,I)
      B(K2+5,JJ)   = BVT(2,2,I)
      B(K2+6,JJ)   = BVH(2,2,I)
      A(K2+1,1)    = AVC(1,1,I)
      A(K2+2,1)    = AVT(1,1,I)
      A(K2+3,1)    = AVH(1,1,I)
      A(K2+1,2)    = AVC(1,2,I)
      A(K2+2,2)    = AVT(1,2,I)
      A(K2+3,2)    = AVH(1,2,I)
      A(K2+4,JJ-1) = AVC(2,1,I)
      A(K2+5,JJ-1) = AVT(2,1,I)
      A(K2+6,JJ-1) = AVH(2,1,I)
      A(K2+4,JJ)   = AVC(2,2,I)
      A(K2+5,JJ)   = AVT(2,2,I)
      A(K2+6,JJ)   = AVH(2,2,I)
      C(K2+1,1,I)    = CVC(1,1,I)
      C(K2+2,1,I)    = CVT(1,1,I)
      C(K2+3,1,I)    = CVH(1,1,I)
```

```
          C(K2+1,2,I)     = CVC(1,2,I)
          C(K2+2,2,I)     = CVT(1,2,I)
          C(K2+3,2,I)     = CVH(1,2,I)
          C(K2+4,JJ-1,I) = CVC(2,1,I)
          C(K2+5,JJ-1,I) = CVT(2,1,I)
          C(K2+6,JJ-1,I) = CVH(2,1,I)
          C(K2+4,JJ,I)    = CVC(2,2,I)
          C(K2+5,JJ,I)    = CVT(2,2,I)
          C(K2+6,JJ,I)    = CVH(2,2,I)
C
          DO 14 N=1, 6
            L = K2+N
            B(1 ,L) = BI(1,N,I)
            B(JJ,L) = BI(2,N,I)
            A(1 ,L) = AI(1,N,I)
            A(JJ,L) = AI(2,N,I)
   14     CONTINUE
C
          DO 15 L=1, 3
            Z(K2+1,K2+L) = ZVC(1,L+3,I)
            Z(K2+2,K2+L) = ZVT(1,L+3,I)
            Z(K2+3,K2+L) = ZVH(1,L+3,I)
            B(K2+1,K2+L) = BVC(1,L+3,I)
            B(K2+2,K2+L) = BVT(1,L+3,I)
            B(K2+3,K2+L) = BVH(1,L+3,I)
            A(K2+1,K2+L) = AVC(1,L+3,I)
            A(K2+2,K2+L) = AVT(1,L+3,I)
            A(K2+3,K2+L) = AVH(1,L+3,I)
            Z(K2+4,K2+L+3) = ZVC(2,L+3,I)
            Z(K2+5,K2+L+3) = ZVT(2,L+3,I)
            Z(K2+6,K2+L+3) = ZVH(2,L+3,I)
            B(K2+4,K2+L+3) = BVC(2,L+3,I)
            B(K2+5,K2+L+3) = BVT(2,L+3,I)
            B(K2+6,K2+L+3) = BVH(2,L+3,I)
            A(K2+4,K2+L+3) = AVC(2,L+3,I)
            A(K2+5,K2+L+3) = AVT(2,L+3,I)
            A(K2+6,K2+L+3) = AVH(2,L+3,I)
   15     CONTINUE
C
CCC**** Eliminate right Z block
          IF(I.GE.3) THEN
          DO 16 K=1, NBK
            DO 161 J=K1, NBK
              IF(Z(K,J).EQ.0.0) GO TO 161
              DO 1611 L=1, JJ
                B(K,L) = B(K,L) - Z(K,J)*C(J,L,IL)
 1611         CONTINUE
              DO 1612 L=1, NRHS
                DR(K,L,I) = DR(K,L,I) - Z(K,J)*DR(J,L,IL)
```

```
 1612       CONTINUE
  161     CONTINUE
   16   CONTINUE
        ENDIF
C
CCC**** Eliminate B block
        IF(I.GE.2) THEN
        DO 17 K=1, NBK
          DO 171 J=1, NBK
            IF(B(K,J).EQ.0.0) GO TO 171
            DO 1711 L=1, JJ
              A(K,L) = A(K,L) - B(K,J)*C(J,L,IM)
 1711       CONTINUE
            DO 1712 L=1, NRHS
              DR(K,L,I) = DR(K,L,I) - B(K,J)*DR(J,L,IM)
 1712       CONTINUE
  171     CONTINUE
   17   CONTINUE
        ENDIF
C
CCC**** Invert A block
        DO 20 NP=NBK, 2, -1
          NM1 = NP-1
          PIVOT = 1.0/A(NP,NP)
C
C-------- normalize pivot row
          DO 201 L=1, NM1
            A(NP,L) = A(NP,L)*PIVOT
  201     CONTINUE
          DO 202 L=1, JJ
            C(NP,L,I) = C(NP,L,I)*PIVOT
  202     CONTINUE
          DO 203 L=1, NRHS
            DR(NP,L,I) = DR(NP,L,I)*PIVOT
  203     CONTINUE
C
C-------- eliminate pivot column
          DO 210 K=1, NM1
            IF(A(K,NP).EQ.0.0) GO TO 210
            DO 2101 L=1, NM1
              A(K,L) = A(K,L) - A(K,NP)*A(NP,L)
 2101       CONTINUE
            DO 2102 L=1, JJ
              C(K,L,I) = C(K,L,I) - A(K,NP)*C(NP,L,I)
 2102       CONTINUE
            DO 2103 L=1, NRHS
              DR(K,L,I) = DR(K,L,I) - A(K,NP)*DR(NP,L,I)
 2103       CONTINUE
  210     CONTINUE
```

```
   20   CONTINUE
C
C------ normalize last row
        PIVOT = 1.0 / A(1,1)
        DO 21 L=1, JJ
          C(1,L,I) = C(1,L,I)*PIVOT
   21   CONTINUE
        DO 22 L=1, NRHS
          DR(1,L,I) = DR(1,L,I)*PIVOT
   22   CONTINUE
C
C------ back substitute
        DO 24 NP=2, NBK
          NM1 = NP-1
          DO 241 K=1, NM1
            IF(A(NP,K).EQ.0.0) GO TO 241
            DO 2411 L=1, JJ
              C(NP,L,I) = C(NP,L,I) - A(NP,K)*C(K,L,I)
 2411       CONTINUE
            DO 2412 L=1, NRHS
              DR(NP,L,I) = DR(NP,L,I) - A(NP,K)*DR(K,L,I)
 2412       CONTINUE
  241     CONTINUE
   24   CONTINUE
C
 1000 CONTINUE
C
CCC** Backward sweep: Elimination of upper block diagonal (C's).
      DO 2000 I=II-1, 1, -1
        IP = I+1
        DO 51 N=1, NBK
          DO 511 L=1, NRHS
            DO 5111 K=1, JJ
              DR(N,L,I) = DR(N,L,I) - C(N,K,I)*DR(K,L,IP)
 5111       CONTINUE
  511     CONTINUE
   51   CONTINUE
 2000 CONTINUE
C
      RETURN
      END ; FSOLVE
```

CSOLVE is the subroutine that solves the linear system of Newton equations for choked flows.

```
      SUBROUTINE SOLVE
$INCLUDE STATE.INC
$INCLUDE ISES.INC
$INCLUDE ARRAY.INC
C
      NBK = 2*JJ+5
      K1 = JJ+1
      K2 = NBK-6
      JS = JJ/2
      KS = JJ+JS
C
CCC** Forward sweep: Elimination of lower block diagonals (B's and Z's).
      DO 1000 I=1, II
        IM = I-1
        IL = I-2
        DO 5 K=1, NBK
          DO 501 L=1, NBK
            A(K,L) = 0.
            B(K,L) = 0.
            Z(K,L) = 0.
  501     CONTINUE
          DO 502 L=1, JJ
            C(K,L,I) = 0.
  502     CONTINUE
          DO 503 L = NRHS+1, NRHS+3
            DR(K,L,I) = 0.
  503     CONTINUE
    5   CONTINUE
C
CCC**** FIRST, fill 'er up
C
CCC**** I=1 Special treatment (assumes Z,B are zero)
        IF(I.EQ.1) THEN
        DO 6 J=1, JJ
          A(J,J) = A2(J,1)
    6   CONTINUE
        DO 7 J=1, JS-1
          K = K1+J-1
          A(K,J  ) = A6(J,1)
          A(K,J+1) = A7(J,1) - A6(J+1,1)
          A(K,J+2) =         - A7(J+1,1)
          A(K,K  ) = A8(J,1)
          A(K,K+1) =         - A8(J+1,1)
          C(K,J  ,1) = C6(J,1)
          C(K,J+1,1) = C7(J,1) - C6(J+1,1)
```

```
            C(K,J+2,1) =          - C7(J+1,1)
            DO 71 L = 1, NRHS
               DR(K,L,1) = DR(K,L,1) - DR(K+1,L,1)
   71       CONTINUE
    7     CONTINUE
          DO 8 J=JJ-1, JS+1, -1
            K = K1+J-1
            A(K,J-1) = A6(J-1,1)
            A(K,J  ) = A7(J-1,1) - A6(J,1)
            A(K,J+1) =           - A7(J,1)
            A(K,K-1) = A8(J-1,1)
            A(K,K  ) =           - A8(J,1)
            C(K,J-1,1) = C6(J-1,1)
            C(K,J  ,1) = C7(J-1,1) - C6(J,1)
            C(K,J+1,1) =           - C7(J,1)
            DO 81 L = 1, NRHS
               DR(K,L,1) = DR(K-1,L,1) - DR(K,L,1)
   81       CONTINUE
    8     CONTINUE
          A(KS,JS  ) = B6(JS,2)
          A(KS,JS+1) = B7(JS,2)
          A(KS,KS  ) = B8(JS,2)
          C(KS,JS  ,1) = A6(JS,2)
          C(KS,JS+1,1) = A7(JS,2)
          DR(KS,NRHS+1,1) = A8(JS,2)
          DR(KS,NRHS+2,1) = C6(JS,2)
          DR(KS,NRHS+3,1) = C7(JS,2)
          DO 9 L=1, NRHS
            DR(KS,L,1) = DR(KS,L,2)
    9     CONTINUE
          DO 10 K = K2+1, K2+6
            A(K,K) = 1.0
   10     CONTINUE
C
C****** I not equal to 1
          ELSE
          Z(1 ,K2) = Z4(1,I)
          Z(1 ,K1) = Z5(1,I)
          Z(JJ,K2) = Z4(JJ,I)
          Z(JJ,K1) = Z5(JJ,I)
          B(1 ,K2) = B4(1,I)
          B(1 ,K1) = B5(1,I)
          B(JJ,K2) = B4(JJ,I)
          B(JJ,K1) = B5(JJ,I)
          A(1 ,K2) = A4(1,I)
          A(1 ,K1) = A5(1,I)
          A(JJ,K2) = A4(JJ,I)
          A(JJ,K1) = A5(JJ,I)
          DO 111 J=2, JJ-1
```

```
          Z(J,K1+J-2) = Z4(J,I)
          Z(J,K1+J-1) = Z5(J,I)
          B(J,K1+J-2) = B4(J,I)
          B(J,K1+J-1) = B5(J,I)
          A(J,K1+J-2) = A4(J,I)
          A(J,K1+J-1) = A5(J,I)
  111   CONTINUE
        DO 112 J=1, JJ-1
          K = K1+J-1
          Z(K,K) = Z8(J,I)
          B(K,K) = B8(J,I)
          A(K,K) = A8(J,I)
  112   CONTINUE
        Z(K2+1,K1) = ZVC(1,3,I)
        Z(K2+2,K1) = ZVT(1,3,I)
        Z(K2+3,K1) = ZVH(1,3,I)
        Z(K2+4,K2) = ZVC(2,3,I)
        Z(K2+5,K2) = ZVT(2,3,I)
        Z(K2+6,K2) = ZVH(2,3,I)
        B(K2+1,K1) = BVC(1,3,I)
        B(K2+2,K1) = BVT(1,3,I)
        B(K2+3,K1) = BVH(1,3,I)
        B(K2+4,K2) = BVC(2,3,I)
        B(K2+5,K2) = BVT(2,3,I)
        B(K2+6,K2) = BVH(2,3,I)
        A(K2+1,K1) = AVC(1,3,I)
        A(K2+2,K1) = AVT(1,3,I)
        A(K2+3,K1) = AVH(1,3,I)
        A(K2+4,K2) = AVC(2,3,I)
        A(K2+5,K2) = AVT(2,3,I)
        A(K2+6,K2) = AVH(2,3,I)
C
C------ Fill left parts of ? block
        B(1 ,1)    = B2(1 ,I)                    ; <-P
        B(1 ,2)    = B3(1 ,I)                    ; <-P
        B(1 ,JJ-1) = B1(1 ,I)                    ; <-P
        B(1 ,JJ)   = BT(1 ,I)                    ; <-P
        B(JJ,2)    = B3(JJ,I)                    ; <-P
        B(JJ,JJ-1) = B1(JJ,I)                    ; <-P
        B(JJ,JJ)   = B2(JJ,I)                    ; <-P
        B(JJ,1)    = BT(2 ,I)                    ; <-P
        A(1 ,1)    = A2(1 ,I)                    ; <-P
        A(1 ,2)    = A3(1 ,I)                    ; <-P
        A(1 ,JJ-1) = A1(1 ,I)                    ; <-P
        A(1 ,JJ)   = AT(1 ,I)                    ; <-P
        A(JJ,2)    = A3(JJ,I)                    ; <-P
        A(JJ,JJ-1) = A1(JJ,I)                    ; <-P
        A(JJ,JJ)   = A2(JJ,I)                    ; <-P
        A(JJ,1)    = AT(2 ,I)                    ; <-P
```

```
      C(1 ,1   ,I) = C2(1 ,I)                    ; <-P
      C(1 ,2   ,I) = C3(1 ,I)                    ; <-P
      C(1 ,JJ-1,I) = C1(1 ,I)                    ; <-P
      C(1 ,JJ  ,I) = CT(1 ,I)                    ; <-P
      C(JJ,2   ,I) = C3(JJ,I)                    ; <-P
      C(JJ,JJ-1,I) = C1(JJ,I)                    ; <-P
      C(JJ,JJ  ,I) = C2(JJ,I)                    ; <-P
      C(JJ,1   ,I) = CT(2 ,I)                    ; <-P
      DO 131 J=2, JJ-1
        B(J,J-1) = B1(J,I)
        B(J,J  ) = B2(J,I)
        B(J,J+1) = B3(J,I)
        A(J,J-1) = A1(J,I)
        A(J,J  ) = A2(J,I)
        A(J,J+1) = A3(J,I)
        C(J,J-1,I) = C1(J,I)
        C(J,J  ,I) = C2(J,I)
        C(J,J+1,I) = C3(J,I)
131   CONTINUE
      DO 132 J=1, JJ-1
        K = J+JJ
        B(K,J  ) = B6(J,I)
        B(K,J+1) = B7(J,I)
        A(K,J  ) = A6(J,I)
        A(K,J+1) = A7(J,I)
        C(K,J  ,I) = C6(J,I)
        C(K,J+1,I) = C7(J,I)
132   CONTINUE
      B(K2+1,1)    = BVC(1,1,I)
      B(K2+2,1)    = BVT(1,1,I)
      B(K2+3,1)    = BVH(1,1,I)
      B(K2+1,2)    = BVC(1,2,I)
      B(K2+2,2)    = BVT(1,2,I)
      B(K2+3,2)    = BVH(1,2,I)
      B(K2+4,JJ-1) = BVC(2,1,I)
      B(K2+5,JJ-1) = BVT(2,1,I)
      B(K2+6,JJ-1) = BVH(2,1,I)
      B(K2+4,JJ)   = BVC(2,2,I)
      B(K2+5,JJ)   = BVT(2,2,I)
      B(K2+6,JJ)   = BVH(2,2,I)
      A(K2+1,1)    = AVC(1,1,I)
      A(K2+2,1)    = AVT(1,1,I)
      A(K2+3,1)    = AVH(1,1,I)
      A(K2+1,2)    = AVC(1,2,I)
      A(K2+2,2)    = AVT(1,2,I)
      A(K2+3,2)    = AVH(1,2,I)
      A(K2+4,JJ-1) = AVC(2,1,I)
      A(K2+5,JJ-1) = AVT(2,1,I)
      A(K2+6,JJ-1) = AVH(2,1,I)
```

```
          A(K2+4,JJ)    = AVC(2,2,I)
          A(K2+5,JJ)    = AVT(2,2,I)
          A(K2+6,JJ)    = AVH(2,2,I)
          C(K2+1,1,I)     = CVC(1,1,I)
          C(K2+2,1,I)     = CVT(1,1,I)
          C(K2+3,1,I)     = CVH(1,1,I)
          C(K2+1,2,I)     = CVC(1,2,I)
          C(K2+2,2,I)     = CVT(1,2,I)
          C(K2+3,2,I)     = CVH(1,2,I)
          C(K2+4,JJ-1,I) = CVC(2,1,I)
          C(K2+5,JJ-1,I) = CVT(2,1,I)
          C(K2+6,JJ-1,I) = CVH(2,1,I)
          C(K2+4,JJ,I)    = CVC(2,2,I)
          C(K2+5,JJ,I)    = CVT(2,2,I)
          C(K2+6,JJ,I)    = CVH(2,2,I)
C
          DO 14 N=1, 6
            L = K2+N
            B(1 ,L) = BI(1,N,I)
            B(JJ,L) = BI(2,N,I)
            A(1 ,L) = AI(1,N,I)
            A(JJ,L) = AI(2,N,I)
   14     CONTINUE
C
          DO 15 L=1, 3
            B(K2+1,K2+L) = BVC(1,L+3,I)
            B(K2+2,K2+L) = BVT(1,L+3,I)
            B(K2+3,K2+L) = BVH(1,L+3,I)
            A(K2+1,K2+L) = AVC(1,L+3,I)
            A(K2+2,K2+L) = AVT(1,L+3,I)
            A(K2+3,K2+L) = AVH(1,L+3,I)
            B(K2+4,K2+L+3) = BVC(2,L+3,I)
            B(K2+5,K2+L+3) = BVT(2,L+3,I)
            B(K2+6,K2+L+3) = BVH(2,L+3,I)
            A(K2+4,K2+L+3) = AVC(2,L+3,I)
            A(K2+5,K2+L+3) = AVT(2,L+3,I)
            A(K2+6,K2+L+3) = AVH(2,L+3,I)
   15     CONTINUE
C
CCC**** Do JS shift
          IF(I.LT.II-1) THEN
           Z(KS,KS  ) = 0.
           B(KS,JS  ) = 0.
           B(KS,JS+1) = 0.
           B(KS,KS  ) = Z8(JS,I+1)
           A(KS,JS  ) = B6(JS,I+1)
           A(KS,JS+1) = B7(JS,I+1)
           A(KS,KS  ) = B8(JS,I+1)
           C(KS,JS  ,I) = A6(JS,I+1)
```

```
          C(KS,JS+1,I) = A7(JS,I+1)
          DR(KS,NRHS+1,I) = A8(JS,I+1)
          DR(KS,NRHS+2,I) = C6(JS,I+1)
          DR(KS,NRHS+3,I) = C7(JS,I+1)
          DO 151 L = 1, NRHS
            DR(KS,L,I) = DR(KS,L,I+1)
  151     CONTINUE
C
         ELSE IF(I.EQ.II-1) THEN
          DO 152 K=1, K2
            Z(KS,K) = 0.
            B(KS,K) = 0.
            A(KS,K) = 0.
            C(KS,K,I) = 0.
  152     CONTINUE
          DO 153 L=1, NRHS
            DR(KS,L,I) = 0.
  153     CONTINUE
          DO 154 J=1,JJ-1
            K = K1+J-1
            B(KS,K  ) = B(KS,K  ) + Z8(J,I+1)
            A(KS,J  ) = A(KS,J  ) + B6(J,I+1)
            A(KS,J+1) = A(KS,J+1) + B7(J,I+1)
            A(KS,K  ) = A(KS,K  ) + B8(J,I+1)
            C(KS,J  ,I)=C(KS,J  ,I) + A6(J,I+1)
            C(KS,J+1,I)=C(KS,J+1,I) + A7(J,I+1)
            DO 1541 L = 1, NRHS
              DR(KS,L,I) = DR(KS,L,I) + DR(K,L,I+1)
 1541       CONTINUE
  154     CONTINUE
         ENDIF
C
         ENDIF
C
CCC**** Eliminate right Z block
         IF(I.GE.3) THEN
         DO 16 K=1, NBK
           DO 161 J=K1, K2
             IF(Z(K,J).EQ.0.0) GO TO 161
             DO 1611 L=1, JJ
               B(K,L) = B(K,L) - Z(K,J)*C(J,L,IL)
 1611        CONTINUE
             DO 1612 L=1, NRHS
               DR(K,L,I) = DR(K,L,I) - Z(K,J)*DR(J,L,IL)
 1612        CONTINUE
             KS = K1+JS-1
             B(K,KS  ) = B(K,KS  ) - Z(K,J)*DR(J,NRHS+1,IL)
             A(K,JS  ) = A(K,JS  ) - Z(K,J)*DR(J,NRHS+2,IL)
             A(K,JS+1) = A(K,JS+1) - Z(K,J)*DR(J,NRHS+3,IL)
```

```
  161     CONTINUE
   16   CONTINUE
        ENDIF
C
CCC**** Eliminate B block
        IF(I.GE.2) THEN
        DO 17 K=1, NBK
          DO 171 J=1, NBK
            IF(B(K,J).EQ.0.0) GO TO 171
            DO 1711 L=1, JJ
              A(K,L) = A(K,L) - B(K,J)*C(J,L,IM)
 1711       CONTINUE
            DO 1712 L=1, NRHS
              DR(K,L,I) = DR(K,L,I) - B(K,J)*DR(J,L,IM)
 1712       CONTINUE
            A(K,KS    ) = A(K,KS    ) - B(K,J)*DR(J,NRHS+1,IM)
            C(K,JS  ,I) = C(K,JS  ,I) - B(K,J)*DR(J,NRHS+2,IM)
            C(K,JS+1,I) = C(K,JS+1,I) - B(K,J)*DR(J,NRHS+3,IM)
  171     CONTINUE
   17   CONTINUE
        ENDIF
C
CCC**** Invert A block
        DO 20 NP=NBK, 2, -1
          NM1 = NP-1
          PIVOT = 1.0/A(NP,NP)
C
C-------- normalize pivot row
          DO 201 L=1, NM1
            A(NP,L) = A(NP,L)*PIVOT
  201     CONTINUE
          DO 202 L=1, JJ
            C(NP,L,I) = C(NP,L,I)*PIVOT
  202     CONTINUE
          DO 203 L=1, NRHS+3
            DR(NP,L,I) = DR(NP,L,I)*PIVOT
  203     CONTINUE
C
C-------- eliminate pivot column
          DO 210 K=1, NM1
            IF(A(K,NP).EQ.0.0) GO TO 210
            DO 2101 L=1, NM1
              A(K,L) = A(K,L) - A(K,NP)*A(NP,L)
 2101       CONTINUE
            DO 2102 L=1, JJ
              C(K,L,I) = C(K,L,I) - A(K,NP)*C(NP,L,I)
 2102       CONTINUE
            DO 2103 L=1, NRHS+3
              DR(K,L,I) = DR(K,L,I) - A(K,NP)*DR(NP,L,I)
```

```
 2103       CONTINUE
  210     CONTINUE
   20   CONTINUE
C
C------ normalize last row
        PIVOT = 1.0 / A(1,1)
        DO 21 L=1, JJ
          C(1,L,I) = C(1,L,I)*PIVOT
   21   CONTINUE
        DO 22 L=1, NRHS+3
          DR(1,L,I) = DR(1,L,I)*PIVOT
   22   CONTINUE
C
C------ back substitute
        DO 24 NP=2, NBK
          NM1 = NP-1
          DO 241 K=1, NM1
            IF(A(NP,K).EQ.0.0) GO TO 241
            DO 2411 L=1, JJ
              C(NP,L,I) = C(NP,L,I) - A(NP,K)*C(K,L,I)
 2411       CONTINUE
            DO 2412 L=1, NRHS+3
              DR(NP,L,I) = DR(NP,L,I) - A(NP,K)*DR(K,L,I)
 2412       CONTINUE
  241     CONTINUE
   24   CONTINUE
C
 1000 CONTINUE
C
CCC** Backward sweep: Elimination of upper block diagonal (C's).
      DO 2000 I=II-1, 1, -1
        IP = I+1
        IQ = I+2
        DO 51 N=1, NBK
          DO 511 L=1, NRHS
            DO 5111 K=1, JJ
              DR(N,L,I) = DR(N,L,I) - C(N,K,I)*DR(K,L,IP)
 5111       CONTINUE
            IF(I.NE.II-1) DR(N,L,I) = DR(N,L,I)
     &                       - DR(N,NRHS+1,I)*DR(KS,L,IP)
     &                       - DR(N,NRHS+2,I)*DR(JS,L,IQ)
     &                       - DR(N,NRHS+3,I)*DR(JS+1,L,IQ)
  511     CONTINUE
   51   CONTINUE
 2000 CONTINUE
C
      RETURN
      END ; FSOLVE
```

ISOLVE is the subroutine that solves the linear system of Newton equations by an iterative method.

```
      SUBROUTINE SOLVE
$INCLUDE STATE.INC
$INCLUDE ISES.INC
      COMMON/ISOL/ DS(JW,NGLX,-1:IX+1),D(IX,NGLX+2)
C
CCC** Zero changes
      DO 1 I=-1, II+1
        DO 11 K=1, 2*JJ-1
          DS(K,1,I) = 0.
   11   CONTINUE
    1 CONTINUE
C
CCC** Call pre-processor
      CALL PRE
C
CCC** Iterate, sweeping downstream and upstream
C     WRITE(5,*) 'Input NITREL, UNDER'
C     READ(5,*) NITREL, UNDER
C     WRITE(5,*) ' '
      NITREL = 21
      UNDER = 1.0
C
      WRITE(5,*) ' '
      DO 2 ITREL = 1, NITREL
        RESRMS = 0.
        RESMAX = 0.
        IMAX = 0
        JMAX = 0
        LMAX = 0
        DO 21 I = 1, II
          CALL RELAXN(I,UNDER,RESRMS,RESMAX,IMAX,JMAX,LMAX)
   21   CONTINUE
        DO 22 J = 2, JJ-1
          CALL RELAXS(J,UNDER,RESRMS,RESMAX,IMAX,JMAX,LMAX)
   22   CONTINUE
        RESRMS = SQRT(RESRMS/(2*II*JJ*NRHS))
        IF(MOD(ITREL,10).EQ.1) WRITE(5,*) 'Rms, max residual = ',
     &  RESRMS,RESMAX,'  at I,J,L = ',IMAX,JMAX,LMAX
    2 CONTINUE
C
CCC** Call post-processor
      CALL POST
C
      RETURN
      END ; SOLVE
```

```
      SUBROUTINE PRE
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
      DO 1 IO = 1, II-1
        IL = IO-2
        IM = IO-1
        IP = IO+1
C
        DO 11 JO = 1, JJ-1
          JP = JO+1
C
C-------- Process n-mom equations
          IF(IO.GT.2) THEN
           Z2(JO,IO) = Z2(JO,IO) + Z5(JO,IO)*DR1N1M(JO,IL)
           Z3(JO,IO) = Z3(JO,IO) + Z5(JO,IO)*DR1N1P(JO,IL)
           Z1(JP,IO) = Z1(JP,IO) + Z4(JP,IO)*DR1N1M(JO,IL)
           Z2(JP,IO) = Z2(JP,IO) + Z4(JP,IO)*DR1N1P(JO,IL)
           B2(JO,IO) = B2(JO,IO) + Z5(JO,IO)*DR1N2M(JO,IL)
           B3(JO,IO) = B3(JO,IO) + Z5(JO,IO)*DR1N2P(JO,IL)
           B1(JP,IO) = B1(JP,IO) + Z4(JP,IO)*DR1N2M(JO,IL)
           B2(JP,IO) = B2(JP,IO) + Z4(JP,IO)*DR1N2P(JO,IL)
          ENDIF
C
          IF(IO.GT.1) THEN
           B2(JO,IO) = B2(JO,IO) + B5(JO,IO)*DR1N1M(JO,IM)
           B3(JO,IO) = B3(JO,IO) + B5(JO,IO)*DR1N1P(JO,IM)
           B1(JP,IO) = B1(JP,IO) + B4(JP,IO)*DR1N1M(JO,IM)
           B2(JP,IO) = B2(JP,IO) + B4(JP,IO)*DR1N1P(JO,IM)
           A2(JO,IO) = A2(JO,IO) + B5(JO,IO)*DR1N2M(JO,IM)
           A3(JO,IO) = A3(JO,IO) + B5(JO,IO)*DR1N2P(JO,IM)
           A1(JP,IO) = A1(JP,IO) + B4(JP,IO)*DR1N2M(JO,IM)
           A2(JP,IO) = A2(JP,IO) + B4(JP,IO)*DR1N2P(JO,IM)
          ENDIF
C
          A2(JO,IO) = A2(JO,IO) + A5(JO,IO)*DR1N1M(JO,IO)
          A3(JO,IO) = A3(JO,IO) + A5(JO,IO)*DR1N1P(JO,IO)
          A1(JP,IO) = A1(JP,IO) + A4(JP,IO)*DR1N1M(JO,IO)
          A2(JP,IO) = A2(JP,IO) + A4(JP,IO)*DR1N1P(JO,IO)
          C2(JO,IO) = C2(JO,IO) + A5(JO,IO)*DR1N2M(JO,IO)
          C3(JO,IO) = C3(JO,IO) + A5(JO,IO)*DR1N2P(JO,IO)
          C1(JP,IO) = C1(JP,IO) + A4(JP,IO)*DR1N2M(JO,IO)
          C2(JP,IO) = C2(JP,IO) + A4(JP,IO)*DR1N2P(JO,IO)
C
C-------- Process s-mom equations
          IF(IO.GT.2) THEN
```

```
            Z6(JO,IO) = Z6(JO,IO) + Z8(JO,IO)*DR1N1M(JO,IL)
            Z7(JO,IO) = Z7(JO,IO) + Z8(JO,IO)*DR1N1P(JO,IL)
            B6(JO,IO) = B6(JO,IO) + Z8(JO,IO)*DR1N2M(JO,IL)
            B7(JO,IO) = B7(JO,IO) + Z8(JO,IO)*DR1N2P(JO,IL)
           ENDIF
C
           IF(IO.GT.1) THEN
            B6(JO,IO) = B6(JO,IO) + B8(JO,IO)*DR1N1M(JO,IM)
            B7(JO,IO) = B7(JO,IO) + B8(JO,IO)*DR1N1P(JO,IM)
            A6(JO,IO) = A6(JO,IO) + B8(JO,IO)*DR1N2M(JO,IM)
            A7(JO,IO) = A7(JO,IO) + B8(JO,IO)*DR1N2P(JO,IM)
           ENDIF
C
           A6(JO,IO) = A6(JO,IO) + A8(JO,IO)*DR1N1M(JO,IO)
           A7(JO,IO) = A7(JO,IO) + A8(JO,IO)*DR1N1P(JO,IO)
           C6(JO,IO) = C6(JO,IO) + A8(JO,IO)*DR1N2M(JO,IO)
           C7(JO,IO) = C7(JO,IO) + A8(JO,IO)*DR1N2P(JO,IO)
C
   11    CONTINUE
C
         JO = JJ-1
         JP = 1
C
         IF(IO.GT.2) THEN
          Z1(JP,IO) = Z1(JP,IO) + Z4(JP,IO)*DR1N1M(JO,IL)
          ZT(1 ,IO) = ZT(1 ,IO) + Z4(JP,IO)*DR1N1P(JO,IL)
          B1(JP,IO) = B1(JP,IO) + Z4(JP,IO)*DR1N2M(JO,IL)
          BT(1 ,IO) = BT(1 ,IO) + Z4(JP,IO)*DR1N2P(JO,IL)
         ENDIF
C
         IF(IO.GT.1) THEN
          B1(JP,IO) = B1(JP,IO) + B4(JP,IO)*DR1N1M(JO,IM)
          BT(1 ,IO) = BT(1 ,IO) + B4(JP,IO)*DR1N1P(JO,IM)
          A1(JP,IO) = A1(JP,IO) + B4(JP,IO)*DR1N2M(JO,IM)
          AT(1 ,IO) = AT(1 ,IO) + B4(JP,IO)*DR1N2P(JO,IM)
         ENDIF
C
         A1(JP,IO) = A1(JP,IO) + A4(JP,IO)*DR1N1M(JO,IO)
         AT(1 ,IO) = AT(1 ,IO) + A4(JP,IO)*DR1N1P(JO,IO)
         C1(JP,IO) = C1(JP,IO) + A4(JP,IO)*DR1N2M(JO,IO)
         CT(1 ,IO) = CT(1 ,IO) + A4(JP,IO)*DR1N2P(JO,IO)
C
         JO = JJ
         JP = 1
C
         IF(IO.GT.2) THEN
          ZT(2 ,IO) = ZT(2 ,IO) + Z5(JO,IO)*DR1N1M(JP,IL)
          Z3(JO,IO) = Z3(JO,IO) + Z5(JO,IO)*DR1N1P(JP,IL)
          BT(2 ,IO) = BT(2 ,IO) + Z5(JO,IO)*DR1N2M(JP,IL)
```

```
          B3(JO,IO) = B3(JO,IO) + Z5(JO,IO)*DR1N2P(JP,IL)
         ENDIF
C
         IF(IO.GT.1) THEN
          BT(2 ,IO) = BT(2 ,IO) + B5(JO,IO)*DR1N1M(JP,IM)
          B3(JO,IO) = B3(JO,IO) + B5(JO,IO)*DR1N1P(JP,IM)
          AT(2 ,IO) = AT(2 ,IO) + B5(JO,IO)*DR1N2M(JP,IM)
          A3(JO,IO) = A3(JO,IO) + B5(JO,IO)*DR1N2P(JP,IM)
         ENDIF
C
         AT(2 ,IO) = AT(2 ,IO) + A5(JO,IO)*DR1N1M(JP,IO)
         A3(JO,IO) = A3(JO,IO) + A5(JO,IO)*DR1N1P(JP,IO)
         CT(2 ,IO) = CT(2 ,IO) + A5(JO,IO)*DR1N2M(JP,IO)
         C3(JO,IO) = C3(JO,IO) + A5(JO,IO)*DR1N2P(JP,IO)
C
    1  CONTINUE
C
       RETURN
       END



       SUBROUTINE POST
$INCLUDE STATE.INC
$INCLUDE ISES.INC
       COMMON/ISOL/ DS(JW,NGLX,-1:IX+1),D(IX,NGLX+2)
C
       DO 1 IO = 1, II
         IP = IO+1
         DO 11 L = 1, NRHS
           DO 111 JO = 1, JJ
             DR(JO,L,IO) = DS(JO,L,IO)
  111      CONTINUE
           DO 112 JO = 1, JJ-1
             JP = JO+1
             JZ = JO+JJ
             DR(JZ,L,IO) = DS(JZ,L,IO) + DR1N1M(JO,IO)*DS(JO,L,IO)
     &                                 + DR1N1P(JO,IO)*DS(JP,L,IO)
     &                                 + DR1N2M(JO,IO)*DS(JO,L,IP)
     &                                 + DR1N2P(JO,IO)*DS(JP,L,IP)
  112      CONTINUE
   11    CONTINUE
    1  CONTINUE
C
       RETURN
       END
```

```
C
      SUBROUTINE RELAXN(IO,UNDER,RESRMS,RESMAX,IMAX,JMAX,LMAX)
$INCLUDE STATE.INC
$INCLUDE ISES.INC
      COMMON/ISOL/ DS(JW,NGLX,-1:IX+1),D(IX,NGLX+2)
      LOGICAL LPER
C
      IL = IO-2
      IM = IO-1
      IP = IO+1
C
CCC** First relax entropy variables
C
      IF(IO.NE.II) THEN
       DO 1 JO = 1, JJ-1
         JP = JO+1
         JZ = JO+JJ
         DO 101 L = 1, NRHS
           RES = DR(JZ,L,IO) - Z6(JO,IO) * DS(JO,L,IL)
     &                       - Z7(JO,IO) * DS(JP,L,IL)
     &                       - Z8(JO,IO) * DS(JZ,L,IL)
     &                       - B6(JO,IO) * DS(JO,L,IM)
     &                       - B7(JO,IO) * DS(JP,L,IM)
     &                       - B8(JO,IO) * DS(JZ,L,IM)
     &                       - A6(JO,IO) * DS(JO,L,IO)
     &                       - A7(JO,IO) * DS(JP,L,IO)
     &                       - A8(JO,IO) * DS(JZ,L,IO)
     &                       - C6(JO,IO) * DS(JO,L,IP)
     &                       - C7(JO,IO) * DS(JP,L,IP)
C
           RESRMS = RESRMS + RES**2
           IF(ABS(RES).GT.ABS(RESMAX)) THEN
            RESMAX = RES
            IMAX = IO
            JMAX = JO
            LMAX = L
           ENDIF
           DS(JZ,L,IO) = DS(JZ,L,IO) + RES/A8(JO,IO)
C          WRITE(5,*) 'JO= ',JO,'  Entropy RES= ',RES
  101    CONTINUE
    1   CONTINUE
      ENDIF
C
CCC** Second relax node position variables
C
      LPER = (A1(1,IO).NE.0.) .OR. (AT(1,IO).NE.0.) .OR.
     &       (AT(2,IO).NE.0.) .OR. (A3(JJ,IO).NE.0.)
C
```

```
      JO = 1
      JM = JJ-1
      J1 = 2*JJ-1
      J2 = JJ+1
      BB(1) = 0.
      AA(1) = A2(1,IO)
      CC(1) = A3(1,IO)
      DO 2 L = 1, NRHS
        RES = DR(1,L,IO) - Z1(JO,IO) * DS(JM,L,IL)
     &                   - ZT(1 ,IO) * DS(JJ,L,IL)
     &                   - Z2(JO,IO) * DS(1 ,L,IL)
     &                   - Z3(JO,IO) * DS(2 ,L,IL)
     &                   - Z4(JO,IO) * DS(J1,L,IL)
     &                   - Z5(JO,IO) * DS(J2,L,IL)
     &                   - B1(JO,IO) * DS(JM,L,IM)
     &                   - BT(1 ,IO) * DS(JJ,L,IM)
     &                   - B2(JO,IO) * DS(1 ,L,IM)
     &                   - B3(JO,IO) * DS(2 ,L,IM)
     &                   - B4(JO,IO) * DS(J1,L,IM)
     &                   - B5(JO,IO) * DS(J2,L,IM)
        RES = RES        - A1(JO,IO) * DS(JM,L,IO)
     &                   - AT(1 ,IO) * DS(JJ,L,IO)
     &                   - A2(JO,IO) * DS(1 ,L,IO)
     &                   - A3(JO,IO) * DS(2 ,L,IO)
     &                   - A4(JO,IO) * DS(J1,L,IO)
     &                   - A5(JO,IO) * DS(J2,L,IO)
     &                   - C1(JO,IO) * DS(JM,L,IP)
     &                   - CT(1 ,IO) * DS(JJ,L,IP)
     &                   - C2(JO,IO) * DS(1 ,L,IP)
     &                   - C3(JO,IO) * DS(2 ,L,IP)
        RESRMS = RESRMS + RES**2
        IF(ABS(RES).GT.ABS(RESMAX)) THEN
         RESMAX = RES
         IMAX = IO
         JMAX = 1
         LMAX = L
        ENDIF
        D(1,L) = RES
C       WRITE(5,*) 'JO= 1','  Node RES= ',RES
    2 CONTINUE
      D(1,NRHS+1) = A1(1,IO)
      D(1,NRHS+2) = AT(1,IO)
C
      DO 3 JO = 2, JJ-1
        JM = JO-1
        JP = JO+1
        J1 = JO+JJ-1
        J2 = JO+JJ
        BB(JO) = A1(JO,IO)
```

```
          AA(JO) = A2(JO,IO)
          CC(JO) = A3(JO,IO)
          DO 31 L = 1, NRHS
            RES = DR(JO,L,IO) - Z1(JO,IO) * DS(JM,L,IL)
     &                        - Z2(JO,IO) * DS(JO,L,IL)
     &                        - Z3(JO,IO) * DS(JP,L,IL)
     &                        - Z4(JO,IO) * DS(J1,L,IL)
     &                        - Z5(JO,IO) * DS(J2,L,IL)
     &                        - B1(JO,IO) * DS(JM,L,IM)
     &                        - B2(JO,IO) * DS(JO,L,IM)
     &                        - B3(JO,IO) * DS(JP,L,IM)
     &                        - B4(JO,IO) * DS(J1,L,IM)
     &                        - B5(JO,IO) * DS(J2,L,IM)
     &                        - A1(JO,IO) * DS(JM,L,IO)
     &                        - A2(JO,IO) * DS(JO,L,IO)
     &                        - A3(JO,IO) * DS(JP,L,IO)
     &                        - A4(JO,IO) * DS(J1,L,IO)
     &                        - A5(JO,IO) * DS(J2,L,IO)
     &                        - C1(JO,IO) * DS(JM,L,IP)
     &                        - C2(JO,IO) * DS(JO,L,IP)
     &                        - C3(JO,IO) * DS(JP,L,IP)
            RESRMS = RESRMS + RES**2
            IF(ABS(RES).GT.ABS(RESMAX)) THEN
             RESMAX = RES
             IMAX = IO
             JMAX = JO
             LMAX = L
            ENDIF
            D(JO,L) = RES
C           WRITE(5,*) 'JO= ',JO,'  Node RES= ',RES
   31     CONTINUE
          D(JO,NRHS+1) = 0.
          D(JO,NRHS+2) = 0.
    3   CONTINUE
C
        BB(JJ) = A1(JJ,IO)
        AA(JJ) = A2(JJ,IO)
        CC(JJ) = 0.
        JM = JJ-1
        J1 = JJ+1
        J2 = 2*JJ-1
        DO 4 L = 1, NRHS
          RES = DR(JJ,L,IO) - Z1(JJ,IO) * DS(JM,L,IL)
     &                      - Z2(JJ,IO) * DS(JJ,L,IL)
     &                      - ZT(2 ,IO) * DS(1 ,L,IL)
     &                      - Z3(JJ,IO) * DS(2 ,L,IL)
     &                      - Z4(JJ,IO) * DS(J1,L,IL)
     &                      - Z5(JJ,IO) * DS(J2,L,IL)
     &                      - B1(JJ,IO) * DS(JM,L,IM)
```

```
     &                     - B2(JJ,IO) * DS(JJ,L,IM)
     &                     - BT(2 ,IO) * DS(1 ,L,IM)
     &                     - B3(JJ,IO) * DS(2 ,L,IM)
     &                     - B4(JJ,IO) * DS(J1,L,IM)
     &                     - B5(JJ,IO) * DS(J2,L,IM)
        RES = RES          - A1(JJ,IO) * DS(JM,L,IO)
     &                     - A2(JJ,IO) * DS(JJ,L,IO)
     &                     - AT(2 ,IO) * DS(1 ,L,IO)
     &                     - A3(JJ,IO) * DS(2 ,L,IO)
     &                     - A4(JJ,IO) * DS(J1,L,IO)
     &                     - A5(JJ,IO) * DS(J2,L,IO)
     &                     - C1(JJ,IO) * DS(JM,L,IP)
     &                     - C2(JJ,IO) * DS(JJ,L,IP)
     &                     - CT(2 ,IO) * DS(1 ,L,IP)
     &                     - C3(JJ,IO) * DS(2 ,L,IP)
        RESRMS = RESRMS + RES**2
        IF(ABS(RES).GT.ABS(RESMAX)) THEN
         RESMAX = RES
         IMAX = IO
         JMAX = JJ
         LMAX = L
        ENDIF
        D(JJ,L) = RES
C       WRITE(5,*) 'JO= JJ','  Node RES= ',RES
    4 CONTINUE
      D(JJ,NRHS+1) = 0.
      D(JJ,NRHS+2) = 0.
C
C---- Solve for nonperiodic systems
      IF(.NOT.LPER) THEN
       DO 5 JO = 1, JJ-1
         JP = JO+1
         PIVOT = 1./AA(JO)
         CC(JO) = PIVOT*CC(JO)
         DO 51 L = 1, NRHS
           D(JO,L) = PIVOT*D(JO,L)
   51    CONTINUE
         AA(JP) = AA(JP) - BB(JP)*CC(JO)
         DO 52 L = 1, NRHS
           D(JP,L) = D(JP,L) - BB(JP)*D(JO,L)
   52    CONTINUE
    5  CONTINUE
C
       PIVOT = 1./AA(JJ)
       DO 6 L = 1, NRHS
         D(JJ,L) = PIVOT*D(JJ,L)
    6  CONTINUE
C
       DO 7 JO = JJ-1,1,-1
```

```
          DO 71 L = 1, NRHS
            D(JO,L) = D(JO,L) - CC(JO)*D(JO+1,L)
   71     CONTINUE
    7   CONTINUE
C
C---- Solve for periodic sytems
      ELSE
       N1 = NRHS+1
       N2 = NRHS+2
C
       DO 8 JO = 1, JJ-2
         JP = JO+1
         PIVOT = 1./AA(JO)
         CC(JO) = PIVOT*CC(JO)
         DO 81 L = 1, N2
           D(JO,L) = PIVOT*D(JO,L)
   81    CONTINUE
         AA(JP) = AA(JP) - BB(JP)*CC(JO)
         DO 82 L = 1, N2
           D(JP,L) = D(JP,L) - BB(JP)*D(JO,L)
   82    CONTINUE
    8  CONTINUE
C
       JM = JJ-1
       AA(JM) = AA(JM) + D(JM,N1)
       CC(JM) = CC(JM) + D(JM,N2)
       PIVOT = 1./AA(JM)
       CC(JM) = PIVOT*CC(JM)
       DO 9 L = 1, NRHS
         D(JM,L) = PIVOT*D(JM,L)
    9  CONTINUE
C
       PIVOT = 1./ (AA(JJ) - BB(JJ)*CC(JM))
       DO 10 L = 1, NRHS
         D(JJ,L) = PIVOT * (D(JJ,L) - BB(JJ)*D(JM,L))
   10  CONTINUE
       D(JJ,N1) = PIVOT*AT(2,IO)
       D(JJ,N2) = PIVOT*A3(JJ,IO)
C
       DO 11 L = 1, NRHS
         D(JM,L) = D(JM,L) - CC(JM)*D(JJ,L)
   11  CONTINUE
       D(JM,N1) = -CC(JM)*D(JJ,N1)
       D(JM,N2) = -CC(JM)*D(JJ,N2)
C
       DO 12 JO = JJ-2, 2, -1
         JP = JO+1
         T1 = D(JO,N1)
         T2 = D(JO,N2)
```

```
          DO 121 L = 1, NRHS
            D(JO,L) = D(JO,L) -CC(JO)*D(JP,L) -T1*D(JM,L) -T2*D(JJ,L)
  121     CONTINUE
          D(JO,N1) = -CC(JO)*D(JP,N1) -T1*D(JM,N1) -T2*D(JJ,N1)
          D(JO,N2) = -CC(JO)*D(JP,N2) -T1*D(JM,N2) -T2*D(JJ,N2)
   12   CONTINUE
C
        A21 = D(2,N1)
        A22 = D(2,N2) + 1.0
        T1 = D(1,N1)
        T2 = D(1,N2)
        DO 13 L = 1, NRHS
          D(1,L) = D(1,L) -T1*D(JM,L) -T2*D(JJ,L)
   13   CONTINUE
        A11 = -T1*D(JM,N1) -T2*D(JJ,N1) + 1.0
        A12 = -T1*D(JM,N2) -T2*D(JJ,N2) + CC(1)
        DETINV = 1./(A11*A22-A12*A21)
        A11INV =  A22*DETINV
        A12INV = -A12*DETINV
        A21INV = -A21*DETINV
        A22INV =  A11*DETINV
        DO 14 L = 1, NRHS
          T1 = D(1,L)
          T2 = D(2,L)
          D(1,L) = A11INV*T1 + A12INV*T2
          D(2,L) = A21INV*T1 + A22INV*T2
   14   CONTINUE
C
        DO 15 JO = 3, JJ
          DO 151 L = 1, NRHS
            D(JO,L) = D(JO,L) - D(JO,N1)*D(1,L) - D(JO,N2)*D(2,L)
  151     CONTINUE
   15   CONTINUE
C
      ENDIF
C
C---- Add onto DS
      DO 16 JO = 1, JJ
        DO 161 L = 1, NRHS
          DS(JO,L,IO) = DS(JO,L,IO) + UNDER*D(JO,L)
  161   CONTINUE
   16 CONTINUE
C
      RETURN
      END ; RELAXN
```

```
C
      SUBROUTINE RELAXS(JO,UNDER,RESRMS,RESMAX,IMAX,JMAX,LMAX)
$INCLUDE STATE.INC
$INCLUDE ISES.INC
      COMMON/ISOL/ DS(JW,NGLX,-1:IX+1),D(IX,NGLX+2)
      LOGICAL LPER
C
      JM = JO-1
      JP = JO+1
      J1 = JO+JJ-1
      J2 = JO+JJ
C
CCC** Relax node position variables
C
C
      DO 1 IO = 1, II
        IL = IO-2
        IM = IO-1
        IP = IO+1
        ZZ(IO) = Z2(JO,IO)
        BB(IO) = B2(JO,IO)
        AA(IO) = A2(JO,IO)
        CC(IO) = C2(JO,IO)
        DO 11 L = 1, NRHS
          RES = DR(JO,L,IO) - Z1(JO,IO) * DS(JM,L,IL)
     &                      - Z2(JO,IO) * DS(JO,L,IL)
     &                      - Z3(JO,IO) * DS(JP,L,IL)
     &                      - Z4(JO,IO) * DS(J1,L,IL)
     &                      - Z5(JO,IO) * DS(J2,L,IL)
     &                      - B1(JO,IO) * DS(JM,L,IM)
     &                      - B2(JO,IO) * DS(JO,L,IM)
     &                      - B3(JO,IO) * DS(JP,L,IM)
     &                      - B4(JO,IO) * DS(J1,L,IM)
     &                      - B5(JO,IO) * DS(J2,L,IM)
     &                      - A1(JO,IO) * DS(JM,L,IO)
     &                      - A2(JO,IO) * DS(JO,L,IO)
     &                      - A3(JO,IO) * DS(JP,L,IO)
     &                      - A4(JO,IO) * DS(J1,L,IO)
     &                      - A5(JO,IO) * DS(J2,L,IO)
     &                      - C1(JO,IO) * DS(JM,L,IP)
     &                      - C2(JO,IO) * DS(JO,L,IP)
     &                      - C3(JO,IO) * DS(JP,L,IP)
          RESRMS = RESRMS + RES**2
          IF(ABS(RES).GT.ABS(RESMAX)) THEN
           RESMAX = RES
```

```
            IMAX = IO
            JMAX = JO
            LMAX = L
           ENDIF
           D(IO,L) = RES
C          WRITE(5,*) 'JO= ',JO,'  Node RES= ',RES
   11    CONTINUE
    1  CONTINUE
C
C---- Solve
       DO 2 IO = 1, II
         IP = IO+1
         IQ = IO+2
         PIVOT = 1./AA(IO)
         CC(IO) = PIVOT*CC(IO)
         DO 21 L = 1, NRHS
           D(IO,L) = PIVOT*D(IO,L)
   21    CONTINUE
         IF(IP.GT.II) GOTO 2
          AA(IP) = AA(IP) - BB(IP)*CC(IO)
          DO 22 L = 1, NRHS
            D(IP,L) = D(IP,L) - BB(IP)*D(IO,L)
   22     CONTINUE
         IF(IQ.GT.II) GOTO 2
          BB(IQ) = BB(IQ) - ZZ(IQ)*CC(IO)
          DO 23 L = 1, NRHS
            D(IQ,L) = D(IQ,L) - ZZ(IQ)*D(IO,L)
   23     CONTINUE
    2  CONTINUE
C
       DO 3 IO = II-1,1,-1
         DO 31 L = 1, NRHS
           D(IO,L) = D(IO,L) - CC(IO)*D(IO+1,L)
   31    CONTINUE
    3  CONTINUE
C
C---- Add onto DS
      DO 4 IO = 1, II
        DO 41 L = 1, NRHS
          DS(JO,L,IO) = DS(JO,L,IO) + UNDER*D(IO,L)
   41   CONTINUE
    4 CONTINUE
C
      RETURN
      END ; RELAXS
```

UPDATE is the subroutine which calculates the global variables and then updates the solution, with clamping if necessary to prevent excessive changes.

```
      SUBROUTINE UPDATE
$INCLUDE STATE.INC
$INCLUDE ISES.INC
      CHARACTER*1 VVAR, SVAR
C
      RLXI = 1.0
      RLXV = 1.0
C
C---- calculate global iterates
      CALL GLOBIT
C
C---- subtract off extra righthand sides
      DO 10 I=1, II
        DO 11 J=1, 2*JJ+5
          DR(J,0,I) = 0.
          DR(J,1,I) = DR(J,1,I)
     &              - DNINL*DR(J,LNINL,I)
     &              - DNOUT*DR(J,LNOUT,I)
     &              - DCIRC*DR(J,LCIRC,I)
     &              - DALFA*DR(J,LALFA,I)
     &              - DSBLE*DR(J,LSBLE,I)
     &              - DPDF0*DR(J,LPDF0,I)
     &              - DPDF1*DR(J,LPDF1,I)
     &              - DPDFL*DR(J,LPDFL,I)
     &              - DPDX0*DR(J,LPDX0,I)
     &              - DPDX1*DR(J,LPDX1,I)
  11    CONTINUE
  10  CONTINUE
C
C---- clamp DSBLE
      DSLE = 0.50 * SQRT((X(ILE+1,1)-X(ILE+1,JJ))**2
     &                 + (Y(ILE+1,1)-Y(ILE+1,JJ)+PITCH)**2)
      IF(RLXI*ABS(DSBLE).GT.1.5*DSLE) RLXI = 1.5*DSLE/ABS(DSBLE)
C
C---- find max and rms changes
      DNMAX = 0.0
      DNRMS = 0.0
      DRMAX = 0.0
      DRRMS = 0.0
      INMAX = 0
      JNMAX = 0
      IRMAX = 0
      JRMAX = 0
      VVAR = ' '
```

```
      IVMAX = 0
      ISMAX = 0
C
      K2 = 2*JJ-1
      DO 40 I = 1, II-1
C
        DO 401 J = 1, JJ-1
          RRAT = DR(J+JJ,1,I)/R(I,J)
          IF(RLXI*RRAT.GT.1.0) RLXI = 1.0/RRAT          ;  clamp dRHO
          IF(RLXI*RRAT.LT.-.5) RLXI = -.5/RRAT
          IF(ABS(RRAT) .GT. ABS(DRMAX)) THEN
           DRMAX = RRAT
           IRMAX = I
           JRMAX = J
          ENDIF
          DRRMS = DRRMS + RRAT**2
  401   CONTINUE
C
        DO 402 J = 1, JJ
          IF(ABS(DR(J,1,I)) .GT. ABS(DNMAX)) THEN
           DNMAX = DR(J,1,I)
           INMAX = I
           JNMAX = J
          ENDIF
          DNRMS = DNRMS + DR(J,1,I)**2
  402   CONTINUE
C
        DHI = 1.7
        DLO = -.6
        IF(.NOT.LVISC .OR. I.LE.ILE) GO TO 40
         DO 403 IS=1, 2
           DV = DR(K2-2+3*IS,1,I)/UEDG(I,IS)
           IF(RLXV*DV.GT.DHI .OR. RLXV*DV.LT.DLO) THEN
            IVMAX = I
            ISMAX = IS
            VVAR = 'U'
           ENDIF
           IF(RLXV*DV.GT.DHI) RLXV = DHI/DV             ;  clamp dUEDGE
           IF(RLXV*DV.LT.DLO) RLXV = DLO/DV
C
           DV = DR(K2-1+3*IS,1,I)/THET(I,IS)
           IF(RLXV*DV.GT.DHI .OR. RLXV*DV.LT.DLO) THEN
            IVMAX = I
            ISMAX = IS
            VVAR = 'T'
           ENDIF
           IF(RLXV*DV.GT.DHI) RLXV = DHI/DV             ;  clamp dTHETA
           IF(RLXV*DV.LT.DLO) RLXV = DLO/DV
C
```

```
            DV = DR(K2  +3*IS,1,I)/DISP(I,IS)
            IF(RLXV*DV.GT.DHI .OR. RLXV*DV.LT.DLO) THEN
             IVMAX = I
             ISMAX = IS
             VVAR = 'D'
            ENDIF
            IF(RLXV*DV.GT.DHI) RLXV = DHI/DV            ;  clamp dDSTAR
            IF(RLXV*DV.LT.DLO) RLXV = DLO/DV
C
            DDIF = DR(K2-1+3*IS,1,I) - DR(K2+3*IS,1,I)
            VDIF = DISP(I,IS) - THET(I,IS)
            IF(VDIF.LT.RLXV*DDIF) THEN
             RLXV = 0.75*VDIF/DDIF                      ;  clamp dH
             IVMAX = I
             ISMAX = IS
             VVAR = 'H'
            ENDIF
C
  403    CONTINUE
C
   40 CONTINUE
C
C---- limit max iterate to CLAMP% relative change
      RLX = AMIN1(1.0,RLXI,RLXV)
C
C---- update density
      DO 50 I=1, II-1
        DO 501 J = 1, JJ-1
          R(I,J) = R(I,J) + RLX*DR(J+JJ,1,I)
 501    CONTINUE
  50  CONTINUE
C
C---- update grid
      DO 51 I=1, II
        DO 511 J=1, JJ
          X(I,J) = X(I,J) + RLX*DR(J,1,I)*NX(I,J)
          Y(I,J) = Y(I,J) + RLX*DR(J,1,I)*NY(I,J)
 511    CONTINUE
  51  CONTINUE
C
C---- adjust inlet stagnation streamline nodes
      SLEN = 0.
      DO 53 I=2, ILE
        SLEN = SLEN + SQRT((X(I,1)-X(I-1,1))**2 + (Y(I,1)-Y(I-1,1))**2)
   53 CONTINUE
      XOLD = X(1,1)
      YOLD = Y(1,1)
      SLENM = 0.
      DO 55 I=2, ILE-1
```

```
          DXM = X(I,1) - XOLD
          DYM = Y(I,1) - YOLD
          DXP = X(I+1,1) - X(I,1)
          DYP = Y(I+1,1) - Y(I,1)
          DSM = SQRT(DXM*DXM + DYM*DYM)
          DSP = SQRT(DXP*DXP + DYP*DYP)
          SLENO = SLENM + DSM
          DX = DXM + DXP
          DY = DYM + DYP
          DELTS = SLEN*SINL(I) - SLENO
          XOLD = X(I,1)
          YOLD = Y(I,1)
          X(I,1) = X(I,1) + DELTS * DX/SQRT(DX*DX + DY*DY)
          Y(I,1) = Y(I,1) + DELTS * DY/SQRT(DX*DX + DY*DY)
          X(I,JJ) = X(I,1)
          Y(I,JJ) = Y(I,1) + PITCH
          SLENM = SLENO
   55 CONTINUE
C
      IF(LVISC) THEN
C----- update BL parameters
       K2 = 2*JJ-1
       DO 62 I=ILE, II-1
         UEDG(I,1) = UEDG(I,1) + RLX*DR(K2+1,1,I)
         UEDG(I,2) = UEDG(I,2) + RLX*DR(K2+4,1,I)
         THET(I,1) = THET(I,1) + RLX*DR(K2+2,1,I)
         THET(I,2) = THET(I,2) + RLX*DR(K2+5,1,I)
         DISP(I,1) = DISP(I,1) + RLX*DR(K2+3,1,I)
         DISP(I,2) = DISP(I,2) + RLX*DR(K2+6,1,I)
  62   CONTINUE
      ENDIF
C
C---- update global variables
      SBLE = SBLE + RLX*DSBLE
      CIRC = CIRC + RLX*DCIRC
      ALFA = ALFA + RLX*DALFA
      PDF0 = PDF0 + RLX*DPDF0
      PDF1 = PDF1 + RLX*DPDF1
      PDFL = PDFL + RLX*DPDFL
      PDX0 = PDX0 + RLX*DPDX0
      PDX1 = PDX1 + RLX*DPDX1
C
C---- set new mass vector
      DO 65 J=1, JJ-1
        M(J) = MASS*MFRACT(J)
  65  CONTINUE
C
      IF(LDESI.OR.LMIXI) CALL NEWBLD   ; set and spline new blade geometry
      CALL NEWDIS                      ; offset grid from blade by Dstar
```

```
C
      DRRMS = SQRT(DRRMS/FLOAT((II-1)*(JJ-1)))
      DNRMS = SQRT(DNRMS/FLOAT((II-1)*(JJ-1)))
C
C---- set SMOVE flag if necessary
      IF(ABS(SBLE-SBLOLD).GT.0.5*DSLE) THEN
       LSMOVE = .TRUE.
       SVAR = 'S'
       SBLOLD = SBLE
      ELSE
       LSMOVE = .FALSE.
       SVAR = ' '
      ENDIF
C
      IF(RLX .LT.1.0) WRITE(5,*) ' '
      IF(RLXI.LT.1.0) WRITE(5,1200) RLXI
      IF(RLXV.LT.1.0) WRITE(5,1210) RLXV,VVAR,IVMAX,ISMAX
      WRITE(5,1240) ITER,SVAR, DRRMS, DRMAX, IRMAX, JRMAX
      WRITE(5,1250)             DNRMS, DNMAX, INMAX, JNMAX
      WRITE(5,1260)             DR(1,1,1), DR(1,1,II), DSBLE
      IF(LDESI) WRITE(5,1270) DPDF0, DPDF1, DPDFL
      IF(LMIXI) WRITE(5,1280) DPDX0, DPDX1
 1200 FORMAT(/X,'RLXI: ',F8.6)
 1210 FORMAT( X,'RLXV: ',F8.6,4X,A1,' clamped at I = ',I3,', side ',I3)
 1240 FORMAT(/X,I4,X,A,' rms(dR): ',E9.3,'  Max(dR): ',E9.3,'  at ',2I4)
 1250 FORMAT( X,6X,    ' rms(dN): ',E9.3,'  Max(dN): ',E9.3,'  at ',2I4)
 1260 FORMAT( X,6X,' Inl(dN): ',E9.3,'  Out(dN): ',E9.3,
     &             ' LE(dN) : ',E9.3)
 1270 FORMAT( X,6X,' dDOF0  : ',E9.3,'  dDOF1  : ',E9.3,
     &             ' dDOFL  : ',E9.3)
 1280 FORMAT( X,6X,' dDOF0  : ',E9.3,'  dDOF1  : ',E9.3)
      RETURN
      END ; UPDATE



      SUBROUTINE NEWBLD
$INCLUDE STATE.INC
$INCLUDE ISES.INC
      REAL XNEW(IBX), YNEW(IBX), SNEW(IBX)
C
C---- set new blade coordinates
      K2 = 2*JJ-1
      IB = 0
      SNEW(1) = 0.
      DO 80 I=ITE, ILE, -1
        IB = IB+1
        SS = (1.0 - SG(I-ILE+1,1))*SBLE
        XOLD = SEVAL(SS,XB,XPB,SB,IIB)
```

```
        YOLD = SEVAL(SS,YB,YPB,SB,IIB)
        DX   =  DEVAL(SS,YB,YPB,SB,IIB)
        DY   = -DEVAL(SS,XB,XPB,SB,IIB)
        DS = SQRT(DX*DX + DY*DY)
        IF(.NOT.LVISC) DX = 0.0
        IF(.NOT.LVISC) DY = 0.0
        XNEW(IB) = XOLD + (NX(I,1)*DR(1,1,I) - DX*DR(K2+3,1,I)/DS)*RLX
        YNEW(IB) = YOLD + (NY(I,1)*DR(1,1,I) - DY*DR(K2+3,1,I)/DS)*RLX
        IF(IB.GT.1) SNEW(IB) = SNEW(IB-1)
     &    + SQRT((XNEW(IB)-XNEW(IB-1))**2 + (YNEW(IB)-YNEW(IB-1))**2)
   80 CONTINUE
      IF(IBLE.GT.0) THEN
       IB=IB+1
       XNEW(IB) = XNEW(IB-1)
       YNEW(IB) = YNEW(IB-1)
       SNEW(IB) = SNEW(IB-1)
      ENDIF
      DX = 0.
      DY = 0.
      DS = 0.
      DO 82 I=ILE+1, ITE
        IB = IB+1
        SS = SBLE + (SB(IIB)-SBLE)*SG(I-ILE+1,2)
        XOLD = SEVAL(SS,XB,XPB,SB,IIB)
        YOLD = SEVAL(SS,YB,YPB,SB,IIB)
        DX   =  DEVAL(SS,YB,YPB,SB,IIB)
        DY   = -DEVAL(SS,XB,XPB,SB,IIB)
        DS = SQRT(DX*DX + DY*DY)
        IF(.NOT.LVISC) DX = 0.0
        IF(.NOT.LVISC) DY = 0.0
        XNEW(IB) = XOLD + (NX(I,JJ)*DR(JJ,1,I) - DX*DR(K2+6,1,I)/DS)*RLX
        YNEW(IB) = YOLD + (NY(I,JJ)*DR(JJ,1,I) - DY*DR(K2+6,1,I)/DS)*RLX
        SNEW(IB) = SNEW(IB-1)
     &    + SQRT((XNEW(IB)-XNEW(IB-1))**2 + (YNEW(IB)-YNEW(IB-1))**2)
   82 CONTINUE
C
C---- fix up TE gap
      TEGAP = SQRT((XB(IIB)-XB(1))**2 + (YB(IIB)-YB(1))**2)
      XTE = 0.5*(XB(1)   + XB(IIB)  )       ; preserve axial chord
      YTE = 0.5*(YNEW(1) + YNEW(IIB))
      XNEW(1)   = XTE - 0.5*DX*TEGAP/DS
      XNEW(IIB) = XTE + 0.5*DX*TEGAP/DS
      YNEW(1)   = YTE - 0.5*DY*TEGAP/DS
      YNEW(IIB) = YTE + 0.5*DY*TEGAP/DS
C
      IF(IBLE.EQ.0) THEN
       CALL SPLINE(XNEW,XPB,SNEW,IIB)
       CALL SPLINE(YNEW,YPB,SNEW,IIB)
      ELSE
```

```
       CALL SPLINE(XNEW,XPB,SNEW,IBLE)
       CALL SPLINE(YNEW,YPB,SNEW,IBLE)
       IBP = IBLE+1
       CALL SPLINE(XNEW(IBP),XPB(IBP),SNEW(IBP),IIB-IBLE)
       CALL SPLINE(YNEW(IBP),YPB(IBP),SNEW(IBP),IIB-IBLE)
      ENDIF
C
C---- set blade surface grid points to match SG arrays
      IB = 0
      SBLE = SNEW(NBLD)
      SB(1) = 0.
      DO 90 IG=NBLD, 1, -1
        IB = IB+1
        SS = SBLE * (1.0-SG(IG,1))
        XB(IB) = SEVAL(SS,XNEW,XPB,SNEW,IIB)
        YB(IB) = SEVAL(SS,YNEW,YPB,SNEW,IIB)
        IF(IB.GT.1) SB(IB) = SB(IB-1)
     &          + SQRT((XB(IB)-XB(IB-1))**2 + (YB(IB)-YB(IB-1))**2)
   90 CONTINUE
      IF(IBLE.GT.0) THEN
       IB=IB+1
       XB(IB) = XB(IB-1)
       YB(IB) = YB(IB-1)
       SB(IB) = SB(IB-1)
      ENDIF
      DO 92 IG=2, NBLD
        IB = IB+1
        SP = SBLE + (SNEW(IIB)-SBLE)*SG(IG,2)
        XB(IB) = SEVAL(SP,XNEW,XPB,SNEW,IIB)
        YB(IB) = SEVAL(SP,YNEW,YPB,SNEW,IIB)
        SB(IB) = SB(IB-1)
     &          + SQRT((XB(IB)-XB(IB-1))**2 + (YB(IB)-YB(IB-1))**2)
   92 CONTINUE
C
C---- respline blade arrays
      IF(IBLE.EQ.0) THEN
       CALL SPLINE(XB,XPB,SB,IIB)
       CALL SPLINE(YB,YPB,SB,IIB)
      ELSE
       CALL SPLINE(XB,XPB,SB,IBLE)
       CALL SPLINE(YB,YPB,SB,IBLE)
       IBP = IBLE+1
       CALL SPLINE(XB(IBP),XPB(IBP),SB(IBP),IIB-IBLE)
       CALL SPLINE(YB(IBP),YPB(IBP),SB(IBP),IIB-IBLE)
      ENDIF
      RETURN
      END ; NEWBLD
```

```
      SUBROUTINE NEWDIS
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
C---- set leading edge Dstar
      DISP(ILE,1) = 0.
      IF(IBLE.EQ.0) DISP(ILE,1) = 0.5*(DISP(ILE+1,1)+DISP(ILE+1,2))
      DISP(ILE,2) = DISP(ILE,1)
C
C---- offset grid from blade surface
      DO 10 IG=1, NBLD
        I = ILE + IG - 1
        DIS = 0.
        IF(LVISC) DIS = DISP(I,1)
        SS = SBLE * (1.0-SG(IG,1))
        DY = -DEVAL(SS,XB,XPB,SB,IIB)
        DX =  DEVAL(SS,YB,YPB,SB,IIB)
        DS = SQRT(DX*DX + DY*DY)
        X(I,1) = SEVAL(SS,XB,XPB,SB,IIB) + DIS*DX/DS
        Y(I,1) = SEVAL(SS,YB,YPB,SB,IIB) + DIS*DY/DS
   10 CONTINUE
      DO 20 IG=1, NBLD
        I = ILE + IG - 1
        DIS = 0.
        IF(LVISC) DIS = DISP(I,2)
        SS = SBLE + (SB(IIB)-SBLE)*SG(IG,2)
        DY = -DEVAL(SS,XB,XPB,SB,IIB)
        DX =  DEVAL(SS,YB,YPB,SB,IIB)
        DS = SQRT(DX*DX + DY*DY)
        X(I,JJ) = SEVAL(SS,XB,XPB,SB,IIB) + DIS*DX/DS
        Y(I,JJ) = SEVAL(SS,YB,YPB,SB,IIB) + DIS*DY/DS + PITCH
   20 CONTINUE
C
      RETURN
      END ; NEWDIS



      SUBROUTINE GLOBIT
$INCLUDE STATE.INC
$INCLUDE ISES.INC
      REAL QQ(NGLX,0:NGLX)
C
      CALL LCALC
C
      DO 888 K=1, NRHS
        DO 777 L=0, NRHS
          QQ(K,L) = 0.
  777   CONTINUE
```

```
  888 CONTINUE
C
      DO 1000 K=1, NGLOB
        GO TO (1,2,3,4,5,6,7,8,9,10,11,12), KGCON(K)
C
C------ Inlet slope
    1   DO 19   JO = 1, JJ-1
          JP = JO+1
          SX = X(2,JO) + X(2,JP) - X(1,JO) - X(1,JP)
          SY = Y(2,JO) + Y(2,JP) - Y(1,JO) - Y(1,JP)
          QQ(K,1) = QQ(K,1) + M(JO) * (SY/SX - SINLIN)
          DO 191 L = 1, NRHS
            DSX = NX(2,JO)*DR(JO,L,2) + NX(2,JP)*DR(JP,L,2)
     &          - NX(1,JO)*DR(JO,L,1) - NX(1,JP)*DR(JP,L,1)
            DSY = NY(2,JO)*DR(JO,L,2) + NY(2,JP)*DR(JP,L,2)
     &          - NY(1,JO)*DR(JO,L,1) - NY(1,JP)*DR(JP,L,1)
            QQ(K,L) = QQ(K,L) + M(JO) * (SX*DSY - SY*DSX) / SX**2
  191     CONTINUE
   19   CONTINUE
        GOTO 1000
C
C------ Outlet slope
    2   IM = II-1
        DO 21  JO = 1, JJ-1
          JP = JO+1
          SX = X(II,JO) + X(II,JP) - X(IM,JO) - X(IM,JP)
          SY = Y(II,JO) + Y(II,JP) - Y(IM,JO) - Y(IM,JP)
          QQ(K,1) = QQ(K,1) + M(JO) * (SY/SX - SOUTIN)
          DO 211 L = 1, NRHS
            DSX = NX(II,JO)*DR(JO,L,II) + NX(II,JP)*DR(JP,L,II)
     &          - NX(IM,JO)*DR(JO,L,IM) - NX(IM,JP)*DR(JP,L,IM)
            DSY = NY(II,JO)*DR(JO,L,II) + NY(II,JP)*DR(JP,L,II)
     &          - NY(IM,JO)*DR(JO,L,IM) - NY(IM,JP)*DR(JP,L,IM)
            QQ(K,L) = QQ(K,L) + M(JO) * (SX*DSY - SY*DSX) / SX**2
  211     CONTINUE
   21   CONTINUE
        GOTO 1000
C
C------ leading edge Kutta
    3   IO = ILE
        IM = IO - 1
        IP = IO + 1
        JO = 1
        JP = 2
        JM = JJ-1
        DO 31 L=1, NRHS
          QQ(K,L) = B4S(IO,2) * DR(JM+JJ,L,IM)
     &            + A4S(IO,2) * DR(JM+JJ,L,IO)
     &            + B5S(IO,1) * DR(JO+JJ,L,IM)
```

```
     &            + A5S(IO,1) * DR(JO+JJ,L,IO)
     &            + B2S(IO,1) * DR(JO,L,IM)
     &            + A2S(IO,1) * DR(JO,L,IO)
     &            + C2S(IO,1) * DR(JO,L,IP)
     &            + B3S(IO,1) * DR(JP,L,IM)
     &            + A3S(IO,1) * DR(JP,L,IO)
     &            + C3S(IO,1) * DR(JP,L,IP)
     &            + B1S(IO,2) * DR(JM,L,IM)
     &            + A1S(IO,2) * DR(JM,L,IO)
     &            + C1S(IO,2) * DR(JM,L,IP)
     &            + B2S(IO,2) * DR(JJ,L,IM)
     &            + A2S(IO,2) * DR(JJ,L,IO)
     &            + C2S(IO,2) * DR(JJ,L,IP)
  31   CONTINUE
       QQ(K,1) = QQ(K,1) + 2.0*(PI(IO,JJ)-PI(IO,1))
       GO TO 1000
C
C------ trailing edge Kutta
   4   IO = ITE
       IM = IO - 1
       IP = IO + 1
       JO = 1
       JP = 2
       JM = JJ-1
       DO 41 L=1, NRHS
         QQ(K,L) = B4S(IO,2) * DR(JM+JJ,L,IM)
     &           + A4S(IO,2) * DR(JM+JJ,L,IO)
     &           + B5S(IO,1) * DR(JO+JJ,L,IM)
     &           + A5S(IO,1) * DR(JO+JJ,L,IO)
     &           + B2S(IO,1) * DR(JO,L,IM)
     &           + A2S(IO,1) * DR(JO,L,IO)
     &           + C2S(IO,1) * DR(JO,L,IP)
     &           + B3S(IO,1) * DR(JP,L,IM)
     &           + A3S(IO,1) * DR(JP,L,IO)
     &           + C3S(IO,1) * DR(JP,L,IP)
     &           + B1S(IO,2) * DR(JM,L,IM)
     &           + A1S(IO,2) * DR(JM,L,IO)
     &           + C1S(IO,2) * DR(JM,L,IP)
     &           + B2S(IO,2) * DR(JJ,L,IM)
     &           + A2S(IO,2) * DR(JJ,L,IO)
     &           + C2S(IO,2) * DR(JJ,L,IP)
  41   CONTINUE
       QQ(K,1) = QQ(K,1) + 2.0*(PI(IO,JJ)-PI(IO,1))
       GO TO 1000
C
C------ Alpha
   5   QQ(K,LALFA) = 1.0
       QQ(K,1) = ALFAIN - ALFA
       GO TO 1000
```

```
C
C------ Lift
    6   JO = 1
        JP = 2
        JM = JJ-1
        DO 61 IO=ILE, ITE
          IM = IO-1
          IP = IO+1
C
          BXS = 0.5*(X(IP,JO)-X(IM,JO))
          BXP = 0.5*(X(IP,JJ)-X(IM,JJ))
          BYS = 0.5*(Y(IP,JO)-Y(IM,JO))
          BYP = 0.5*(Y(IP,JJ)-Y(IM,JJ))
          PIS = PI(IO,JO)
          PIP = PI(IO,JJ)
C
          DO 611 L=1, NRHS
            SUMS = B5S(IO,1) * DR(JO+JJ,L,IM)
     &           + A5S(IO,1) * DR(JO+JJ,L,IO)
     &           + B2S(IO,1) * DR(JO,L,IM)
     &           + A2S(IO,1) * DR(JO,L,IO)
     &           + C2S(IO,1) * DR(JO,L,IP)
     &           + B3S(IO,1) * DR(JP,L,IM)
     &           + C3S(IO,1) * DR(JP,L,IP)
     &           + A3S(IO,1) * DR(JP,L,IO)
            SUMP = B4S(IO,2) * DR(JM+JJ,L,IM)
     &           + A4S(IO,2) * DR(JM+JJ,L,IO)
     &           + B1S(IO,2) * DR(JM,L,IM)
     &           + A1S(IO,2) * DR(JM,L,IO)
     &           + C1S(IO,2) * DR(JM,L,IP)
     &           + B2S(IO,2) * DR(JJ,L,IM)
     &           + A2S(IO,2) * DR(JJ,L,IO)
     &           + C2S(IO,2) * DR(JJ,L,IP)
            DPIS = -.5*SUMS
            DPIP = 0.5*SUMP
            DBXS = 0.5*(NX(IP,JO)*DR(JO,L,IP) - NX(IM,JO)*DR(JO,L,IM))
            DBXP = 0.5*(NX(IP,JJ)*DR(JJ,L,IP) - NX(IM,JJ)*DR(JJ,L,IM))
C
            QQ(K,L) = QQ(K,L) + (BXP*DPIP - BXS*DPIS)
     &                        + (DBXP*PIP - DBXS*PIS)
  611     CONTINUE
   61   CONTINUE
C
        QQ(K,1) = QQ(K,1) + LIFT - LIFTIN
        GO TO 1000
C
C------ Zero leading edge gap
    7   DO 71 L=1, NRHS
          QQ(K,L) = DR(1,L,ILE) - DR(JJ,L,ILE)
```

```
   71   CONTINUE
        GO TO 1000
C
C------ Preserve trailing edge gap
    8   DX = X(ITE+1,1) + X(ITE+1,JJ) - X(ITE-1,1) - X(ITE-1,JJ)
        DY = Y(ITE+1,1) + Y(ITE+1,JJ) - Y(ITE-1,1) - Y(ITE-1,JJ)
        DS = SQRT(DX*DX + DY*DY)
        DX = -DY/DS
        DY =  DX/DS
CCC     DOTP = DX*NX(ITE,1) + DY*NY(ITE,1)
        K2 = 2*JJ-1
        DO 81 L=1, NRHS
          QQ(K,L) = (DR(1,L,ITE) - DR(JJ,L,ITE))
     &            - DR(K2+3,L,ITE) - DR(K2+6,L,ITE)
   81   CONTINUE
        GO TO 1000
C
C------ Zero leading edge movement
    9   JP = 1
        JM = JJ
        DO 91 L=1, NRHS
          QQ(K,L) = DR(JP,L,ILE) + DR(JM,L,ILE)
   91   CONTINUE
        GO TO 1000
C
C------ Zero trailing edge movement
   10   JP = 1
        JM = JJ
        DO 101 L=1, NRHS
          QQ(K,L) = DR(JP,L,ITE) + DR(JM,L,ITE)
  101   CONTINUE
        GO TO 1000
C
C------ Zero IX0 point movement
   11   IS = ISMIX
        K2 = 2*JJ-1
        IF(IS.EQ.1) THEN
         DO 111 L=1, NRHS
           QQ(K,L) = DR(1,L,IX0) - DR(K2+3,L,IX0)
  111    CONTINUE
        ELSE
         DO 112 L=1, NRHS
           QQ(K,L) = DR(JJ,L,IX0) + DR(K2+6,L,IX0)
  112    CONTINUE
        ENDIF
        GO TO 1000
C
C------ Zero IX1 point movement
   12   IS = ISMIX
```

```
        K2 = 2*JJ-1
        IF(IS.EQ.1) THEN
         DO 121 L=1, NRHS
           QQ(K,L) = DR(1,L,IX1) - DR(K2+3,L,IX1)
  121     CONTINUE
        ELSE
         DO 122 L=1, NRHS
           QQ(K,L) = DR(JJ,L,IX1) + DR(K2+6,L,IX1)
  122     CONTINUE
        ENDIF
C
 1000 CONTINUE
C
C     DO 333 KK= 1,NGLOB
C       WRITE(5,1234) (QQ(KK,KKK),KKK=1,NGLOB+1)
C333  CONTINUE
C1234 FORMAT(1X,10F9.5)
C
C---- STS
      IF(NGLOB.NE.0) CALL SOLVIT(NGLX,NGLOB,QQ(1,2),QQ(1,1))
      DNINL = 0.
      DNOUT = 0.
      DCIRC = 0.
      DALFA = 0.
      DSBLE = 0.
      DPDF0 = 0.
      DPDF1 = 0.
      DPDFL = 0.
      DPDX0 = 0.
      DPDX1 = 0.
      IF(LNINL.GT.0) DNINL = QQ(LNINL-1,1)
      IF(LNOUT.GT.0) DNOUT = QQ(LNOUT-1,1)
      IF(LCIRC.GT.0) DCIRC = QQ(LCIRC-1,1)
      IF(LALFA.GT.0) DALFA = QQ(LALFA-1,1)
      IF(LSBLE.GT.0) DSBLE = QQ(LSBLE-1,1)
      IF(LPDF0.GT.0) DPDF0 = QQ(LPDF0-1,1)
      IF(LPDF1.GT.0) DPDF1 = QQ(LPDF1-1,1)
      IF(LPDFL.GT.0) DPDFL = QQ(LPDFL-1,1)
      IF(LPDX0.GT.0) DPDX0 = QQ(LPDX0-1,1)
      IF(LPDX1.GT.0) DPDX1 = QQ(LPDX1-1,1)
C
      RETURN
      END ; GLOBIT



      SUBROUTINE SOLVIT(NSIZ,NN,Z,R)
      DIMENSION Z(NSIZ,NSIZ), R(NSIZ)
```

```
C
      DO 1 NP=1, NN-1
        NP1 = NP+1
C
C------ find max pivot index NX
        NX = NP
        DO 11 N=NP1, NN
          IF(ABS(Z(N,NP))-ABS(Z(NX,NP))) 11,11,111
  111      NX = N
   11   CONTINUE
C
        PIVOT = 1.0/Z(NX,NP)
C
C------ switch pivots
        Z(NX,NP) = Z(NP,NP)
C
C------ switch rows & normalize pivot row
        DO 12 L=NP1, NN
          TEMP = Z(NX,L)*PIVOT
          Z(NX,L) = Z(NP,L)
          Z(NP,L) = TEMP
   12   CONTINUE
C
        TEMP = R(NX)*PIVOT
        R(NX) = R(NP)
        R(NP) = TEMP
C
C------ forward eliminate everything
        DO 15 K=NP1, NN
          DO 151 L=NP1, NN
            Z(K,L) = Z(K,L) - Z(K,NP)*Z(NP,L)
  151     CONTINUE
          R(K) = R(K) - Z(K,NP)*R(NP)
   15   CONTINUE
C
    1 CONTINUE
C
C---- solve for last row
      R(NN) = R(NN)/Z(NN,NN)
C
C---- back substitute everything
      DO 2 NP=NN-1, 1, -1
        NP1 = NP+1
        DO 21 L=NP1, NN
          R(NP) = R(NP) - Z(NP,L)*R(L)
   21   CONTINUE
    2 CONTINUE
C
      RETURN
```

```
      END ; SOLVIT




C
      SUBROUTINE SPLIN2(XD,SD,IID,XS,XPS,SS,IIS)
C
      DIMENSION XD(300),SD(300),XS(300),XPS(300),SS(300)
      DIMENSION A(3,3,300),B(3,3,300),C(3,3,300),D(3,300)
C
      DO 1 IO = 1, IIS
        DO 11 J = 1, 3
          D(J,IO) = 0.
          DO 111 K = 1, 3
            B(J,K,IO) = 0.
            A(J,K,IO) = 0.
            C(J,K,IO) = 0.
 111      CONTINUE
  11    CONTINUE
   1  CONTINUE
C
      IO = 1
      IP = 2
      DSP = SS(IP) - SS(IO)
C
      A(3,1,IO) = A(3,1,IO) + 3.
      A(3,2,IO) = A(3,2,IO) + 2.*DSP
      C(3,1,IO) = C(3,1,IO) - 3.
      C(3,2,IO) = C(3,2,IO) + DSP
C
      A(1,3,IO) = A(1,3,IO) + 3.
      A(2,3,IO) = A(2,3,IO) + 2.*DSP
      B(1,3,IP) = B(1,3,IP) - 3.
      B(2,3,IP) = B(2,3,IP) + DSP
C
      DO 2 IO = 2, IIS-1
C
        IM = IO - 1
        IP = IO + 1
        DSM = SS(IO) - SS(IM)
        DSP = SS(IP) - SS(IO)
C
        B(3,1,IO) = B(3,1,IO) + 3.*(DSP/DSM)
        B(3,2,IO) = B(3,2,IO) + DSP
        A(3,1,IO) = A(3,1,IO) - 3.*(DSP/DSM - DSM/DSP)
        A(3,2,IO) = A(3,2,IO) + 2.*(DSM+DSP)
        C(3,1,IO) = C(3,1,IO) - 3.*(DSM/DSP)
```

```
        C(3,2,IO) = C(3,2,IO) + DSM
C
        C(1,3,IM) = C(1,3,IM) + 3.*(DSP/DSM)
        C(2,3,IM) = C(2,3,IM) + DSP
        A(1,3,IO) = A(1,3,IO) - 3.*(DSP/DSM - DSM/DSP)
        A(2,3,IO) = A(2,3,IO) + 2.*(DSM+DSP)
        B(1,3,IP) = B(1,3,IP) - 3.*(DSM/DSP)
        B(2,3,IP) = B(2,3,IP) + DSM
C
  2   CONTINUE
C
      IO = IIS
      IM = IIS - 1
      DSM = SS(IO) - SS(IM)
C
      B(3,1,IO) = B(3,1,IO) + 3.
      B(3,2,IO) = B(3,2,IO) + DSM
      A(3,1,IO) = A(3,1,IO) - 3.
      A(3,2,IO) = A(3,2,IO) + 2.*DSM
C
      C(1,3,IM) = C(1,3,IM) + 3.
      C(2,3,IM) = C(2,3,IM) + DSM
      A(1,3,IO) = A(1,3,IO) - 3.
      A(2,3,IO) = A(2,3,IO) + 2.*DSM
C
      IO = 1
C
      DO 3 ID = 1, IID
C
  32    IP = IO+1
        IF(SD(ID).LT.SS(IP) .OR. IP.EQ.IIS) GOTO 31
        IO = IP
        GOTO 32
C
  31    DS = SS(IP) - SS(IO)
        T = (SD(ID)-SS(IO)) / DS
C
        FP = 3.*T*T - 2.*T*T*T
        FM = 1. - 3*T*T + 2.*T*T*T
        FPP = DS * ( -T*T + T*T*T )
        FPM = DS * ( T - 2.*T*T + T*T*T )
C
        A(1,1,IO) = A(1,1,IO) + FM*FM
        A(1,2,IO) = A(1,2,IO) + FM*FPM
        C(1,1,IO) = C(1,1,IO) + FM*FP
        C(1,2,IO) = C(1,2,IO) + FM*FPP
        D(1,  IO) = D(1,  IO) + FM*XD(ID)
C
        A(2,1,IO) = A(2,1,IO) + FPM*FM
```

```
        A(2,2,IO) = A(2,2,IO) + FPM*FPM
        C(2,1,IO) = C(2,1,IO) + FPM*FP
        C(2,2,IO) = C(2,2,IO) + FPM*FPP
        D(2,  IO) = D(2,  IO) + FPM*XD(ID)
C
        B(1,1,IP) = B(1,1,IP) + FP*FM
        B(1,2,IP) = B(1,2,IP) + FP*FPM
        A(1,1,IP) = A(1,1,IP) + FP*FP
        A(1,2,IP) = A(1,2,IP) + FP*FPP
        D(1,  IP) = D(1,  IP) + FP*XD(ID)
C
        B(2,1,IP) = B(2,1,IP) + FPP*FM
        B(2,2,IP) = B(2,2,IP) + FPP*FPM
        A(2,1,IP) = A(2,1,IP) + FPP*FP
        A(2,2,IP) = A(2,2,IP) + FPP*FPP
        D(2,  IP) = D(2,  IP) + FPP*XD(ID)
C
    3 CONTINUE
C
      CALL SOLV2(A,B,C,D,IIS)
C
      DO 4 IO = 1, IIS
        XS(IO)  = D(1,IO)
        XPS(IO) = D(2,IO)
    4 CONTINUE
C
      RETURN
      END ; SPLIN2



      SUBROUTINE SOLV2(A,B,C,D,II)
C
      DIMENSION A(3,3,1),B(3,3,1),C(3,3,1),D(3,1),AI(3,3)
C
      DO 1 I = 1, II
C
        DETA = A(1,1,I)*( A(2,2,I)*A(3,3,I) - A(2,3,I)*A(3,2,I) )
     &       + A(1,2,I)*( A(2,3,I)*A(3,1,I) - A(2,1,I)*A(3,3,I) )
     &       + A(1,3,I)*( A(2,1,I)*A(3,2,I) - A(2,2,I)*A(3,1,I) )
C
        AI(1,1) = ( A(2,2,I)*A(3,3,I) - A(2,3,I)*A(3,2,I) ) / DETA
        AI(2,1) = ( A(2,3,I)*A(3,1,I) - A(2,1,I)*A(3,3,I) ) / DETA
        AI(3,1) = ( A(2,1,I)*A(3,2,I) - A(2,2,I)*A(3,1,I) ) / DETA
C
        AI(1,2) = ( A(3,2,I)*A(1,3,I) - A(3,3,I)*A(1,2,I) ) / DETA
        AI(2,2) = ( A(3,3,I)*A(1,1,I) - A(3,1,I)*A(1,3,I) ) / DETA
        AI(3,2) = ( A(3,1,I)*A(1,2,I) - A(3,2,I)*A(1,1,I) ) / DETA
C
```

```
      AI(1,3) = ( A(1,2,I)*A(2,3,I) - A(1,3,I)*A(2,2,I) ) / DETA
      AI(2,3) = ( A(1,3,I)*A(2,1,I) - A(1,1,I)*A(2,3,I) ) / DETA
      AI(3,3) = ( A(1,1,I)*A(2,2,I) - A(1,2,I)*A(2,1,I) ) / DETA
C
      E1 = AI(1,1)*D(1,I) + AI(1,2)*D(2,I) + AI(1,3)*D(3,I)
      E2 = AI(2,1)*D(1,I) + AI(2,2)*D(2,I) + AI(2,3)*D(3,I)
      E3 = AI(3,1)*D(1,I) + AI(3,2)*D(2,I) + AI(3,3)*D(3,I)
      D(1,I) = E1
      D(2,I) = E2
      D(3,I) = E3
C
      IF(I.EQ.II) GOTO 1
C
      DO 2 K = 1, 3
        E1 = AI(1,1)*C(1,K,I) + AI(1,2)*C(2,K,I) + AI(1,3)*C(3,K,I)
        E2 = AI(2,1)*C(1,K,I) + AI(2,2)*C(2,K,I) + AI(2,3)*C(3,K,I)
        E3 = AI(3,1)*C(1,K,I) + AI(3,2)*C(2,K,I) + AI(3,3)*C(3,K,I)
        C(1,K,I) = E1
        C(2,K,I) = E2
        C(3,K,I) = E3
  2   CONTINUE
C
      IP = I+1
      DO 3 J = 1, 3
        DO 31 L = 1, 3
          D(J,IP) = D(J,IP) - B(J,L,IP) * D(L,I)
          DO 311 K = 1, 3
            A(J,K,IP) = A(J,K,IP) - B(J,L,IP)*C(L,K,I)
 311      CONTINUE
  31    CONTINUE
   3  CONTINUE
C
   1  CONTINUE
C
      DO 4 I = II-1, 1, -1
        IP = I+1
        DO 41 J = 1, 3
          DO 411 L = 1, 3
            D(J,I) = D(J,I) - C(J,L,I) * D(L,IP)
 411      CONTINUE
  41    CONTINUE
   4  CONTINUE
C
      RETURN
      END ; SOLV2
```

```
C
      SUBROUTINE SPLINE(X,XP,S,II)
C
      DIMENSION X(300),XP(300),S(300)
      DIMENSION A(300),B(300),C(300),D(300)
C
      DO 1 I = 2, II-1
C
        IO = I
        IM = I - 1
        IP = I + 1
        DSM = S(IO) - S(IM)
        DSP = S(IP) - S(IO)
C
        B(IO) = DSP
        A(IO) = 2. * ( DSM + DSP )
        C(IO) = DSM
        D(IO) = 3.*( DSM*(X(IP)-X(IO))/DSP + DSP*(X(IO)-X(IM))/DSM )
C
  1   CONTINUE
C
      A(1) = 2.
      C(1) = 1.
      D(1) = 3. * (X(2)-X(1)) / (S(2)-S(1))
      B(II) = 1.
      A(II) = 2.
      D(II) = 3. * (X(II)-X(II-1)) / (S(II)-S(II-1))
C
      CALL SOLV(A,B,C,D,II)
C
      DO 2 I = 1, II
        XP(I) = D(I)
  2   CONTINUE
C
      RETURN
      END ; SPLINE



      SUBROUTINE SOLV(A,B,C,D,KK)
C
      DIMENSION A(1),B(1),C(1),D(1)
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 ; SOLVE



      FUNCTION SEVAL(SS,X,XP,S,N)
      REAL X(1), XP(1), S(1)
C
      DO 10 I=2, N
        IF(SS.LT.S(I)) GO TO 11
   10 CONTINUE
      I = N
C
   11 DS = S(I) - S(I-1)
      T = (SS-S(I-1)) / DS
      CX1 = DS*XP(I-1) - X(I) + X(I-1)
      CX2 = DS*XP(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,XP,S,N)
      REAL X(1), XP(1), S(1)
C
      DO 10 I=2, N
        IF(SS.LT.S(I)) GO TO 11
   10 CONTINUE
      I = N
C
   11 DS = S(I) - S(I-1)
      T = (SS-S(I-1)) / DS
      CX1 = DS*XP(I-1) - X(I) + X(I-1)
      CX2 = DS*XP(I)   - X(I) + X(I-1)
      DEVAL = X(I) - X(I-1) + (1.-4.*T+3.*T*T)*CX1 + T*(3.*T-2.)*CX2
      DEVAL = DEVAL / DS
C
      RETURN
      END ; DEVAL
```

```
      FUNCTION CURV(SS,X,XP,Y,YP,S,N)
      REAL X(1), XP(1), Y(1), YP(1), S(1)
C
      DO 10 I=2, N
        IF(SS.LT.S(I)) GO TO 11
   10 CONTINUE
      I = N
C
   11 DS = S(I) - S(I-1)
      T = (SS-S(I-1)) / DS
C
      CX1 = DS*XP(I-1) - X(I) + X(I-1)
      CX2 = DS*XP(I)   - X(I) + X(I-1)
      XS = X(I) - X(I-1) + (1.-4.*T+3.*T*T)*CX1 + T*(3.*T-2.)*CX2
      XSS = (6.*T-4.)*CX1 + (6.*T-2.0)*CX2
C
      CY1 = DS*YP(I-1) - Y(I) + Y(I-1)
      CY2 = DS*YP(I)   - Y(I) + Y(I-1)
      YS = Y(I) - Y(I-1) + (1.-4.*T+3.*T*T)*CY1 + T*(3.*T-2.)*CY2
      YSS = (6.*T-4.)*CY1 + (6.*T-2.0)*CY2
C
      CURV = (XS*YSS - YS*XSS) / (DS*(XS**2 + YS**2))
C
      RETURN
      END ; CURV
```

```
C
      SUBROUTINE SMOVE
$INCLUDE STATE.INC
$INCLUDE ISES.INC
      REAL DXI(IX)
      REAL XT(4),YT(4),KT(4,4)
C
      DO 99 I=1, II-1
        DSI = SQRT((X(I+1,1)-X(I,1))**2 + (Y(I+1,1)-Y(I,1))**2)
        DXI(I) = SQRT(SQRT(DSI))
   99 CONTINUE
C
      DO 1 I=1, II
        DO 2 J=1, JJ
          A1(J,I) = 0.
          A2(J,I) = 0.
```

```
          A3(J,I) = 0.
          B1(J,I) = 0.
          B2(J,I) = 0.
          B3(J,I) = 0.
          C1(J,I) = 0.
          C2(J,I) = 0.
          C3(J,I) = 0.
          DR(J,1,I) = 0.
          DR(J,2,I) = 0.
  2     CONTINUE
  1   CONTINUE
C
C------ Assemble stiffness matrices
C
      DO 3 JO = 1, JJ-1
        IF(JO.EQ.JJJ) GOTO 3
        JP = JO + 1
C
        DO 4 IO = 1, II-1
          IP = IO + 1
C
          XT(1) = X(IO,JO)
          XT(2) = X(IP,JO)
          XT(3) = X(IP,JP)
          XT(4) = X(IO,JP)
          YT(1) = Y(IO,JO)
          YT(2) = Y(IP,JO)
          YT(3) = Y(IP,JP)
          YT(4) = Y(IO,JP)
C
          CALL STIFF(XT,YT,KT)
C
          DXIO = DXI(IO)
C
          DR(JO,1,IO) = DR(JO,1,IO) - DXIO*( KT(1,2) + KT(1,3) )
          A2(JO,IO) = A2(JO,IO) + KT(1,1)
          A3(JO,IO) = A3(JO,IO) + KT(1,4)
          C2(JO,IO) = C2(JO,IO) + KT(1,2)
          C3(JO,IO) = C3(JO,IO) + KT(1,3)
C
          DR(JO,1,IP) = DR(JO,1,IP) + DXIO*( KT(2,1) + KT(2,4) )
          B2(JO,IP) = B2(JO,IP) + KT(2,1)
          B3(JO,IP) = B3(JO,IP) + KT(2,4)
          A2(JO,IP) = A2(JO,IP) + KT(2,2)
          A3(JO,IP) = A3(JO,IP) + KT(2,3)
C
          DR(JP,1,IP) = DR(JP,1,IP) + DXIO*( KT(3,1) + KT(3,4) )
          B1(JP,IP) = B1(JP,IP) + KT(3,1)
          B2(JP,IP) = B2(JP,IP) + KT(3,4)
```

```
            A1(JP,IP) = A1(JP,IP) + KT(3,2)
            A2(JP,IP) = A2(JP,IP) + KT(3,3)
C
            DR(JP,1,IO) = DR(JP,1,IO) - DXIO*( KT(4,2) + KT(4,3) )
            A1(JP,IO) = A1(JP,IO) + KT(4,1)
            A2(JP,IO) = A2(JP,IO) + KT(4,4)
            C1(JP,IO) = C1(JP,IO) + KT(4,2)
            C2(JP,IO) = C2(JP,IO) + KT(4,3)
C
   4      CONTINUE
C
   3    CONTINUE
C
        CALL DSOLV
C
C---- Move nodes
C
        DSRMS = 0.
        DSMAX = 0.
C
        DO 7 J = 2, JJ-1
          X1 = X(1,J)
          Y1 = Y(1,J)
          X2 = X(2,J)
          Y2 = Y(2,J)
C
          DO 8 I = 2, II-1
            X3 = X(I+1,J)
            Y3 = Y(I+1,J)
            DSH = - DR(J,2,I)/(DXI(I)+DXI(I-1)+DR(J,2,I+1)-DR(J,2,I-1))
            X(I,J) = X(I,J) + DSH * (X3-X1)
            Y(I,J) = Y(I,J) + DSH * (Y3-Y1)
CCC         DSRMS = DSRMS + DR(J,2,I)**2
CCC         DSMAX = AMAX1(DSMAX,ABS(DR(J,2,I)))
            X1 = X2
            Y1 = Y2
            X2 = X3
            Y2 = Y3
  8       CONTINUE
C
  7     CONTINUE
CCC     DSRMS = SQRT(DSRMS/FLOAT((II-2)*(JJ-2)))
CCC     WRITE(5,*) ' '
CCC     WRITE(5,*) 'SMOVE:    rms(ds) = ',DSRMS,'     max(ds) = ',DSMAX
C
        RETURN
        END ; SMOVE
```

```
      SUBROUTINE STIFF(X,Y,K)
C
C-----------------------------------------------------------------------
C
C     THIS PROGRAM CALCULATES THE STIFFNESS MATRIX FOR SOLVING LAPLACE'S
C     EQUATION USING A FOUR NODE ISOPARAMETRIC ELEMENT.
C
C     WRITTEN BY KENNETH C. HALL
C     COPYRIGHT JULY, 1984
C
C-----------------------------------------------------------------------
C
      REAL K(4,4),X(4),Y(4),JAC(2,2),PPRIME(2,4), WT(2),ETA(2),KSI(2)
      REAL XY(4,2),JINV(2,2),TEMP2(2,4),TEMP3(4,4)
C
      DATA ETA / -.57735, 0.57735 /
      DATA KSI / -.57735, 0.57735 /
      DATA WT  / 1.00000, 1.00000 /
C
C---- SET INITIAL VALUE OF STIFFNESS MATRIX TO ZERO
C
      DO 10 I=1,4
        DO 5 J=1,4
          K(I,J) = 0.0
 5      CONTINUE
 10   CONTINUE
C
      DO 20 I=1,4
        XY(I,1) = X(I)
        XY(I,2) = Y(I)
 20   CONTINUE
C
C---- DO SUMMATION OVER THE 2 X 2 GAUSSIAN STATIONS
C
      DO 210 I=1,2
C
C------ SET VALUES OF PPRIME, THE INTERPOLATION FUNCTION
C
        PPRIME(2,1) = -0.25*(1. - KSI(I))
        PPRIME(2,2) = -0.25*(1. + KSI(I))
        PPRIME(2,3) = +0.25*(1. + KSI(I))
        PPRIME(2,4) = +0.25*(1. - KSI(I))
C
        DO 200 J=1,2
          PPRIME(1,1) = -0.25*(1. - ETA(J))
          PPRIME(1,2) = +0.25*(1. - ETA(J))
          PPRIME(1,3) = +0.25*(1. + ETA(J))
```

```
          PPRIME(1,4) = -0.25*(1. + ETA(J))
C
C-------- COMPUTE JACOBIAN MATRIX
C
          DO 50 II=1,2
            DO 45 JJ=1,2
              SUM = 0.0
              DO 43 KK=1,4
                SUM = SUM + PPRIME(II,KK)*XY(KK,JJ)
 43           CONTINUE
              JAC(II,JJ) = SUM
 45         CONTINUE
 50       CONTINUE
C
C-------- COMPUTE INVERSE OF JACOBIAN
C
          DET = JAC(1,1)*JAC(2,2) - JAC(1,2)*JAC(2,1)
          DETINV = 1./DET
          JINV(1,1) = JAC(2,2)*DETINV
          JINV(1,2) = -JAC(1,2)*DETINV
          JINV(2,1) = -JAC(2,1)*DETINV
          JINV(2,2) = JAC(1,1)*DETINV
C
C-------- FIND JINV*PPRIME
C
          DO 70 II=1,2
            DO 60 JJ=1,4
              SUM = 0.0
              DO 55 KK=1,2
                SUM = SUM + JINV(II,KK)*PPRIME(KK,JJ)
 55           CONTINUE
              TEMP2(II,JJ) = SUM
 60         CONTINUE
 70       CONTINUE
C
          DO 90 II=1,4
            DO 80 JJ=1,4
              SUM = 0.0
              DO 75 KK=1,2
                SUM = SUM + TEMP2(KK,II)*TEMP2(KK,JJ)
 75           CONTINUE
              TEMP3(II,JJ) = SUM
 80         CONTINUE
 90       CONTINUE
C
          DO 30 II=1,4
            DO 25 JJ=1,4
              K(II,JJ) = K(II,JJ) + WT(I)*WT(J)*DET*TEMP3(II,JJ)
 25         CONTINUE
```

```
 30       CONTINUE
C
 200    CONTINUE
 210  CONTINUE
C
      RETURN
      END ; STIFF


      SUBROUTINE DSOLV
$INCLUDE STATE.INC
$INCLUDE ISES.INC
C
      DO 1 KOUNT=1, 5
C
        DSRMS = 0.
        DSMAX = 0.
        DO 11 IO=2, II-1
          IM = IO-1
          IP = IO+1
          CC(1) = 0.
          DD(1) = 0.
          DO 111 JO=2, JJ-1
            JM = JO-1
            JP = JO+1
            BB(JO) = A1(JM,IO)
            AA(JO) = A2(JO,IO)
            CC(JO) = A3(JP,IO)
            DD(JO) = DR(JO,1,IO)
     &               - B1(JO,IO)*DR(JM,2,IM)
     &               - B2(JO,IO)*DR(JO,2,IM)
     &               - B3(JO,IO)*DR(JP,2,IM)
     &               - A1(JO,IO)*DR(JM,2,IO)
     &               - A2(JO,IO)*DR(JO,2,IO)
     &               - A3(JO,IO)*DR(JP,2,IO)
     &               - C1(JO,IO)*DR(JM,2,IP)
     &               - C2(JO,IO)*DR(JO,2,IP)
     &               - C3(JO,IO)*DR(JP,2,IP)
            AA(JO) = AA(JO) - BB(JO)*CC(JM)
            DD(JO) = DD(JO) - BB(JO)*DD(JM)
            CC(JO) = CC(JO) / AA(JO)
            DD(JO) = DD(JO) / AA(JO)
 111      CONTINUE
C
          DD(JJ) = 0.
C
          DO 113 JO=JJ-1, 2, -1
            DD(JO) = DD(JO) - CC(JO)*DD(JO+1)
            DR(JO,2,IO) = DR(JO,2,IO) + DD(JO)
```

```
CCC         DSRMS = DSRMS + DD(JO)**2
CCC         DSMAX = AMAX1(DSMAX,ABS(DD(JO)))
 113      CONTINUE
C
  11    CONTINUE
C
        DO 12 JO=2, JJ-1
C
          JM = JO-1
          JP = JO+1
C
          CC(1) = 0.
          DD(1) = 0.
C
          DO 121 IO=2, II-1
            IM = IO-1
            IP = IO+1
            BB(IO) = B2(JO,IM)
            AA(IO) = A2(JO,IO)
            CC(IO) = C2(JO,IP)
            DD(IO) = DR(JO,1,IO)
     &             - B1(JO,IO)*DR(JM,2,IM)
     &             - B2(JO,IO)*DR(JO,2,IM)
     &             - B3(JO,IO)*DR(JP,2,IM)
     &             - A1(JO,IO)*DR(JM,2,IO)
     &             - A2(JO,IO)*DR(JO,2,IO)
     &             - A3(JO,IO)*DR(JP,2,IO)
     &             - C1(JO,IO)*DR(JM,2,IP)
     &             - C2(JO,IO)*DR(JO,2,IP)
     &             - C3(JO,IO)*DR(JP,2,IP)
            AA(IO) = AA(IO) - BB(IO)*CC(IM)
            DD(IO) = DD(IO) - BB(IO)*DD(IM)
            CC(IO) = CC(IO) / AA(IO)
            DD(IO) = DD(IO) / AA(IO)
 121      CONTINUE
C
          DD(II) = 0.
C
          DO 123 IO=II-1, 2, -1
            DD(IO) = DD(IO) - CC(IO)*DD(IO+1)
            DR(JO,2,IO) = DR(JO,2,IO) + DD(IO)
CCC         DSRMS = DSRMS + DD(JO)**2
CCC         DSMAX = AMAX1(DSMAX,ABS(DD(JO)))
 123      CONTINUE
C
  12    CONTINUE
C
CCC     DSRMS = SQRT(DSRMS/FLOAT(2*(II-2)*(JJ-2)))
CCC     WRITE(5,*) 'SMOVE:   rms(ds) = ',DSRMS,'    max(ds) = ',DSMAX
```

```
CCC      IF(DSMAX.LT.1.0E-4) RETURN
    1  CONTINUE
C
       RETURN
       END ; DSOLV


       SUBROUTINE DEKINK
$INCLUDE STATE.INC
$INCLUDE ISES.INC
       LOGICAL KINKED
C
       DO 1 J=2, JJ-1
         IF(J.EQ.JJJ .OR. J.EQ.JJJ+1) GO TO 1
         KINKED = .FALSE.
         XM = X(1,J)
         YM = Y(1,J)
         XO = X(2,J)
         YO = Y(2,J)
         DO 11 I=2, II-1
           XP = X(I+1,J)
           YP = Y(I+1,J)
           DOTP = (XO-XM)*(XP-XO) + (YO-YM)*(YP-YO)
           IF(DOTP.LT.0.0) THEN
            X(I,J) = 0.5*(XP+XM)
            Y(I,J) = 0.5*(YP+YM)
            KINKED = .TRUE.
           ENDIF
           XM = XO
           YM = YO
           XO = XP
           YO = YP
    11   CONTINUE
         IF(KINKED) WRITE(5,*) J,'''th streamline dekinked'
     1 CONTINUE
         RETURN
         END ; DEKINK
```