EMBEDDED MESH SOLUTIONS OF THE EULER EQUATION USING A MULTIPLE-GRID METHOD

by

WILLIAM JAMES USAB JR.

B.S. in Aeronautical and Astronautical Engineering
Purdue University
(1978)

S.M. in Aeronautics and Astronautics
Massachusetts Institute of Technology
(1981)

SUBMITTED TO THE DEPARTMENT OF
AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS OF THE DEGREE OF

DOCTOR OF PHILOSOPHY IN
COMPUTATIONAL FLUID DYNAMICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 1983

Signature of Author_____
Department of Aeronautics and Astronautics
December 28, 1983

Certified by_____
Professor Earll M. Murman
Thesis Supervisor

Certified by_____
Professor Judson R. Baron
Thesis Committee Member

Certified by_____
Professor Wesley L. Harris Sr.
Thesis Committee Member

Accepted by_____
Professor Harold H. Wachman
Chairman, Departmental Graduate Committee

EMBEDDED MESH SOLUTIONS OF THE EULER EQUATION USING A MULTIPLE-GRID METHOD

by

WILLIAM JAMES USAB JR.

Submitted to the Department of
Aeronautics and Astronautics
in December, 1983 in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computational fluid Dynamics

ABSTRACT

A computational procedure using a multiple-grid method with embedded mesh regions is developed for solving the two dimensional Euler equations. A pointer system is used to define the general multiple grid structure, which may include one or more, single or multiple embedded mesh regions of the same grid topology as the global mesh. The solution algorithm, based on the Ni multiple-grid method, has been extended to embedded mesh structures with the formulation of proper global/embedded interface conditions. The present approach combines the fast convergence to steady-state of multiple-grid methods with the flexibility and efficiency of an embedded mesh structure in resolving important flow features. Results are presented for several two dimensional subsonic and transonic airfoils using embedded meshes to resolve flow details in the leading edge, trailing edge, and shock regions. The present method is shown to retain the global coarse mesh convergence rates while gaining the flow resolution in embedded regions of a correspondingly globally refined mesh. Through the use of embedded meshes the total storage and computational work is significantly reduced over that of a equivalent global refinement.

In addition to the development of a embedded mesh approach the basic multiple-grid algorithm has been studied and improved on the areas of boundary conditions and residual transfers. All boundary conditions have been implemented in characteristic form. For lifting airfoils a vortex far field boundary condition has been developed which models the far field flow as the superposition of a uniform freestream and a compressible point vortex whose strength is determined by the

calculated lift on the airfoil. With this far field formulation the far field boundary may be placed much closer to the airfoil than for solutions in which the traditional uniform free stream boundary condition is used. Several different residual transfer operator formulations have been studied. Proper formulation of the residual transfer operator has been shown to be very important for computations on highly stretched meshes. A transfer operator based on the distribution formula of the base solver is shown to give the best performance for highly stretched meshes.

Thesis Supervisor: Earll M. Murman
Title: Professor of Aeronautics and Astronautics

ACKNOWLEDGEMENTS

The successful completion of this thesis would not have been possible without the support and encouragement of a great many friends and colleagues. To all those who have helped me, I owe an enormous debt of gratitude. Foremost, I would like to thank my chairman and advisor, Professor Earll Murman, for his guidance, advice, and untiring enthusiasm throughout this project. He always seemed to know when to criticize and when to encourage. I would also like to express my appreciation to the other members of my committee, Professors Judson Baron and Wesley Harris, for their many helpful comments and discussions. In addition, I would like to thank Dr. Ron-Ho Ni for his interest and suggestions in extending his method.

I would like to express my appreciation to my friends at MIT, expecially my office mates, for the late night discussions, constructive criticism and unending encouragement.

Finally, I would like to thank Karen and my family for their support, understanding, and love through the years I've been in school.

This work was partially supported by the Air Force Office of Scientific Research under grant AFOSR-82-0136, Dr. J. Wilson technical monitor, and by the National Aeronautics and Space Aministration under grant NGT 22-009-901, Dr. Randolph Groves, technical monitor.

TABLE OF CONTENTS

LIST OF SYMBOLS

a	speed of sound
D( )	change in
d( )	change in
e	total internal energy per unit volume
F	x flux vector
G	y flux vector
H	total enthalpy
h	mesh spacing
J	Jacobian
K	non-dimensional constant in Sutherland's law
n	iteration; normal coordinate
p	pressure
Pr	Prandtl number
q	speed
R	x viscous flux vector
Re	Reynolds number
S	y viscous flux vector
s	tangential direction
T	temperature; transfer operator
t	time
U	conservation variables
u	x velocity component
V	area; volume
v	y velocity component

W	characteristic variable
x	spatial variable
y	spacial variable
α	angle of attack
γ	ratio of specific heats
η	computational coordinate
θ	angle from the x axis
λ	eigenvalue
μ	viscosity
ξ	computational coordinate
ρ	density
τ	shear stress
∂	partial derivative operator

Subscripts:

c	cell center condition; corrected condition
FS	free stream condition
i	node index
o	reference conditions
P	predicted condition
v	vortex far field condition
w	wall

Superscripts:

l	direction
m	direction

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The field of computational fluid dynamics has evolved over the last two decades from the first attempts at solving model fluid flows to the stage where computational methods are playing an important role in aerodynamic design. The latest generation of aircraft are the first to have a significant amount of design done with computational methods [1-4]. This rapid evolution of computational methods has been driven by the need for faster more accurate design tools and the increasing cost of experimental design. In addition to cost, experimental testing is time consuming, and is limited in the flight regimes which can be tested and quantities which can be measured. Computational design tools on the other hand are becoming faster, less expensive, and more accurate due to the rapid development of numerical methods and increasing performance of computers. These tools have allowed the study of a much wider range of designs. They can predict information in regions which often can't physically be measured and without the interference of walls, probes, and other apparatus.

The ultimate goal of computational fluid dynamics for transonic flows is the calculation of a complete aircraft configuration including wings, body, engine nacelles, and any external stores. Such a flow involves both complexity in respect to the range of fluid mechanic features of the flow and the geometric complexity of the problem. The flow includes subsonic regions, supersonic regions, shocks, regions where the flow is essentially inviscid, and others where viscous effects dominate. While all these aspects of the flow are described by the full Navier-Stokes equations, a solution of these equations for a complete aircraft configuration is impossible at the present time and will most likely remain so in the near future. This conclusion is based on the performance of present Navier-Stokes solvers and a simple estimate of the computer resources required, which vastly exceed any available today.

Rather than solve the full Navier-Stokes equations the approach taken has been to consider a series of simplifying approximations resulting in a model equation set which is much simpler to solve. Based on the observations first made by Prandtl that for high Reynolds number flows the effects of viscosity are confined to thin layers near the surface of the body, a majority of the flow may be considered essentially inviscid. While the thin viscous shear layers in the flow are important in determining the location of separation and vorticity generated in the flow, the assumption of inviscid flow described by the Euler Equations is a good approximation

for a majority of the flow. Although this step simplifies the governing equations substantially, the Euler equations are still difficult to solve. Observing that for external flows, in addition to being inviscid, much of the flow is also irrotational then leads to the next lower approximation to the flow described by the full potential equation. Note, like the first approximation, important information about the flow is lost but the resulting model equation is much easier to solve. In particular since the flow is irrotational there is no way to generate entropy through shocks. This limits the flow range for which the model equation may be considered a good approximation. An even lower approximation is to assume the body is thin and to limit the flow range even further to a region near Mach 1. Under these conditions the small disturbance approximation can be made resulting in the small disturbance form of the potential equation.

Table 1-1 summarizes the levels of approximation and resulting model equations for transonic flows. The level of difficulty in solving these equations increases from bottom to top in the table. Therefore it comes as no surprise that the level of development of algorithms for solving these flows is most advanced at the bottom and also decreases as one moves up the table. The development process for any new algorithm can be viewed in terms of three stages. In the first stage the concentration is on formulation of the algorithm with the algorithm being tested for simple model problems. The second stage is the validation of the code for two dimensional flows

(such as airfoils, ducts, etc.) and development of the three dimensional extension. Finally, if the algorithm is successful, it is incorporated into the design process and an ongoing process of extending the solver to increasingly complex geometries begins. It is clear then that the level of development of a solver then determines the level of complexity of the flow geometries which can be solved. Therefore the level of geometric complexity of the problems which can be solved decreases with the level of approximation of the model equation.

Table 1-1: Summary of Current Transonic Solver Development

MODEL EQUATION	APPROXIMATION MADE	LEVEL OF DEVELOPMENT	COMPLEXITY OF SOLUTIONS
Navier-Stokes Equations		Low	Simple 2-D flows
Euler	Inviscid	|	2-D and Simple 3-D
Full Potential	Inviscid Irrotational	|	2-D and 3-D flows
Small Disturbance Equation	Above Plus M near 1 Thin bodies	High	Complex 3-D flows

At this point consider the current level of development of algorithms for solving the model equations of table 1-1. The first sucessful transonic calculations were made with the solution of the transonic small disturbance equation for flow about two dimensional airfoils by Murman and Cole [5]. Their algorithm has served as the foundation for the many 2-D, axisymmetric, and 3-D small disturbance potential solvers in use today. Of particular importance was the combination of

multi-grid methods by South and Brandt [6] and approximate factorization methods by Ballhaus, Jameson and Albert [7] with a small disturbance solver to obtain accelerated convergence for 2-D airfoil calculations. The extension of this algorithm to complicated 3-D aircraft configurations (including body, wings, and nacelle) has been made by Boppe [8,9]. Boppe's work represents a landmark in the calculation of complicated geometries. At the present time methods for solving the small disturbance equation are highly developed and well integrated into the design process.

With the progress and experience gained from development of small disturbance potential solvers the concentration shifted to solution of the full potential equations. Fast and efficient finite volume methods such as those of Jameson [10] and Caspar, Hobbs and Davis [11] have been developed and applied to a wide range of 2-D flow problems. These methods, which give good results for a much wider range of flows and geometries, have replaced many of the small disturbance solvers as design tools. In addition these methods have been extended to 3-D flows with a great deal of success. The solution of simple aircraft configurations (wing/body and wing/body/tail geometries) have been demonstrated by Jameson and Caughy [12]. The limiting factor in extending these methods to more complicated and realistic configurations appears to be the difficulty in generating global body-fitted grid systems required by current solvers. Atta and Vadyak [13] have taken a new and promising approach to this problem

with wing/body/nacelle configurations by patching a cylindrical mesh which fits the nacelle into a global wing/body mesh.

The Euler equations are a much more difficult class of equations to solve numerically than either of the two previous classes. While both implicit and explicit algorithms for solving these equations have existed for some time, the amount of computional work required due to the poor convergence of these methods has made them unacceptable for design applications. Only very recently have new efficient algorithms been developed which show a great deal of promise. Of these, Ni's method [14] using a conservative Lax-Wendroff scheme combined with a multiple-grid scheme has shown greatly accelerated convergence rates. A second scheme using a conservative finite volume algorithm coupled with an explicit Runge-Kutta time stepping scheme has been presented by Jameson, Schmidt and Turkel [15]. This method has been applied to flow past lifting airfoils and extended to 3-D wing calculations by Jameson, Schmidt and Whitfield [16]. Rizzi [17] has also used this scheme for wing and wing/body configurations. Even more recently Jameson [18] formulated a multiple-grid acceleration technique to further accelerate convergence, demonstrating this scheme for 2-D airfoil solutions. With the development of these new algorithms it is now possible to obtain Euler solutions to flow problems with the same order of computational work as required by potential solvers. Currently these new solvers are being integrated

into the design process.

To summarize, the level of development of algorithms for solving the model equations of table 1-1 decreases as one moves to better approximations to the flow. The geometric complexity of the flow which can be solved also decreases as the level of approximation increases. The major stumbling block with each of these methods, as one proceeds to increasingly complex geometries, has not been with the solvers but the problem of generating properly distributed mesh systems. Most of these methods require a continuous body-fitted mesh which covers the entire flow domain. A coordinate transformation is then used to map this domain into a single rectangular box. This approach works quite well for simple geometries such as 2-D problems with singly connected domains (airfoils, cylinders, ducts) or individual 3-D aircraft components (such as the body, wing, nacelle). Even though wing/body meshes have been generated by Eriksson [19] and wing/body/tail meshes by Jameson and Baker [20], the extension of this approach to complete aircraft configurations would be extremely difficult, if at all possible. In addition these mesh generation schemes often introduce singularities in the grid which must be handled by the solver. These grid structures often result in poor grid resolution in critical areas and large numbers of unnecessary mesh points in others. One obvious cause for these problems is the requirement of body-fitted meshes. An alternative adopted by Boppe [8,9] and others to simplify the grid generation problem is to use

nonbody-fitted grids, but this results in extremely complicated boundary condition formulations.

The problem of grid generation is only amplified as one moves from the small disturbance equations to higher model equations. With the higher approximations, more and more flow detail can accurately be modeled but to take advantage of this requires better grid resolution in these feature areas. To gain this resolution requires better grid control. In addition, to compensate for the higher resolution in critical areas, it is important to minimize the number of unnecessary points in the grid structure if the overall computation times are to remain realistic.

Even if the present problems associated with global grid generation for complex geometries can be overcome, one must question whether this approach is leading to more universal and easily adaptable codes. At the present time this approach is creating increasingly complex and specialized codes. Each new geometry or higher approximation in the governing equations results in a new and more difficult grid generation problem. Once the grid generation problem is overcome the code must be rewritten to operate on this grid.

An alternate approach to problems of increasing geometric complexity is to view the solution domain as the sum of simple subdomains rather than one continuous global domain. In this view each subdomain is defined by some characteristic geometric feature. For example a complete aircraft

configuration might be viewed in terms of a body region, wing region, nacelle region and so on. The global solution is then obtained by using a global solution scheme which couples and provides interaction between individual solutions of each subdomain. Such a component structure is more easily adaptable to different geometries with increasing complexity. Currently these approaches have taken one of the following two forms; patching methods and multi-grid methods.

Patching methods involve dividing the domain into any number of subdomains where the subdomains either butt against each other or overlap each other such that the sum covers the global solution domain. A simple body conforming or body resolving refined mesh is then defined for each subdomain. The global solution is found by cycling the solver between these subdomains with proper boundary conditions defined on subdomain boundaries. A landmark in the calculation of complicated geometries has been set by Boppe [8,9] in solving the small disturbance potential equation for full aircraft configurations. Boppe's approach was the use a coarse global Cartesian grid overlapping locally refined Cartesian grid subdomains for each of the aircraft components (body, wing, nacelle). He incorporated Dirichlet type boundary conditions on the overlapping boundaries. The solution is then obtained by iterating between solving for the solution on the coarse global grid and each of the subdomains with boundary conditions being interpolated from adjoining subdomains. This work stands as proof that solutions to full aircraft

configurations are attainable with the computational resources available today. The actual method however is limited in value since these are small disturbance solutions, a poor approximation for realistic aircraft, and also since good results appear to be strongly user dependent. A similar approach for solving the full potential equation in overlapping mesh structures has been studied by Atta and Vadyak [13,21]. Atta began by considering solutions for transonic airfoil where a coarse global Cartesian mesh is overlapped with a local body-fitted O-type mesh. Dirichlet type boundary conditions were used on the outer boundary of the O-type mesh, while a Neumann type condition was used on the inner boundary of the global Cartesian mesh. With this model formulation he then studied the effect of variations in the two grid domains, overlap size, and cycling process on the accuracy and convergence of the solution as compared with the standard global calculation. He found that equivalent accuracies are possible with a savings in computation time with a proper grid sizing. Atta and Vadyak [13] then applied this approach to the calculation of a wing/body/nacelle aircraft configuration using a body-fitted cylindrical mesh around the nacelle which overlapped with a global wing/body mesh. One of the critical problems encountered in this extension was the complexity of three dimensional interpolations between the two mesh systems. While very preliminary in nature the results suggest that this is a promising approach to complex configurations. Finally, Forester [22] employed an overlapping grid system for

calculation of subsonic potential flow in a lobed mixer nozzle of a jet engine.

A second form of coupling of subdomains is through the use of multi-grid methods. Multi-grid methods were originally developed by Brandt [23,24] as a very fast and efficient way of solving elliptic type equations. The basic concept of multi-grid is to discretize the governing equations on a series of increasingly coarser meshes and then to systematically cycle through these meshes using a relaxation scheme to simultaneously liquidate errors of all wavelengths contained in the solution. In addition to the acceleration of convergence, Brandt suggests that the multi-grid structure provides the perfect framework for embedding areas of local mesh refinement. In this manner it is then possible to create any number of local subdomains of the same grid topology as the global grids with the multi-grid algorithm providing the coupling between the subdomains and the global mesh. This approach has the advantage of actually coupling the the entire solutions rather than relying solely on boundary conditions to couple the solution, as is done with patching methods. Unfortunately there has been no published demonstration of this approach by Brandt. The method has been implemented by Brown [25] for the solution of the transonic potential equation. Brown used a local embedded mesh refinement in the leading edge region of a isolated nacelle to resolve the local flow detail. With this approach he showed that there was a great savings in computational work over the equivalent global

mesh refinement.

From the preceeding review it is clear that the development of transonic flow solvers may be characterized as a step by step progression of higher approximations to the actual flow. Beginning with the solution of simplified model equations for simple geometries the development has steadily moved toward better approximations of both the governing equations and more realistic geometries. The point has now been reached where current algorithms are capable of solving complicated realistic aircraft configurations. Unfortunately the complexity of the solvers, and in particular the mesh generators, have grown in proportion to the problem complexity. Currently, small changes in geometry require a tremendous amount of code development. Rather than continue the development of solvers on this case by case basis what is needed is a more general approach to solving flow problems which is easily adaptable to changing geometries and flow features. Another perspective in analyzing complex flows is to view the flow as composed of a number of regions or subdomains distinguished by fluid dynamic or geometric features. To accurately solve for a fluid dynamic feature (such as inviscid regions, viscous shear layers, shocks, etc.) requires modeling the dominant fluid process with the correct governing equations in that region. In addition, associated with both fluid dynamic features and geometric features is a characteristic structure and length scale which must be resolved through proper definition of the grid structure in

the subdomain. Recognizing that a general flow may be decomposed into components leads to the idea of a general modular approach to solving flow problems. With a general modular approach then the global domain is broken into any number of suitable subdomains. A simple grid structure is defined for each subdomain based on the features and associated scales to be resolved. A general solver is then used to simultaneously solve for the flow in each subdomain while coupling the subdomain solutions and allowing proper interaction. The patching and multi-grid formulations previously mentioned represent an initial step in this direction by using subdomains to handle complex geometries, but the present modular concept is intended to lead to a much broader and more systematic approach.

A general multiple-grid mesh structure provides an ideal framework for a modular approach to solving flow problems by providing a systematic way of assembling and coupling the subdomains together. The multiple-grid structure consists of a relatively crude global grid covering the entire solution domain, and any number of embedded local grids providing adequate resolution of local flow features. In this mesh structure the local subdomains are defined through creation of the embedded mesh regions where the resolution determined by the number of embedded levels. The solution of the discrete equations on the coarse global levels provide a very efficient way of coupling the embedded region solutions together since the coupling takes place over the entire domain rather than a

simple patching technique. This modular approach using a multiple-grid structure follows the general approach suggested by Brandt [23,24] for embedded solutions of elliptic type equations. In principle this approach can be carried much further by including a change of grid topology within the embedded mesh region. By changing the grid structure much better resolution of both fluid dynamic and geometric features within the subdomain would be possible. A second extension would be to allow a change in the level of approximation of the governing equations within a subdomain. This change in equation approximation would be beneficial since the finer mesh scale within the embedded region would be capable of resolving the feature scales associated with the higher approximation. Conceptually the multiple-grid structure provides a flexible framework for the development of a general modular approach to solving flow problems.

The objective of the present thesis is to begin the formulation of a general modular approach to solving complex flow problems. The first step in such a formulation is the development of a method for solving a single governing equation set on a general multiple-grid structure with one or more embedded mesh regions. To this point no mention has been made of the solver which will be used. It is clear that this choice will depend at least to some extent on the governing equation set to be solved. The Euler equations have been chosen as the appropriate governing equation set for the following reasons. First and foremost, the Euler equations

apply to a broad range of transonic flows, modeling inviscid flow features to a higher approximation than either the small disturbance or full potential equations. The higher approximation of flow features in turn makes the high resolution of an embedded mesh approach worth while. In addition, Euler solvers do not have the nonuniqueness problems associated with potential solvers [26,27]. Considering the current state of algorithm development for the various model equations of table 1-1, the computational efficiency of the recently developed Euler solvers are in the same range as small disturbance and potential solvers. These new Euler solvers will be replacing the current potential solvers as future design tools. Therefore an embedded multiple-grid approach for solving the Euler equations is in keeping with the current state of the art for flow solvers. A final consideration is the close relation between the Euler equations and the full Navier-Stokes equations. The Euler equations are a natural subset of the Navier-stokes equation in the limit of zero viscosity. With the proper solver formulation the viscous terms could be added in embedded regions where these terms are important, thus adding equation embedding to the general modular approach.

Of the new Euler solvers which have been recently presented, Ni's multiple-grid algorithm [14] was chosen for the present formulation. While this solver is not a true multi-grid method in the sense of Brandt's work, Ni's method is formulated around a global multiple-grid structure where

modified discrete equations are solved on the coarser mesh levels to accelerate solution convergence. This method will be extended to general embedded mesh structures which contain one or more embedded mesh regions. With this extension the solver now plays dual roles of accelerating the convergence of the solution and also providing the coupling mechanism between embedded and the global mesh solutions. The storage with this algorithm is kept to a minimum since the required solution information is only stored once for each mesh point on the finest level in each region.

In order to extend this scheme to completely general grid structures the solution algorithm must be separated from the grid structure. That is, the organization of the computational data base, comprised of the variables at node points, must not be determined by the solution algorithm. This has been accomplished through the development of a pointer system which defines the grid structure. The usual subscripted index notation (i,j) of finite difference procedures is replaced by a single numerical subscript to identify mesh points. The pointer system is very similar to the connectivity array which is used to define general finite element systems. Boundary conditions and their location, which also vary from problem to problem, must likewise be defined in this pointer system. With the grid-structure defined through a pointer system, a general solver may now be written in terms of these pointers. This separation of grid structure from the solver is the key to creation of a general

modular approach to problems.

While a majority of the present work is concerned with the solution of the Euler equations, Johnson and Chima [28,29] and Davis [30] have demonstrated that Ni's multiple-grid accelerator is easily extendible to the Reynolds averaged Navier-Stokes equation. The present author, in an unpublished pilot study, also drew the same conclusion. It is felt that the multiple-grid structure method given herein should prove to be an attractive algorithm for extension to embedded viscous regions where the Navier-Stokes would be solved.

The modular approach which has been developed has been applied to the solution transonic flow about 2-D airfoils. While these transonic flows are not geometrically complex they do contain important flow features such as shocks, stagnation points, leading edge detail and trailing edge detail which must be correctly resolved for an accurate solution. With conventional global solvers these features are solved through the use of grid packing. Unfortunately, when grid packing is used with a global mesh to resolve these features, packing also occurs in the far field regions resulting in a large number of unnecessary mesh points. The present modular approach with embedded mesh regions can be used to resolve the flow features while minimizing the total number of mesh points and therefore the computational work required. A second reason for choosing 2-D transonic airfoil flows to demonstrate the current method is that analytical solutions for some configurations are known. Using these cases, the accuracy of

both the global solver and the embedded mesh solver can be evaluated. Finally since the code development and testing was performed using a VAX 750 mini-computer it was necessary to choose a problem for which the solution computation times were reasonable.

In the chapters which follow, the governing equations will be defined followed by review of the basic Ni scheme. The conditions used at farfield and solid wall boundaries are described together with the Kutta condition. The extension to embedded mesh regions is then made through formulation of proper cell integrations at embedded mesh boundaries. Finally, with the general solver formulated, the pointer system which directs the solver is presented.

CHAPTER 2

GOVERNING EQUATIONS

2.1 NAVIER-STOKES EQUATIONS

The compressible form of the Navier-Stokes equations express the laws of conservation of mass, momentum and energy for viscous flows. When combined with an equation of state, such as the perfect gas law, the constant Prandtl number assumption and an expression relating viscosity to temperature there then results a complete set of equations for laminar flows at standard pressures and temperatures. The two-dimensional Navier-Stokes equations for unsteady compressible flow may be expressed in conservation form for a cartesian coordinate system as

$$U_t + F_x + G_y + R_x + S_y = 0 \qquad (2.1a)$$

where

$$U = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ e \end{vmatrix} \qquad F = \begin{vmatrix} \rho u \\ \rho uu + p \\ \rho uv \\ \rho uH \end{vmatrix} \qquad G = \begin{vmatrix} \rho v \\ \rho uv \\ \rho vv + p \\ \rho vH \end{vmatrix} \qquad (2.1b)$$

$$R = (1/Re_0)\begin{vmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xx}\,u + \tau_{yx}\,v - (\mu/((\gamma-1)Pr))\,T_x \end{vmatrix}$$

$$S = (1/Re_0)\begin{vmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yy}\,v + \tau_{xy}\,u - (\mu/((\gamma-1)Pr))\,T_y \end{vmatrix}$$

and where

$$\begin{aligned} \tau_{xx} &= -\mu[\,(4/3)u_x - (2/3)v_y\,] \\ \tau_{yy} &= -\mu[\,(4/3)v_y - (2/3)u_x\,] \\ \tau_{xy} &= \tau_{yx} = -\mu[\,u_y + v_x\,] \end{aligned} \qquad (2.1c)$$

in terms of density ρ, cartesian (x,y) velocity components (u,v), temperature T, total internal energy per unit volume e, viscosity coefficient μ, and Prandtl number Pr. The pressure p and total enthalpy H are then defined for a perfect gas as

$$\begin{aligned} p &= (\gamma - 1)[\,e - 0.5\,\rho(\,uu + vv\,)] \\ H &= (\,e + p\,)/\rho \end{aligned} \qquad (2.2)$$

where γ is the ratio of specific heats.

To complete the set of governing equations for laminar flows an expression relating the viscosity to temperature is required. Sutherlands law, an empirical relation describing the viscosity μ as a function of temperature T is given as

$$\mu = T^{1.5} \, (1 + K)/(T + K) \qquad (2.3)$$
$$K = (110 \, K)/T_o$$

where T_o is the reference temperature.

The above equations have been non-dimensionalized with respect to stagnation reference conditions ρ_o, a_o, T_o, and reference length l_o, resulting in the appearance of the reference Reynolds number, Re_o, in the above equations. A detailed description of this non-dimensionalization is presented on appendix A. Historically, the governing equations have been non-dimensionalized to identify the relevant non-dimensional parameters (such as Mach number, Reynolds number, Prandtl number, etc.) and to determine the relative order of magnitude of different terms in the equations. For numerical calculations, scaling of the equations performs two important functions. First, a proper choice of reference conditions scales all computational variables to similar order which reduces computational truncation errors. Secondly, it eliminates concern over carrying a consistent set of dimensions throughout the code and reduces errors in definition of input by the user. The present scaling based on stagnation reference conditions is well suited for transonic flows where a majority of the flow is near M = 1. It is also interesting to note that with this scaling the Euler equation form is the same as the unscaled equations.

The governing equations (2.1) are presented in strong conservation law form (SCLF) in terms of conservation variables U, as opposed to the non-conservation form (NCF) which would be expressed in terms of primitive variables (ρ, u, v, and p). As shown by Hindman [31], the choice of which form to use requires consideration of the types of flows to be solved and also has an impact on the way the equations should be discretized. In particular the SCLF form of the equations properly captures the weak shock solution with the correct shock jump conditions while the NCF does not. Since the present work is concerned with transonic flows with shocks, where these shocks are resolved through shock capturing rather than shock fitting, the SCLF is important. While the governing equations of (2.1) are written for a Cartesian coordinate system this SCLF can be preserved upon transformation to a general nonorthogonal coordinate system as shown by Viviand [32] and Warsi [33]. Consider the following transformation from cartesian system (x,y) to nonorthogonal system (ξ,η) defined as

$$\xi = \xi(x,y) \qquad \eta = \eta(x,y). \tag{2.3}$$

The SCLF form can be maintained as

$$(U/J)_t + [\, y_\eta F - x_\eta G \,]_\xi + [\, x_\xi G - y_\xi F \,]_\eta + [\, y_\eta R - x_\eta S \,]_\xi + [\, x_\xi S - y_\xi R \,]_\eta = 0 \tag{2.4}$$

where the transformation Jacobian J is

$$(1/J) = x_\xi y_\eta - x_\eta y_\xi \tag{2.5}$$

While this represents the SCLF for the differential equation of a general coordinate system, Hindman [31] and Thompkins, et al [34] point out that great care must be taken in discretizing this system for a given algorithm if the final scheme is to remain conservative. In particular, the transformation matrices (x_ξ, y_ξ, x_η, y_η) must be defined properly if the desired result is a conservative finite volume method. It is often helpful in constructing finite volume methods if, in addition to equation (2.4), the corresponding integral equation is considered. Integrating and applying the divergence theorem, the governing equations may be cast in integral form as

$$\frac{\partial}{\partial t}\iint_V U \, dA = \oint_{\partial V} (F,G).\vec{n} \, dS + \oint_{\partial V} (R,S).\vec{n} \, dS \qquad (2.6)$$

Approximation of this equation then leads to a finite volume method in conservation form.

For a majority of the viscous flows of interest the flow is not laminar but turbulent. The laminar Navier-Stokes equations can be extended to turbulent flows by modeling the Reynolds stress terms of the Reynolds averaged Navier-Stokes equations with an eddy viscosity model (such as Cebeci and Smith [35] or Baldwin and Lomax [36]). These terms are then included by replacing the laminar viscosity μ with a total effective viscosity μ_T, defined as the sum of the laminar viscosity and turbulent eddy viscosity ε_m. In addition the thermal conductivity μ/Pr in the energy equation is replaced with a total effective conductivity, the sum of laminar and turbulent parts as $\mu/Pr + \varepsilon_m/Pr_T$.

2.2 EULER EQUATIONS

For high Reynolds number flows commonly encountered in aircraft designs the viscous effects are often limited to very thin shear layers near the body. Under these conditions the assumption of inviscid flow is often a very good approximation to the flow. Eliminating the viscous terms in the governing equations (2.1) reduces them to the standard two dimensional Euler equations. Setting R=S=0 results in the follow equation for a cartesian system,

$$U_t + F_x + G_y = 0 \tag{2.8}$$

and for the general nonorthogonal system in (SCLF) form

$$(U/J)_t + [\, y_\eta F - x_\eta G \,]_\xi + [\, x_\xi G - y_\xi F \,]_\eta = 0 \tag{2.9}$$

Following eqn 2.6, the corresponding integral equation is

$$\frac{\partial}{\partial t}\iint_V U \, dA = \oint_{\partial V} (F,G).\vec{n} \, dS \tag{2.10}$$

which is useful in constructing a finite volume method in conservation form.

2.3 PHYSICAL BOUNDARY CONDITIONS

With the governing differential equations defined, it is now appropriate to define the physical boundary conditions which are required for solution of transonic airfoil problems. The term physical boundary condition is used here to describe the known flow conditions along the boundary of the domain to be solved. These boundary conditions should not be confused with the implementation of the boundary conditions in solving

the discrete equations which include not only the physical conditions but may also include numerical conditions required to close the system of discrete equations. The actual implementation of these conditions will be discussed later in chapter 3. Consider a typical airfoil in a transonic freestream as shown in figure 2-1. There are basically two types of boundaries for this problem, a solid wall boundary at the airfoil surface and the farfield boundary at infinity. For the solid wall boundary with viscous flow the physical boundary condition is zero velocity and either specified surface temperature or heat flux. Therefore,

$$u = 0 \quad v = 0$$

$$\left[T = T_w \quad \text{or} \quad T_n = (T_n)_w \right] \tag{2.11}$$

The corresponding boundary condition for inviscid flow is no flux through the surface,

$$(u,v).\vec{n} = 0 \tag{2.12}$$

The physical boundary conditions for the farfield boundary of figure 2-1 are uniform freestream flow at infinity,

$$u = (u)_{FS} \quad v = (v)_{FS} \quad p = (p)_{FS} \quad \rho = (\rho)_{FS} \tag{2.13}$$

This farfield boundary condition applies to both inviscid and viscous flows every where with exception of the wake region. In this region the pressure must be constant. Therefore,

$$p = (p)_{FS} \tag{2.14}$$

across the wake on the downstream boundary. Unfortunately for numerical calculations the farfield boundary is not placed at infinity but at some large finite distance R from the airfoil as shown by the dashed boundary in the figure. While the free stream conditions presented above are very commonly used for lifting airfoil problems, for accurate results the boundary must be placed a large distance from the airfoil. In practice, it is not uncommon to see a far field radius on the order of 100 chords [37]. This results form the fact that, while there is a net circulation around lifting airfoils, the farfield condition assumes zero circulation. If the flow is irrotational (both inviscid and shock free) with a subsonic freestream, a much better approximation is possible by viewing the farfield flow as the superposition of uniform flow and a compressible point vortex centered at the airfoil. With this formulation the outer boundary can be placed much closer to the airfoil.

The compressible potential for this flow has been derived by Ludford [38] as

$$\phi = q_\infty R \cos(\theta - \alpha) - (\Gamma/2\pi)\tan^{-1}[\beta \tan(\theta - \alpha)] \qquad (2.15a)$$

where

$$\beta = \sqrt{1 - M^2} \qquad (2.15b)$$

The circulation Γ is based on the lift coefficient found from a surface integration of the pressure around the airfoil

$$\Gamma = 0.5\, q_\infty\, c\, C_L \qquad (2.16)$$

Using the freestream conditions of (2.13) combined with the compressible Bernoulli equation and the above potential, the farfield flow conditions (denoted with subscript v) are obtained as

$$u_v = u_{FS} + \sin(\theta) q_\infty c C_L \beta / \{4\pi R [\cos^2(\theta-\alpha) + \beta^2 \sin^2(\theta-\alpha)]\} \quad (2.17)$$

$$v_v = v_{FS} - \cos(\theta) q_\infty c C_L \beta / \{4\pi R [\cos^2(\theta-\alpha) + \beta^2 \sin^2(\theta-\alpha)]\}$$

$$p_v = \{ p_{FS}^{\frac{\gamma-1}{\gamma}} + (\gamma-1)\rho_{FS} [q_\infty^2 - u_{FS}^2 - v_{FS}^2]/(2 p_{FS}^{1/\gamma}) \}^{\frac{\gamma}{\gamma-1}}$$

$$\rho_v = \rho_{FS} (p_v / p_{FS})^{1/\gamma}$$

It is clear that an assumption of uniform flow at the far field boundary is simply a lower order approximation of the above expressions.

The vortex far field boundary condition presented above was developed under the assumption of irrotational and therefore, shock free flow. This assumption allows one to equate the circulation around the airfoil surface directly to the circulation around the far field boundary. For transonic flows with shocks the net circulation around the far field boundary is not equal to the bound airfoil circulation due to the additional vorticity within the rotational wake region generated by the shock. Proper calculation of the far field circulation would require both a surface integration for the

bound circulation and a field integration of the wake vorticity.

A simplified model is developed here. The important assumption is that the flow is inviscid and at some distance behind the airfoil the pressure and the velocity direction return to freestream values. Consider the three contours shown in figure 2-2 (1 the airfoil surface contour, 2 an intermediate radius contour and 3 the finite far field radius contour). The following observations can be made. First, a momentum integration for the lift on the airfoil using each of the three contours results in the following result:

$$L_1 = L_2 = L_3 \tag{2.18}$$

Second, with the presence of the shock, the circulation corresponding to contour 1 can not be related to the lift on the airfoil using equation 2.16, since this expression assumes the flow is irrotational. In addition, with vorticity in the wake region, the circulation for each of the three contours can be different.

$$\Gamma_1 \neq \Gamma_2 \neq \Gamma_3 \tag{2.19}$$

If, however, contour 2 is chosen to be a sufficient distance from the airfoil to satisfy the stated assumptions, then the circulation for contour 2 will be the same as contour 3. This can be shown by performing an integration of the vorticity over the area between contours 2 and 3. Noting that the flow is irrotational everywhere outside the wake region then this

integral reduces to the integration over the portion of the wake between 2 and 3. After changing to a contour integration then

$$\Gamma_3 - \Gamma_2 = \oint_4 q.ds \qquad (2.20)$$

where contour 4 is defined as a path around the portion of the wake, as shown in figure 2-2. Evaluating this integral in segments, it is clear that since the flow conditions above and below the wake are at the same freestream flow conditions, the upper segment will cancel the lower. Since the flow is unidirectional, the contributions from the sides also cancel, and equation 2.20 reduces to the following,

$$\Gamma_2 = \Gamma_3 \qquad (2.21)$$

What this says is that although the wake is rotational, there is no net vorticity.

Now by viewing the flow field from a far field perspective the flow at (or outside) contour 2 may once again be represented as that of a compressible point vortex centered at the airfoil with circulation of sufficient strength to generate the lift determined by momentum integration using contour 2. Since the lift is independent of the contour chosen, the correct circulation is

$$\Gamma = \Gamma_2 = L_2 / \rho q_\infty = L_1 / \rho q_\infty = 0.5\, q_\infty c\, C_L \qquad (2.22)$$

Note that this is exactly the same expression for the vortex strength as that used for irrotational (shock free) flows.

Therefore, equation 2.17 is also a good approximation for the far field flow of transonic airfoils with shocks.

For inviscid flows the definition of the far field flow and body geometry alone are not sufficient to determine a unique solution for lifting airfoils. To make the flow solution unique requires also specifying the circulation about the body. The lift is then determined by this circulation. For lifting airfoils with sharp trailing edges the circulation is fixed by the Kutta condition. The Kutta condition states that a body with a sharp trailing edge in motion through a fluid creates about itself a circulation of sufficient strength to hold the rear stagnation point at the trailing edge. The Kutta condition may be interpreted as the requirement that the flows over the upper and lower surfaces merge smoothly at a sharp trailing edge for lifting airfoils.

CHAPTER 3

BASIC MULTIPLE-GRID METHOD

3.1 INTRODUCTION

The foundation on which the present solver for a general multiple-grid structure is based is the Ni multiple-grid algorithm for the solution of the Euler Equations [14]. This algorithm is composed of two parts, a "base solver" and a "coarse mesh accelerator". To illustrate the basic Ni algorithm, the solution mesh is considered to be comprised of a single global grid called the level h mesh. The first part, the base solver, is a single step explicit Lax-Wendroff type time marching method used on this solution mesh. The second part is a coarse mesh accelerator which operates on residuals transferred from the solution mesh to one or more progressively coarser grids. The key to both parts of Ni's multiple-grid scheme is the formulation of the discrete equations in terms of a control volume integration of the governing equations over each grid cell. The sum of this control volume integration, which may be called the cell residual or change, is then transferred to the surrounding grid points by way of a "distribution" formula. The resulting

formulae for the corrections to grid point variables is equivalent to a standard Lax-Wendroff time step at each grid point. One of the advantages of the Ni multiple-grid method is that both the base solver and the coarse mesh accelerator operate on the solution U and the change in solution dU ($dU = U^{n+1}-U^{n}$) which need be only stored for the solution mesh. This represents a savings in storage over the traditional multi-grid algorithm which stores the solution on each level.

While the present scheme uses a Lax-Wendroff type time marching scheme as the base solver, the method is not restricted to only this base solver. Johnson [39] has shown the coarse mesh accelerator to work equally well when combined with other base solvers such as those of MacCormack, Lapidus, and Burstein. Therefore, a great deal of flexibility is possible in the choice of the base solver. The advantages of using one particular base solver over another are in the reduction in computational work required for each relaxation sweep, ease of application of boundary conditions, and possible improvements in acceleration to convergence. For the present work the prime concern has been to demonstrate the extension of the multiple-grid method to general embedded mesh structures. In this light we have remained with the original Lax-Wendroff base solver, it being a well established starting point, while trying not to restrict the work to this solver. For the inviscid transonic flow problems which have been considered during the code development this base solver has

proven to be very robust, providing solutions even with poor boundary condition formulations and minor errors during development. This point was particularly helpful during the development of the embedded mesh formulations.

It is not the intent here to rederive the Ni formulation presented in [14]. Rather, for completeness, the final formulation of the multiple-grid method will be presented for a general nonorthogonal grid system. Where possible, observations from the current work with the method have been included to help clarify areas that are unclear in the original paper. These areas include the implementation of boundary conditions and parts of the coarse mesh accelerator which were not described in the original paper and therefore are probably different from the implementation used by Ni. In the present formulation, both the base solver and the coarse mesh accelerator will be expressed in a cell reference frame using numerical values for grid points, cell centers, etc.. This choice of reference frame has been made in preparation for the pointer system to be presented in chapter 4.

3.2 BASE SOLVER

The base solver performs a Lax-Wendroff step in time for each point on the finest mesh, referred to here as the h mesh. This process is implemented in three passes over the mesh. First, the mesh is swept node by node initializing all grid point corrections ($dU = U^{n+1} - U^{n}$) to zero. Next the h mesh is

swept cell by cell calculating the new grid point corrections. This is the solver sweep where a control volume flux balance and distribution are performed for each cell. Finally, the boundary conditions are applied and the mesh is swept once again by nodes updating the solution. To help clarify the general flow of the base solver during the discussion which follows figure 3-1 presents a flow chart which summarizes this process.

After the initialization sweep the solution sweep is made, cell by cell, performing a flux balance and distribution for each cell. For the typical cell shown in figure 3-2 this involves the following 3 steps.

STEP 1: Finite volume approximation

$$DU_c = \text{Cell Residual}$$

$$= (Dt/DV)\{ [0.5(F_1+F_2)(y_2-y_1)-0.5(G_1+G_2)(x_2-x_1)] \quad (3.1a)$$

$$-[0.5(F_3+F_4)(y_3-y_4)-0.5(G_3+G_4)(x_3-x_4)]$$

$$+[0.5(G_1+G_4)(x_4-x_1)-0.5(F_1+F_4)(y_4-y_1)]$$

$$-[0.5(G_2+G_3)(x_3-x_2)-0.5(F_2+F_3)(y_3-y_2)]\}$$

where

$$DV = -0.5[(x_3-x_1)(y_4-y_2) - (x_4-x_2)(y_3-y_1)] \quad (3.1b)$$

This step is a discrete approximation to the governing integral equation (2.10) on a cell volume whose shape is invariant with time.

STEP 2: Distribution formulae

$$dU_1 = dU_1 + 0.25[\, DU_c - Df_c - Dg_c \,] \qquad (3.2a)$$

$$dU_2 = dU_2 + 0.25[\, DU_c - Df_c + Dg_c \,]$$

$$dU_3 = dU_3 + 0.25[\, DU_c + Df_c + Dg_c \,]$$

$$dU_4 = dU_4 + 0.25[\, DU_c + Df_c - Dg_c \,]$$

where

$$Df_c = (Dt/ DV)[\, DF_c \overset{1}{Dy} - DG_c \overset{1}{Dx} \,] \qquad (3.2b)$$

$$Dg_c = (Dt/ DV)[\, DG_c \overset{m}{Dx} - DF_c \overset{m}{Dy} \,]$$

and

$$DF_c = (\partial F/\partial U)_c DU_c \qquad DG_c = (\partial G/\partial U)_c DU_c \qquad (3.2c)$$

$$\overset{1}{Dx} = 0.5(x_2 + x_3 - x_1 - x_4) \qquad \overset{1}{Dy} = 0.5(y_2 + y_3 - y_1 - y_4)$$

$$\overset{m}{Dx} = 0.5(x_3 + x_4 - x_1 - x_2) \qquad \overset{m}{Dy} = 0.5(y_3 + y_4 - y_1 - y_2)$$

$$U_c = 0.25(U_1 + U_2 + U_3 + U_4)$$

This step "distributes" the cell residual of step 1 proportionally to the solution grid points resulting in a Lax-Wendroff type formulation of the grid point correction equations. Expressed in this form, the numerical signal propagation phenomena appears similar in nature to characteristics propagation [14]. In this distribution formula, $(\partial F/\partial U)_c$ and $(\partial G/\partial U)_c$ are the Jacobian matrices

evaluated at the cell center in terms of U_c. As Ni points out, a significant number of operations can be saved if DF_c and DG_c are directly formulated in terms of U_c and DU_c before coding.

STEP 3: Smoothing formulation

While Lax-Wendroff type algorithms are known to have a significant amount of implicit artificial smoothing, for transonic and supersonic flows with shocks additional explicit artificial smoothing is required to stabilize the solution. From the author's experience, when the multiple-grid accelerator is used this smoothing greatly improves the convergence rate, and, in many cases, is required for convergence. The present smoothing formulation, expressed here in a distribution format, would in practice be included in step 2.

$$dU_1 = dU_1 + 0.25\mu [\, U_c - U_1 \,] \qquad (3.3)$$

$$dU_2 = dU_2 + 0.25\mu [\, U_c - U_2 \,]$$

$$dU_3 = dU_3 + 0.25\mu [\, U_c - U_3 \,]$$

$$dU_4 = dU_4 + 0.25\mu [\, U_c - U_4 \,]$$

$$\mu = \sigma \, Dt \, [\, Dl + Dm \,] / DV$$

$$Dl = \sqrt{(Dx_l)^2 + (Dy_l)^2} \qquad Dm = \sqrt{(Dx_m)^2 + (Dy_m)^2}$$

While the distribution format presented above might suggest a smoothing applied over the cell, this is really a

smoothing operator applied directly to the nodal solution. The net contributions from the four surrounding cells results in the standard nine point Laplacian smoothing operator applied on the computational mesh. For a cartesian coordinate system, with Dx = Dy, this smoothing is equivalent to adding a term of order Dx to the original governing equations of the following form,

$$\sigma \; Dx \; \{ U_{xx} + U_{yy} \} \qquad (3.4)$$

In practice, the type and amount of smoothing is often determined through a trial and error process without any rigorous mathematical study. The addition of any type of artificial smoothing will add an error to the solution. It is hoped that this error will be very localized, stabilizing the solution near singularities in the flow while minimizing the overall global effect. One common way of reducing the detrimental effects of smoothing is to choose a form which adds one or more terms of higher order to the original governing equations. The present smoothing is particularly disturbing in this respect since a first order term is added to a second order accurate scheme, making the scheme spatially only first order accurate. Quantitatively, as will be shown in the results section, the present smoothing has a surprisingly small effect on global parameters of interest, such as the force coefficients, but does create errors in regions of rapid expansion or isentropic compressions. In addition, it does enhance convergence of multiple-grid

solutions and allows the calculation of transonic flows with shocks. Since smoothing is required by the present algorithm, much further work should be done in this area to formulate a smoothing which reduces these errors, preferably a smoothing operator which is of higher order than the present.

Once the solution sweep has been performed over each cell on the fine mesh, the required boundary conditions are applied to the boundary nodes. These will be discussed in more detail later. Finally, the third sweep over the h mesh is made, node by node, to update the dependent variables.

$$U_i^{n+1} = U_i^n + dU_i \tag{3.5}$$

The newly calculated value of U_i is equivalent to a second order accurate (in time) Lax-Wendroff method.

This completes the formulation of the basic solver on the solution mesh with the exception of the definition of the time step restriction. This time step restriction is determined by the stability limit of the Ni scheme applied to the governing equations (2.9). Unfortunately since the governing equations are nonlinear a stability analysis can not be done directly for this system. A good indication of the stability limit can be gained through analysis of a similar linear model equation, in this case the 2-D scalar wave equation. A Von Neuman analysis of the current scheme applied to the 2-D wave equation is presented in appendix B. The result of this analysis for a Cartesian system with Dx = Dy is

the stability restriction of $CFL \leq 1/\sqrt{2}$. Note for the corresponding 1-D equation the restriction is $CFL \leq 1$. On this basis one should expect the stability limit for the general Euler system to also be $CFL \leq 1/\sqrt{2}$, where the CFL number is appropriately defined for the new system.

The definition of the CFL number for the Euler equations is defined in terms of the maximum eigenvalues of the Jacobian matrices ($\partial F/\partial U$ and $\partial G/\partial U$) of the quasilinear form of the Euler equations for a general nonorthogonal coordinate system (eqn 2.9) and the cell dimensions. For a general two dimensional grid system we must satisfy a stability condition in both coordinate directions. For the present system these are,

$$CFL_\xi = \frac{Dt[|uDy^1 - vDx^1| + aD1]}{DV} \qquad CFL_\eta = \frac{Dt[|uDy^m - vDx^m| + aDm]}{DV} \tag{3.6}$$

where a is the speed of sound. A derivation of these expressions is presented on appendix C.

In practice both Ni and the present author have found that the stability limit $CFL \leq 1/\sqrt{2}$ is in fact much more restrictive than necessary. The limit has been found to be $CFL \leq 1$, giving the following time step restriction,

$$Dt <= MIN \left| \frac{DV}{|uDy^1 - vDx^1| + aD1}, \frac{DV}{|uDy^m - vDx^m| + aDm} \right| \tag{3.7}$$

The reasons for the less restrictive limit remains unanswered but the following observations can be made. First, it is

important to note that the true governing equation system is non-linear unlike the present analysis. It is possible that the non-linearity stabilizes the solutions scheme, but this seems unlikely since other numerical schemes, such as MacCormack's method, applied to the Euler Equations have stability limits that agree quite well with the results of 2-D wave equation analysis. Secondly, the addition of artificial viscosity to the model system as implemented here reduces the stability limit, eliminating it as a cause. Finally it is interesting to note that the observed stability limit corresponds to the 1-D limit. Since in practice the grid tends to be aligned with the flow in those regions where there are rapid changes on the flow, these regions could be viewed as 1-D along the coordinate direction. The stability limit might then in effect be the one dimensional limit. This same relaxation of the stability limit has been confirmed by Dannenhoffer [40], for 2-D solutions of the wave equation when the flow is aligned with the mesh while the expected limit of $CFL \leq 1/\sqrt{2}$ is required if it is not.

If the basic solver is used without the coarse grid accelerator, marching with a global time step based on the above relation, yields second order time accurate solutions. However, if only steady state solutions are of interest, much faster convergence is possible if each cell is advanced at the local rather than global time step condition. Of course, if the multiple-grid accelerator is used then the solutions are no longer time accurate and local time stepping is also used.

3.3 COARSE MESH ACCELERATOR

One of the largest problems with algorithms which have been developed in the past for solving the Euler equations has been their slow convergence to the steady state solutions. To be accepted as a design tool for repetitive calculations the method must provide steady state solutions as fast as or faster than current potential flow solvers. As might be expected, since the base solver is a single step Lax-Wendroff time marching scheme, the base solver used alone converges very slowly to the steady state solution. The poor convergence rate is due to the severe time step restriction, common with all explicit methods, for solutions on meshes with good flow resolution. To accelerate the convergence rate Ni developed the present multiple grid method which, while sacrificing time accuracy, accelerates convergence rates to the same order as current potential solvers.

The multiple grid method is formulated by considering a series of increasingly coarser meshes, defined as the 2h, 4h, 8h, etc. levels, which overlay the fine h mesh. If the governing equations were discretized on each of these levels it is then clear that the time step restriction would grow with the mesh scale as one moves to increasingly coarser meshes. Equivalently, the distance of propagation of disturbances during each time step will be on the order of the mesh scale. Therefore, disturbances propagate much faster with each step on coarser levels. Unfortunately, there is

also a corresponding loss of resolution and accuracy on the coarser levels. What is desired then, is a way of coupling the solutions of each level in such a way that disturbances which can be resolved on coarser levels are propagated on those levels while preserving the accuracy of the fine mesh solution.

One of the first attempts at this type of coupling was in solving for a solution in terms of a series of repeated mesh refinements. First the problem is solved for a very coarse mesh, where a very fast solution is possible due to the small number of mesh points and large time step possible. This solution is then interpolated to a finer mesh and used as the initial condition for the next solution. The process is then repeated for finer and finer meshes until the desired mesh is reached. The net work required using this method of mesh refinement is less that solving the fine mesh problem directly, but it still remains unacceptable for solution of Euler flow problems.

The Multi-grid methods developed by Brandt [23,24,41] take full advantage of the coupling of discrete equations on a system of mesh levels to provide very fast solutions to elliptic or near elliptic type equations. In multi-grid methods the discrete equations of each level are expressed in terms of the discrete governing equations of the given level plus a correction term relating the solution to the next finer level. The solution is then solved for simultaneously on all

levels by cycling through the levels, using a relaxation method on each level. One of the keys to the rapid convergence of the solution is the choice of a relaxation scheme which rapidly smooths the solution errors of the frequency of the mesh scale, often in just as few relaxation sweeps. By cycling through all levels of the mesh structure the complete spectrum of error frequencies are rapidly reduced. In addition, since a majority of the relaxation sweeps take place on coarser mesh levels where the work per sweep is small, the total computational work is reduced to the order of 6-9 work units. A work unit is defined here as the work for one relaxation sweep on the fine mesh. Unfortunately, such multi-grid methods are not directly applicable to solution of the Euler equations since the governing equation system is not elliptic.

Ni's multiple-grid method for solving the Euler equations represents a very different approach from the true multi-grid methods but the under lying concept of taking advantage of the propagation (or smoothing as Brandt calls it) of disturbances on coarser mesh levels remains the same. To help eliminate confusion of the present algorithm with true multi-grid methods the present algorithm will be called the coarse mesh accelerator. Ni begins by viewing the base solver in terms of a flux balance followed by a distribution. The flux balance defines the cell centered residual or solution change for the center of the cell. The distribution step then moves this residual to the surrounding nodes defining the

cell. It is the distribution step then which defines how disturbances propagate numerically throughout the domain. Now, due to the stability limit for the Lax-Wendroff scheme, the distance of propagation is limited to no more than one cell per time step. If the same distribution formula is considered for a coarser mesh level the maximum distance of propagation is still one cell, but now it is a coarse mesh cell which is a multiple of fine mesh cells. To retain the fine mesh accuracy Ni proposed defining the coarse mesh flux balance in terms of some weighted average of the fine mesh node residuals given by the base solver. Defined in this way a coarse mesh solution sweep filters out the lower frequency components of the fine mesh residual and propagates them as they would be by using the base solver on the coarse mesh. Therefore much faster convergence is possible for these low frequency errors than on the fine mesh. The accuracy of the fine mesh is preserved since the residuals are defined by the fine mesh and although redistributed, the net change at any point after the coarse mesh sweep will be of the same order as the fine mesh residuals to begin with. In other words, since the coarse mesh sweep operates on the residuals of the fine mesh discrete equation and not the solution, accuracy is preserved.

In practice the solution process begins with application of the base solver on the h mesh. The coarse mesh accelerator is then applied on the 2h mesh defined by eliminating every other mesh line in both directions. The 2h mesh solution

changes are then interpolated back to the fine mesh and added to the solution. The coarse mesh accelerator is then repeated on the 4h mesh, the 8h mesh, and so on. This defines one complete multiple-grid cycle. The mesh cycle is then repeated until convergence is reached. As will be shown later, use of the coarse mesh accelerator typically reduces the number of iterations or cycles to reach convergence by a factor three or better over the base solver alone.

Comparing the coarse mesh accelerator to the multi-grid methods of Brandt the following should be noted. The coarse mesh accelerator differs from true multi-grid, since only the distribution step on each level is the same as the base solver rather than the entire discrete equations. The coarse mesh accelerator simply accelerates the propagation of the fine mesh residuals. This could also be viewed as a smoothing of the fine mesh residuals on the coarser mesh levels. The similarity between the two is that the coarser levels are used to efficiently propagate or smooth solution errors of the frequency of the coarse grid scale. Finally while the multi-grid method are limited to elliptic type problems the present coarse mesh accelerator works well for convective type problems.

The actual coarse mesh accelerator consists of application of the following procedure on one or more progressively coarser meshes. Figure 3-3 presents a simple flow chart of this process. The process begins by elimination

of every other grid line in both coordinate directions resulting in what will be called the 2h mesh. A typical coarse grid cell is shown in figure 3-4. Note we now have access to the next level finer grid points, which we will refer to as secondary nodes, shown as points 5-9 in the figure. First dU at all 2h grid points is initialized to zero. Then, following the basic solver, the 2h mesh is swept cell by cell performing the following steps:

STEP 1: Residual Transfer

To retain the accuracy of the level h mesh solution the change, or cell residual, for the center of the 2h cell is determined from a weighted average of the level h mesh corrections. Ni denotes this symbollically by defining a transfer operator T_h^{2h} as,

$$DU_c^{2h} = T_h^{2h}\, dU^h \tag{3.8}$$

The simplest form is straight injection of the fine grid corrections as

$$DU_c^{2h} = dU_5^h \tag{3.9}$$

Injection of the cell centered residual physically represents a good approximation to the flux balance over the coarse mesh cell. This can be shown by writing out the complete expression for dU_5^h which is the sum of distributed changes from the four surrounding fine mesh cell. This sum is

equivalent to an application of the integral equation using all the h mesh points plus an extra second order time term as shown below,

$$DU_5^h = 0.25(DU_A^h + DU_B^h + DU_C^h + DU_D^h) + Dt^2 (\text{other terms}) \qquad (3.10)$$

where subscripts A, B, C, and D correspond to the four surrounding fine mesh cells as shown in figure 3-5. Note that a flux balance defined in this manner is a lagged flux balance based on conditions of the flow before application of the base solver.

For moderately stretched meshes simple injection works quite well. However, as the mesh becomes highly stretched in either or both coordinate directions the overall performance of the coarse mesh accelerator has been found to decrease, eventually failing completely. This breakdown has been found to be related to the type of transfer operator used. To correct this breakdown a series of different operators have been studied. In comparing the relative performance of the following operators it is convenient to use the simple injection formulation as a base line.

The first type of transfer operators considered were different algebraic weightings of the residuals of the nine nodes defining the cell. Straight injection of the cell center residual is the simplest of this class of operators. Of this class the following seemed promising,

$$DU^{2h} = [\, 4dU^{h}_{5} + 2(\, dU^{h}_{6} + dU^{h}_{7} + dU^{h}_{8} + dU^{h}_{9} \,) + (\, dU^{h}_{1} + dU^{h}_{2} + dU^{h}_{3} + dU^{h}_{4} \,) \,]/16 \qquad (3.11)$$

This represents an averaging of the residuals for each of the four h cells, which are then averaged for the 2h cell center value. This is consistent with the base solver flux balance since is assumes a linear distribution for residual along each face of the h cells. Unfortunately the above weighting, as well as all other purely algebraic averaging formulas considered were found to give the same performance as simple injection. Since each requires many more computational operations over simple injection there is a net loss in the efficiency by using these transfer formulations.

Judging by the performance of algebraic type weightings and noting that the decrease in performance occurs with high mesh stretchings, the resulting breakdown might be attributed to the variation of cell volumes between the four h mesh cells. The algebraic weightings do not take this cell volume variation into account. On this basis the following cell volume weighting was proposed,

$$DU^{2h}_{C} = [\, DV_A\, dU^{h}_{A} + DV_B\, dU^{h}_{B} + DV_C\, dU^{h}_{C} + DV_D\, dU^{h}_{D} \,]/DV_T \qquad (3.12a)$$

where

$$DV_T = DV_A + DV_B + DV_C + DV_D$$

and

$$dU^h_A = [\, dU^h_1 + dU^h_6 + dU^h_5 + dU^h_9 \,]/4 \qquad dU^h_C = [\, dU^h_5 + dU^h_7 + dU^h_3 + dU^h_8 \,]/4$$

$$dU^h_B = [\, dU^h_6 + dU^h_2 + dU^h_7 + dU^h_5 \,]/4 \qquad dU^h_D = [\, dU^h_9 + dU^h_5 + dU^h_8 + dU^h_4 \,]/4 \tag{3.12b}$$

This volume weighting also gave the same performance as simple injection with no improvement in performance. While it was felt that the volume weighting should play a role in the transfer operation it was clear that a key element was still missing from the formulation.

The missing element in the transfer operator for highly stretched meshes has been found to be the incorporation of the signal propagation characteristics of the equation into the transfer process. Assuming a definition of the residual dU at the center of each of the four h mesh cells (A,B,C,D of figure 3-5), the signal propagation must be considered in combining these residuals for the 2h cell centered residual dU . The signal propagation is determined by the distribution step (recall eqn. 3.2). Using this as a guide then,

$$\begin{aligned} DU^{2h}_C = {} & [\, dU^h_A + Df^\star_A + Dg^\star_A \,]DV_A/DV_T + [\, dU^h_B + Df^\star_B - Dg^\star_B \,]DV_B/DV_T \\ & + [\, dU^h_C - Df^\star_C - Dg^\star_C \,]DV_C/DV_T + [\, dU^h_D - Df^\star_D + Dg^\star_D \,]DV_D/DV_T \end{aligned} \tag{3.13}$$

where volumes are defined by equation (3.1b) and for example

$$Df_A^{\star} = \left(Dt/DV \right)_A \left[DF_A^{\star\, l} Dy_A - DG_A^{\star\, l} Dx_A \right] \qquad (3.13b)$$

$$Dg_A^{\star} = \left(Dt/DV \right)_A \left[DG_A^{\star\, m} Dx_A - DF_A^{\star\, m} Dy_A \right]$$

$$DF_A^{\star} = \left(\partial F/\partial U \right)_A dU_A^h \qquad DG_A^{\star} = \left(\partial G/\partial U \right)_A dU_A^h$$

There now remains the question of how to define dU_A, dU_B, dU_C, and dU_D. The proper definition of these h mesh cell residuals was determined by testing various averages. Three of the possible formulations tried are described below. The first is simply the cell center average defined above in equation (3.12b). Using this definition with the distribution defined above results in a transfer operator which is stable for highly stretched meshes but with an acceleration of convergence which is still less than satisfactory.

A second definition which improves the convergence rate is

$$dU_A^h = \left[dU_1^h + dU_5^h \right]/2 \qquad dU_C^h = \left[dU_2^h + dU_5^h \right]/2$$

$$dU_B^h = \left[dU_3^h + dU_5^h \right]/2 \qquad dU_D^h = \left[dU_4^h + dU_5^h \right]/2 \qquad (3.14)$$

At this point a trend can be seen. This definition provides a stronger weighting and influence of the residuals at the 2h cell corner nodes (points 3,5,7,9 of fig 3-5). Taking this one step farther results in the following weighting.

$$\begin{aligned} {}^{h}_{A}dU &= {}^{h}_{1}dU & {}^{h}_{C}dU &= {}^{h}_{3}dU \\ {}^{h}_{B}dU &= {}^{h}_{2}dU & {}^{h}_{D}dU &= {}^{h}_{4}dU \end{aligned} \qquad (3.15)$$

This final form provides the best convergence for highly stretched meshes. In hind sight there are several reasons to expect this result. First, of all weightings considered, the last takes the fullest advantage of the propagation of information in a manner consistent with the numerical propagation of information by the discrete equations. Defining the transfer operator in this way is equivalent to a time step on the 2h mesh. Second, equation (3.15) operates only on the residual of the 2h mesh at the 2h mesh points, thus filtering the residuals in the same manner as the discrete equations applied on a 2h mesh would filter the solution. This means the frequency of the errors resolved are the same that would be normally resolved by the discrete equations. The improvement in convergence by using a transport operator has been confirmed by Ni [42].

Table 3-1 summarizes quantitatively the relative performance of the different transport operators in terms of the multiple grid cycles for a converged solution. Two cases are shown on the table. Both are for a subsonic flow about a NACA0012 airfoil using a 65*17 O-type mesh with 3 mesh levels (65*17 fine mesh plus two coarser level). In the first case a moderately stretched mesh was used, placing the farfield

boundary at 5 chords from the airfoil. The second is for a highly stretched mesh placing the farfield radius at 20 chords from the airfoil. As the table shows all the operators perform about the same for the moderately stretched mesh with a slight improvement using the distribution formulations. For the more severe case however, all but the distribution based transfer operators fail. Of the distribution formulations it is clear the final formulation provides the best overall performance.

Table 3-1: Comparison of Transfer Operator Performance

TRANSFER OPERATOR	CASE 1: MODERATE STRETCHING, R = 5	CASE 2: HIGH STRETCHING, R = 20
SIMPLE INJECTION (Eqn. 3.9)	616	Failed to Converge
ALGEBRAIC WEIGHTING (Eqn. 3.11)	690	Failed to Converge
VOLUME WEIGHTING (Eqn. 3.11)	588	Failed to converge
DISTRIBUTION TYPE 1: (Eqn. 3.12b)	572	850
DISTRIBUTION TYPE 2: (Eqn. 3.14)	573	854
DISTRIBUTION TYPE 3: (Eqn. 3.15)	578	619

STEP 2: Distribution formulae

Following the distribution step of the base solver equation (3.2) the 2h mesh distribution is defined as

$$dU_1 = dU_1 + 0.25[\, DU_c - Df_c - Dg_c \,] \qquad (3.16a)$$

$$dU_2 = dU_2 + 0.25[\, DU_c - Df_c + Dg_c \,]$$

$$dU_3 = dU_3 + 0.25[\, DU_c + Df_c + Dg_c \,]$$

$$dU_4 = dU_4 + 0.25[\, DU_c + Df_c - Dg_c \,]$$

where

$$Df_c = (Dt/ DV)[\, DF_c \, Dy^1 - DG_c \, Dx^1 \,] \qquad (3.16b)$$

$$Dg_c = (Dt/ DV)[\, DG_c \, Dx^m - DF_c \, Dy^m \,]$$

and

$$DF_c = (\partial F/ \partial U)_c \, DU_c \qquad DG_c = (\partial G/ \partial U)_c \, DU_c \qquad (3.16c)$$

$$Dx^1 = 0.5(x_2 + x_3 - x_1 - x_4) \qquad Dy^1 = 0.5(y_2 + y_3 - y_1 - y_4)$$

$$Dx^m = 0.5(x_3 + x_4 - x_1 - x_2) \qquad Dy^m = 0.5(y_3 + y_4 - y_1 - y_2)$$

Since the solution is now known at the cell center (i.e. node 5), this is used for calculation of $(\partial F/\partial U)_c$ and $(\partial G/\partial U)_c$.

Once the above steps have been performed at all 2h cells, the boundary conditions are applied at all 2h boundary points. Note that no smoothing is done for the coarse mesh sweeps as is done in step 3 of the basic solver. Then the corrections are interpolated back to the fine mesh using bilinear interpolation.

$$
\begin{aligned}
{}^{h}dU_{6} &= 0.5\left[{}^{2h}dU_{1} + {}^{2h}dU_{2} \right] & {}^{h}dU_{7} &= 0.5\left[{}^{2h}dU_{2} + {}^{2h}dU_{3} \right] \\
{}^{h}dU_{8} &= 0.5\left[{}^{2h}dU_{3} + {}^{2h}dU_{4} \right] & {}^{h}dU_{9} &= 0.5\left[{}^{2h}dU_{1} + {}^{2h}dU_{4} \right] \\
{}^{h}dU_{5} &= 0.5\left[{}^{h}dU_{6} + {}^{h}dU_{8} \right] & & \qquad (3.17)
\end{aligned}
$$

It should be noted that the above interpolation is a bilinear interpolation in computational space. It has been found that while it might seem to be better to perform a bilinear interpolation in physical space the above form actually gives better convergence rates. In addition the above formulation requires far less operations than the corresponding formulation in physical space, providing a significant savings in computational work.

Finally the boundary conditions are applied once again on the fine mesh and the solution is updated using equation (3.5). The above coarse mesh accelerator is then repeated for progressively coarser meshes (i.e. 4h,8h,....). A complete multiple-grid cycle consists of one sweep through the level h solution mesh followed by a coarse 2h mesh sweep, followed by a 4h sweep, and so on to the coarsest mesh.

3.4 BOUNDARY CONDITIONS

Each of the boundary conditions used has been implemented in a predictor/corrector form. The

predictor/corrector form follows from the fact that second order numerical integration schemes for internal points incorporate a mathematical signal propagation phenomena analogous to the theory of characteristics. For example, Abbett [43] and others have viewed MacCormack's scheme as computing the solution of two simple waves, the solutions of which are summed to yield a complete solution. In the same sense Ni suggests that the "distribution" formula represent similar simple wave solutions. On boundaries the predictor step consists of summing contributions from cells interior to the boundary. The corrector step consists of enforcement of the appropriate boundary conditions (i.e. inflow, outflow, solid wall, or Kutta) using a simple wave type of treatment.

In this section, subscript "p" defines predicted values obtained by distributions from the two boundary cells belonging to point i. Subscript "c" refers to the corrected values after application of the boundary conditions. Once found the corrected change at boundary points is then

$$dU_i = U_c - U_i^n \tag{3.18}$$

The corrector step for the farfield and solid wall boundaries is based on a characteristic analysis of the linearized Euler equations in a coordinate system tangential and normal to the boundary at point 1, as shown in figure 3-6. A general and easy to follow development of this characteristic analysis is presented by McCartin [44].

Rewriting the original governing equation (2.8) into this local cartesian reference frame with normal and tangential coordinates (n,s) gives

$$U_t + F_n + G_s = 0 \tag{3.19}$$

Assuming locally that the tangential variation is much smaller than the normal variation of U, this equation reduces to

$$U_t + F_n = 0 \tag{3.20a}$$

which may be rewritten as

$$U_t + F_U U_n = 0 \tag{3.20b}$$

Now performing a local linearization by freezing the values of F_U this equation can then be transformed into a system of uncoupled scalar equations, the characteristic equations

$$W_t + \lambda W_n = 0 \tag{3.21}$$

If q_n and q_s are defined as the normal and tangent velocity components, and a is the speed of sound, then the eigenvalues λ and corresponding characteristic variables W of this equation in the reference frame normal to the boundary are

$$\lambda = \begin{vmatrix} q_n \\ q_n \\ q_n + a \\ q_n - a \end{vmatrix} \qquad W = \begin{vmatrix} \rho - p/(\bar{a}^2) \\ q_s \\ [q_n + p/(\bar{\rho}\bar{a})]/\sqrt{2} \\ [-q_n + p/(\bar{\rho}\bar{a})]/\sqrt{2} \end{vmatrix} \tag{3.22}$$

Barred quantities are linearized state conditions which are

taken as the predictor state (p).

3.4.1 Far Field Boundary

The number of boundary conditions which may be specified at the boundary is equal to the number of positive eigenvalues. For the far field the specification of the boundary conditions depends upon whether the normal velocity is positive (inflow) or negative (outflow) and supersonic or subsonic. For subsonic inflow ($0<q_n<a$) the three positive eigenvalues require three boundary conditions be applied while W4 must come from the flow interior to the boundary. The interior flow is represented by the predicted values. The boundary conditions are set by defining W1, W2, and W3 in terms of the finite radius far field vortex conditions, $(\ )_v$. These correspond to a farfield flow projected normal and tangential to the boundary. We thus have the following system of equations defining the corrected state, $(\ )_c$.

$$\rho_c - p_c/(\bar{a}^2) = \rho_v - p_v/(\bar{a}^2) \tag{3.23}$$

$$q_{s_c} = q_{s_v}$$

$$q_{n_c} + p_c/(\bar{\rho}\bar{a}) = q_{n_v} + p_v/(\bar{\rho}\bar{a})$$

$$-q_{n_c} + p_c/(\bar{\rho}\bar{a}) = -q_{n_p} + p_p/(\bar{\rho}\bar{a})$$

After recombination we have,

$$q_{s_c} = q_{s_v} \tag{3.24}$$

$$p_c = 0.5[\, p_v + p_p + \bar{\rho}\bar{a}(q_{n_v} - q_{n_p})]$$

$$\rho_c = \rho_v + (p_c - p_v)/(\bar{a}^2)$$

$$q_{n_c} = q_{n_v} + (p_v - p_c)/(\bar{\rho}\bar{a})$$

For supersonic inflow ($q_n > a$) all eigenvalues are positive and four boundary conditions are required. In this case the inflow boundary is frozen at the freestream conditions.

For subsonic outflow ($-a < q_n < 0$) there is only one positive eigenvalue and therefore one required boundary condition. On the outflow boundary the upstream traveling characteristic W3 is set at the freestream value. W1, W2, and W4 are determined from the predicted flow conditions. After rearranging we have the following relations,

$$p_c = 0.5[\, p_v + p_p + \bar{\rho}\bar{a}(q_{n_v} - q_{n_p})] \tag{3.25}$$

$$q_{s_c} = q_{s_p}$$

$$\rho_c = \rho_p + (p_v - p_p)/(\bar{a}^2)$$

$$q_{n_c} = q_{n_p} + (p_v - p_p)/(\bar{\rho}\bar{a})$$

For supersonic outflow ($q_n < -c$) all information comes from the predicted state p.

3.4.2 Solid Wall Boundary

Finite volume methods with the state vectors defined at cell centers (e.g. [15,17]) only require the pressure on the solid wall. Incorporation of the solid wall condition for Ni's scheme requires all flow quantities be known or determined at the solid surface. For this reason, a characteristic analysis is also used at the solid walls. Referring back to the boundary cells shown in figure 3-6 and with $q_n=0$ in eqn. 3.22, there is one positive eigenvalue requiring one boundary condition be set. The condition used is $q_n=0$. W1, W2, and W3 are then determined based on the predicted state (p) where

$$(U_1)_p = U_1^n + 2(dU_1)_p \qquad (3.26)$$

The factor of two in the above expression is used to accelerate convergence. This might be thought of as either a crude application of the reflection principle at the solid wall or merely a over relaxation of the predicted change. After substitution and recombination, the corrected conditions are found to be

$$q_{n_c} = 0 \qquad (3.27)$$

$$q_{s_c} = q_{s_p}$$

$$p_c = p_p + q_{n_p} \bar{\rho} \bar{a}$$

$$\rho_c = \rho_p + q_{n_p} \bar{\rho}/\bar{a}$$

It is interesting to note that this solid wall boundary condition is a linearized version of the common simple wave boundary condition, where the corrected state is based on the generation of an isentropic expansion or compression wave normal to the boundary which is of sufficient strength to cancel q_n .

3.4.3 Kutta Condition

All airfoil solutions to be presented have been obtained on either O-type or C-type meshes. This places a mesh point at the trailing edge of the airfoil which is a singular point in the flow field. At this point the procedure used to enforce the body boundary condition should be modified to enforce a Kutta condition. As stated in chapter 2 the Kutta condition required that the flow on the upper and lower surfaces of a sharp trailing edge must merge smoothly. In real flows it is the viscosity, no matter how small, which guarantees that the Kutta condition will be satisfied. While

theoretically an inviscid solution requires a Kutta condition, for numerical calculations the artificial smoothing, both that implicit in the algorithm and that added explicitly for stability, have been found to impose this condition automatically. The artificial smoothing performs the same function as the viscous terms in a true viscous flow. Therefore for the present calculations the Kutta condition is met without any special boundary condition. This approach agrees with other published Euler calculations [15,17].

One question now remains in the treatment of the trailing edge point. Since it is also a singular point in the flow field, what boundary conditions should be imposed at this point? The best results have been found to depend on the type of mesh used. For an O-type mesh the trailing edge point is consider to be single valued. The most reliable procedure is to not apply any condition and simply use the predicted changes at this point. For C-type meshes the best procedure is to consider the trailing edge point a double valued point and applying the standard solid wall boundary condition using the local tangent for the upper and lower surface at respective upper and lower points. While these treatments of the trailing edge points may seem rather arbitrary, they were choosen after studying many possible formulations including enforcing a stagnation point and various flow angle conditions. It was found that while the local flow detail at the trailing edge varied with various conditions imposed, there was vary little variation in the global properties of

the solution. This conclusion is consistent with results of other calculations [45].

3.4.4 Boundary Smoothing Formulation

In addition to the implementation of the physical boundary conditions a special formulation of the smoothing is required along the boundaries of the computational domain. As noted in the smoothing discussion (Step 3 of the base solver), the smoothing currently used for the internal solution points is a nine point Laplacian type smoothing operator. Without specifying information outside the computational domain, this operator cannot be constructed along the domain boundary. The approach currently used is to drop the smoothing along computational lines running into the domain, thereby reducing the smoothing operator to a one dimensional operator tangent to the boundary. Considering the boundary cell shown in figure 3-7, where points 1 and 4 lie along the boundary, the corresponding one-dimensional smoothing operator is

$$dU_1 = dU_1 + 0.25\mu [\, U_4 - U_1 \,] \qquad (3.28)$$

$$dU_4 = dU_4 + 0.25\mu [\, U_1 - U_4 \,]$$

To insure the smoothing is continuous from points internal to the boundary to the boundary the viscosity μ is defined as the viscosity of the boundary cell and calculated using equation (3.3).

For O-type mesh calculations wiggles in the solution were found in some cases in the trailing edge region along the surface. The odd/even decoupling in this region is caused by the poor grid structure in this region (highly non-orthogonal). To eliminate this decoupling the surface smoothing was symmetrically increased about the trailing edge by linearly increasing the artificial viscosity from its normal value several points from the trailing edge to a maximum at the trailing edge. This amounts to nothing more than a cosmetic correction producing a smooth solution in this region but without affecting the global parameters of interest.

3.5 GLOBAL MULTIPLE-GRID SOLVER SOLUTIONS.

The basic multiple-grid Euler solver and boundary conditions have been verified for several different flow problems and a range of flow conditions. The objective of the present section is twofold; first, to validate the present formulation by comparing with known theoretical results, and second, to illustrate the sensitivity of these results to boundary conditions, grid resolution, smoothing and other important parameters. The first problem considered was the calculation of flows in a channel with a circular arc bump presented by Ni [14]. While of value, in the sense that the present formulation could be directly compared with that of Ni, the channel problem is not a good case for truly testing the performance of the solver. Since the present solver

reproduced those results of Ni, they will not be repeated here. The test cases which will be presented consist of several two-dimensional airfoil designs. These airfoils were chosen as good test cases either since an analytical solution is known or because they are commonly accepted test cases for code comparison. Two types of grids have been used for these solutions. The O-type meshes have been generated using a 2-D version of the transfinite interpolation routine described by Eriksson in [19] and supplied by the author. The C-type meshes were generated using a 2-D parabolic mapping routine written by Loyd [46].

The first test case to be considered is a NACA0012 airfoil with a uniform freestream Mach number of 0.63 and 2.0 degrees angle of attack. Under these conditions the flow is completely subsonic. A numerical solution of the streamfunction equation for this case is provided in [47], predicting a lift coefficient of 0.335. Figure 3-8 shows the near field of a 65*17 O-type mesh with a farfield radius of 5 chords. As a base line for comparision, figure 3-9 presents the surface pressure coefficient, surface total pressure loss, the near field Mach number contours, and the near field total pressure contours for a solution using the base solver alone. The streamfunction solution of [47] has been included in figure 3-9a to demonstrate that the correct pressure distribution along the airfoil surface has been found. In addition, the calculated lift coefficient of 0.324 agrees quite well with the streamfunction solution value of 0.335.

The total pressure loss, which should be zero for this case, has been presented since it has been found to be a very sensitive indicator of errors in the solver formulation and of poor grid resolution. Considering the grid resolution with only 65 points on the airfoil the total pressure loss is acceptable. Note that the total pressure loss is generated by the rapid expansion around the leading edge. This total pressure loss is due to the artificial smoothing of the solution. More will be said about the total pressure loss later. Figure 3-10 presents a multiple-grid solution on the same mesh with the same flow conditions where now in addition to the base solver applied on the h mesh the coarse mesh accelerator has been used on the 2h and 4h meshes. Comparing the solution of figure 3-10 with that of 3-9 it is clear that the multiple-grid solver gives almost identical solutions. Note that the lift and drag coefficients also agree as expected.

The convergence histories for these two cases are presented in figure 3-11a. Included in the figure is the convergence history for the intermediate case with only one coarser level. The residual presented here is the average change of $d(\rho U)/Dt$ after the base solver sweep as a function of the multiple grid cycle. Solution convergence for each case has been defined as the point at which this residual falls below 1E-5. This residual level is well beyond the point at which the force coefficients have reached their steady state values. To illustrate this, the lift and drag

coefficients for the solution of figure 3-10 are presented as a function of multiple-grid cycle in figure 3-11b. Each of the other solutions exhibited a similar behavior. As shown in figure 3-11a, the base solver alone required 1562 iterations to converge while the solution with three levels only required 570 cycles to converge resulting in a factor of 3 reduction in the total cycles by using the coarse mesh accelerator. It is also interesting to note that the relative reduction between 1, 2, and 3 levels decreases as the number of mesh levels is increased. This has also been noted by Johnson [39]. The reason for this trend becomes clear if one recalls that the actual finite volume approximation is only performed on the h mesh. All coarser levels are based on a weighted average of the fine mesh residual which is a lagged or old approximation to the flux balances on the fine mesh. In contrast true multi-grid methods always operate on the current solution and therefore do not show this trend.

Now consider a global refinement of the above mesh resulting in the 129*33 mesh with a farfield radius of 5 chords shown in figure 3-12. The previous 65*17 mesh (figure 3-8) was actually created by elimination of every other grid line in the 129*33 mesh. Figure 3-13 presents the solution for the same flow conditions as the previous result using the multiple grid solver with 4 global mesh levels. The only detectable changes from the 65*17 mesh solution are in the reduction in the total pressure loss and the slightly better lift and drag coefficients of .328 and .0009, respectively.

These changes are the direct result of the better mesh resolution. In particular, since the smoothing term scales as Dx, the factor of two reduction in total pressure loss is as expected. Figure 3-14 compares the convergence histories of the 129*33 solution and the 65*17 mesh solution with 3 mesh levels. If in addition to the difference in solution cycles one considers the better than factor of 4 increase in the work per cycle for the 129*33 mesh, the importance of minimizing the total number of grid points is clear.

At this time consider the origin and detail in the total pressure loss of figure 3-9. As previously mentioned, the origin of this loss is the artificial viscosity term which has been added to the governing equations to stabilize the solver. Careful study of the region near the stagnation point at the leading edge shows that in this region there is actually a negative total pressure error. This overshoot in the stagnation total pressure is caused by the smoothing of the rapid compression in this region. The total pressure error generated at the leading edge is also the result of smoothing, in this case the smoothing of the rapid expansion around the leading edge. After the leading edge region, the total pressure error remains almost constant until the trailing edge, which suggests that the formulation of the solid wall boundary condition is correct. Figures 3-15 and 3-16 show the surface pressure coefficient and total pressure errors, respectively, for a range of smoothing coefficients between 0 and 0.08. Note that as the smoothing coefficient decreases to

zero the total pressure loss almost disappears. Unfortunately for the case of zero smoothing (fig 3-15e and 3-16e) the solution failed to converge. It is clear that this breakdown is the result of an odd/even decoupling of the solution, particularly near the trailing edge. As will be shown shortly, this breakdown is due to the skewness of the mesh in the trailing edge region and the implementation of the boundary conditions at the trailing edge. Comparing the lift and drag coefficients as a function of the smoothing it is important to note that the absolute change in both coefficients is about the same. However, on a percentage basis, the drag coefficient, which should be zero for inviscid calculations, is very sensitive to the level of smoothing. For reasonable levels of smoothing (0-.05) it remains quite small. The lift coefficient, on the other hand, is insensitive to the amount of smoothing. As stated in the development of the smoothing, the amount of smoothing has a great effect on the convergence rate of the multiple-grid solutions. Figure 3-17 compares the convergence histories for the solutions of figures 3-15 and 3-16. It is clear that reasonable levels of smoothing greatly accelerate the rate of convergence, but beyond a threshold value, the convergence rate is insensitive to the level of smoothing. In conclusion, since the artificial viscosity is necessary, further work should be done to formulate a better smoothing operator.

One particular detail which remains to be explained is the total pressure spike at the trailing edge. This spike at the trailing edge is a localized effect due to skewness of the O-type mesh in this region. Figure 3-18 shows a blowup of the mesh in this region. This poor mesh structure is one of the problems with using an O-type mesh. The skewness in the mesh causes the breakdown of the solution noted earlier. In addition to the odd/even decoupling, the maximum solution residuals for the converged O-mesh solution always occur in the trailing edge region, giving a second indication of a breakdown in the formulation. This error can be reduced, if the mesh in this region is made more orthogonal to the surface, as shown in figure 3-19 for a 65*17 mesh. Unfortunately, by improving the orthogonality of the mesh in this region, the resolution downstream of the trailing edge is sacrificed.

Another alternative is to switch to a C-type mesh as shown in figure 3-20. This mesh is a 97*17 C-type mesh with 65 mesh points along the airfoil (note the grid distribution along the airfoil surface is not exactly the same as the O-type mesh). The corresponding solution is shown in figure 3-21. By switching to a C-type mesh the spike in total pressure has been removed. In addition, the odd/even decoupling of the solution, common with O-type meshes, no longer occurs. The maximum residual for the converged C-mesh solution no longer occurs in the trailing edge region. This indicates that the formulation is much more stable in this

region with an orthogonal mesh. Figures 3-22 and 3-23 present the surface pressure coefficient and total pressure loss for the same range of smoothing coefficients as used for the O-type mesh solutions of figures 3-15 and 3-16. Note that both the spike in the total pressure and the odd/even decoupling are not found for the C-type mesh solutions. Even with no smoothing the solution converged as shown in figures 3-22e and 3-23e. Also note, with zero smoothing both force coefficients show excellent agreement with the streamfunction solution values. Figure 3-24 shows the convergence histories as a function of the level of smoothing for these cases. The convergence rate is once again greatly improved with smoothing. In addition, comparing C-mesh convergence rates of figure 3-24 with those for the O-mesh of figure 3-17, shows that elimination of the odd/even decoupling greatly improves the rates of convergence. The maximum residual for these converged C-mesh solutions now occur in the stagnation point region of the leading edge. This agrees with the observation of other authors [48] who have found that the Lax-Wendroff scheme is only marginally stable in stagnation regions. There is one important drawback in switching to C-type meshes. While these meshes gain in resolution at the trailing edge, they also require many more points for equivalent resolution of the airfoil surface since packing along the surface leads to a band of unnecessary points stretching out to the farfield boundary.

Each of the above calculations were performed using the vortex far field characteristic boundary condition. Table 3-2 presents the lift and drag coefficients for solutions on a 65*17 O-type mesh with a far field radius ranging from 5 to 50 chords, with and without the far field vortex correction. If each is compared with the streamfunction solution lift coefficient value of 0.335 it is clear that, while the error in both cases drop off with increasing radius, by using the vortex correction the far field boundary may be brought much closer to the airfoil. This in turn reduces the storage and work by reducing the number of mesh points required for equivalent mesh resolution.

Table 4-3: Variation of Force Coefficients with Location and Type of Far Field Boundary Condition (Actual values $C_L = 0.335$ and $C_D = 0.000$)

FAR FIELD RADIUS (CHORDS)	UNIFORM FREESTREAM BOUNDARY CONDITION		VORTEX FREESTREAM BOUNDARY CONDITION	
	C_L	C_D	C_L	C_D
5	0.2873	0.0030	0.3238	0.0019
10	0.3059	0.0022	0.3266	0.0016
20	0.3170	0.0016	0.3276	0.0013
30	0.3211	0.0013	0.3284	0.0011
50	0.3245	0.0010	0.3289	0.0009

Returning to the solid wall boundary condition, recall that the predicted changes were multiplied by a factor of two (equation 3.26) with only a vague physical interpretation. The following example demonstrates the importance of this operation. Figure 3-25 presents the solution on the 65*17 O-type mesh of figure 3-8 without this operation. Comparing this solution with that of figure 3-9, it is clear that this operation has no significant effect on the final solution. It does, however, greatly improve the rate of convergence of the solution, as shown in figure 3-26.

The present global multiple-grid solver has been tested for several other transonic airfoil flow problems. The global solutions will be briefly described here and then used later for comparison to the embedded mesh results of chapter 5. The second case considered is a NACA0012 airfoil at flow conditions of $M = 0.85$ and $\alpha = 1.0$ degree. This is a lifting case with strong shocks at 85% chord on the upper surface and 70% chord on the lower surface. This case is often chosen as a test case because the lift is strongly dependent on the shock location (see for example the GAMM workshop [49]). A good solution then requires a high resolution of the shocks. Figure 3-27 presents a multiple-grid solution using 3 levels for the 65*17 mesh of figure 3-28. While both shocks are apparent, neither shock is very well defined due to the poor mesh resolution in the shock regions. A global mesh refinement was then made, resulting in the 129*33 mesh of figure 3-29. This mesh gives much better mesh resolution in

the shock regions as shown by the corresponding multiple-grid solution using 4 levels in figure 3-30. The total pressure losses across both shock is clearly resolved for the 129*33 global solution. The Mach number, pressure and total pressure jumps across these shocks are within 0.5 percent of those predicted by the normal shock relations based on the Mach number just ahead of the shock. The convergence histories for the 65*17 and 129*33 solutions are presented in figure 3-31.

A second common test case for code comparisons is the RAE2822 supercritical airfoil at M = 0.75 and α = 3.0 degrees [49]. The important features for this case are the very rapid expansion around the leading edge and a strong shock which occurs at 80% chord on the upper surface. Lerat and Sides [50] have published solutions of the Euler equation for this case using a explicit second order accurate finite-volume method. Their calculations predict lift and drag coefficients of 1.108 and 0.0424, respectively. A 65*17 mesh for this airfoil is shown in figure 3-32 with the corresponding multiple-grid solution shown in figure 3-33. Performing a global h/2 mesh refinement results in the 129*33 mesh shown in figure 3-34. A multiple-grid solution with 4 levels is presented in figure 3-35. Once again the better grid resolution of the 129*33 mesh results in a much sharper shock. In addition, as expected the total pressure loss in the leading edge region with the rapid expansion is reduced by a factor of two with the finer mesh. The surface pressure coefficient calculated by Lerat and Sides using a 188*24

C-type mesh with 129 points alone the airfoil has been included in figures 3-33a and 3-35a. The present calculation on the 129*33 fine mesh (figure 3-35a) agrees quite well with their solution. Considering the level of difficultly of this case, the present fine mesh lift and drag coefficient values of 1.088 and 0.0431 are also in very good agreement with the values of 1.108 and 0.0424 calculated by Lerat and Sides (within 2 percent). The convergence histories for these two cases are shown in figure 3-36.

The last case to be presented is the Garabedian and Korn supercritical airfoil [51] with design conditions of M = 0.75 and α = 0.12 degree. Since this airfoil was designed using an inverse hodograph method the theoretical flow solution at the design condition is known. The theoretical lift coefficient for this design condition is 0.63. At these conditions the supersonic region extends over about 60% of the upper surface. In practice, the shock-free solution for this configuration has been found to be very sensitive to the location of the sonic line and the resolution of the flow in the supersonic region. Full potential equation solutions by Jameson [52] and others have shown that a poor resolution of the sonic line will results in the supersonic region being terminated by a shock and not the proper shock-free solution. The location and strength of this shock is directly related to the mesh resolution used. Jameson [52] found that as the mesh was refined the strength the the shock decreased and that with the proper grid resolution (in this case a 256*65 O-type mesh) the

shock could be almost completely eliminated. Figures 3-37 and 3-38 present a 65*17 mesh and the corresponding solution using the present formulation. Comparing the theoretical pressure distribution with calculated solution in figure 3-38a shows that the flow is not being properly predicted over a portion of the supersonic region. This results in a lift coefficient which is 5.8% below the theoretical value of 0.63. The drag coefficient, which should be zero, is also much higher than what might be viewed as acceptable for this grid resolution. One possible explanation for the high drag coefficient is that a very weak shock is terminating the supersonic region. While the existence of such a shock can not be verified by the pressure distribution, due to the poor mesh resolution and possibly high smearing of such a shock, the rise in the total pressure loss (figure 3-38b) in this region would agree with this explanation. A 129*33 global mesh is shown in figure 3-39 with the corresponding multiple-grid solution presented in figure 3-40. Figure 3-40a shows that the calculated pressure coefficient is in slightly better agreement with the theoretical pressure distribution in the supersonic region. The higher mesh resolution has reduced the error in the lift coefficient to 3.4% below the correct value. The error in the drag coefficient has also been reduced by almost a factor of 2. Close inspection of the pressure distribution where the supersonic region terminates shows what could be the formation of a weak shock. Based on the improvement in the solution with this higher mesh resolution, it is resonable to expect

that an additional mesh refinement would result in a solution very close to the correct shock-free solution. With such a refinement the resulting mesh would be of the same resolution as required be Jameson in [52].

CHAPTER 4

POINTER SYSTEM

The transition from global mesh structures to general embedded mesh structures introduces a bookkeeping nightmare to what would appear to be a logical technical extension. The conventional approach in writing a global mesh solver begins with the assumption that the solution mesh will have a certain fixed structure. The solver is then constructed based on this structure making it an integral part of the code. A typical example is to assume the mesh consists of a rectangular array of nodes, for which an (i,j) labeling of the nodes is possible. Then by writing the solver in terms of indices i and j the mesh structure becomes inseparable from the code. With such an approach, the addition of an embedded mesh to the grid structure in essence means a new mesh structure has been defined. This in turn requires development of a new solver. Obviously this approach cannot be extended to handle general embedded mesh structures. The key to solving this dilemma is to separate the definition of the multiple-grid structure from the solver.

A very similar problem is commonly encountered in general finite element solvers where more than one type of element, each having a special shape (bar, triangle, rectangle) and composed of a varying combination of nodes (3,4,6,8,etc.), are assembled into a global finite element system to define a given structure. The key, once again, is to separate the structural definition from the solution algorithm. For finite element calculations this separation is done by assigning each node of the structure a node number (i) and then defining a connectivity array which describes the elements of the structure. The connectivity array often used is a sequential list of the elements, defining for each element the element type, nodes belonging to it, and any other attributes desired. This connectivity array is then used by a general solver as a guide for the systematic construction of the system of equations describing the structure. After solving, the array is then used as a guide for visual display of the structure and solution.

In many respects the separation of the general embedded mesh structure from the Euler solver is similar to the separation required in finite element problems. The embedded mesh structure envisioned would be composed of a combination of several global coarse mesh levels followed by one or more embedded regions containing several increasingly finer levels. The solver must know the location and domain (or domains, since there may be several) of each level. In addition the location and type of physical boundary conditions must be

defined. The solution is to define a pointer system which describes the mesh structure in the same way a connectivity array defines a finite element system. The pointer system then plays the role of a kind of road map for the solver. Separation of this structure definition from the Euler equation algorithm leaves a very general and easy to follow program. Changes in the grid structure then require a new pointer system but the solver remains unchanged.

Adopting a pointer system to describe the grid structure opens the door for many other benefits. Foremost is the flexibility possible in defining the structure. For applications implemented on virtual or array processing machines this flexibility means the pointer structure can be organized to optimize access time, page faulting and computational speeds. In addition, since the code is not modified with changes in the grid, adaptive mesh techniques can be implemented with the addition of new routines which manipulate the pointer system.

There exists many possible choices for definition of this pointer system, each with its own advantages and draw backs. Even the conventional (i,j) indexing for global calculations can be viewed as a very simple pointer system. In the discussion which follows, three possible pointer structures are presented and compared, the last of which was actually chosen. Before these pointer systems are discussed it is important to consider what information is required by

the solver.

With the present Ni formulation the solution U and change dU must be stored once and only once for each grid point on the finest mesh in each region of the total domain. For the full two dimensional Euler equations at least the following 10 quantities must be stored for each node: coordinates x and y, conservation variables ρ, ρu, ρv, and e,and the change in the conservation variables $d(\rho)$, $d(\rho u)$, $d(\rho v)$, and d(e). They are stored in a 10 by N solution matrix Q defined as,

$$Q = [\, Q_{mn} \,] \tag{4.1a}$$

where

$$\begin{aligned} m &= \text{Variable Type} \quad (1<=m<=10) \\ n &= \text{Node Number} \quad (1<=n<=N) \end{aligned} \tag{4.1b}$$

Additional quantities such as cell volumes, projected areas, temporary variables, etc. could also be stored to reduce repetitive calculations. At a minimum this implies a base line storage requirement of 10 real variables per node of the structure, independent of whether a pointer system is used or not.

4.1 POSSIBLE POINTER SYSTEMS

A cell pointer matrix must now be defined which points to the nodes in Q needed for the base solver and coarse mesh accelerator formulae on each level of the multiple-grid

structure. From the formulation of the base solver and coarse mesh accelerator it is clear that the smallest element which contains all required information for the solvers is the four node cell for the base solver and modified nine node cell for the coarse mesh accelerator. The nine node cell is then the most basic element common to both solvers. In view of the cell being the most basic element of the grid structure it follows that the structural definition of the pointer system should be composed around the cell or some higher structural grouping of cells. Figure 4-1 shows three possible base structures from which a global mesh structure could be composed. In order of decreasing structure or increasing flexibility they are: a rectangular block of cells, a line of cells, or simply an individual cell. Assuming the vector string of node information described above, a pointer system can be constructed for each base structure pointing to the nodes in this vector. It now becomes a trade off between the amount of storage and the degree of flexibility in the type of structure which can be defined. In the paragraphs which follow a pointer system using each of the base structures will be outlined. This is followed by a discussion of the advantages and disadvantages. For comparison, the storage of each system will be considered in terms of the storage required per cell of the grid structure.

The first base structure to be considered is the rectangular block structure. In this case we will consider each region (embedded or global) of each level as composed of

one or more rectangular blocks of cells. Considering a typical (N*M) block of cells shown in figure 4-2 with primary nodes (nodes used by both solvers) shown as solid dots and the additional secondary nodes of the next finer level (used by the coarse mesh accelerator) shown as open dots, a two dimensional integer pointer or an equivalent integer vector must be created to point to the proper location in Q for each node of the block. This process is then repeated for each block of cells used to define the total multiple-grid structure. The result is a three dimensional pointer array P, defined as followed,

$$P = [\, P_{inm} \,] \qquad (4.2a)$$

where

$$\begin{aligned} i &= \text{Block Number} && (1 <= i <= I) \\ n &= \text{Column Number of Block} && (1 <= i <= 2N+1) \\ m &= \text{Row Number of the Block} && (1 <= j <= 2M+1) \end{aligned} \qquad (4.2b)$$

Where P_{inm} (n=odd and m=odd) correspond to the primary nodes and all others are secondary nodes. In addition a directory consisting of the starting block number and ending block number of each level would also be required. Of course, in practice to save on storage this pointer array would be cast in a standard integer vector format, using a directory to list the starting and ending locations of each block in the vector along with the block dimension (N*M) and level. The storage required on a per cell basis, neglecting directory storage, can be estimated as follows,

$$\begin{aligned} \text{Storage/Cell} &= (\text{Integers Pointers per Block})/(\text{Cell per Block}) \\ &= [(2N+1)(2M+1)]/NM \\ &= 4 \text{ Integers/cell} \end{aligned}$$

The second possible base structure is a line of cells as shown in figure 4-3. In this case the multiple-grid structure would be decomposed into a set of lines of cells. The primary and secondary nodes are shown as before. Each line can be defined by a 3*N integer pointer array. Then combining all lines in a single 3-D array P we have,

$$P = [\ P_{inm}\ ] \qquad (4.3a)$$

where

$$\begin{aligned} i &= \text{Line Number} && (1<=i<=I) \\ n &= \text{Column Number of Line} && (1<=i<=3) \\ m &= \text{Row Number of the Line} && (1<=j<=2M+1) \end{aligned} \qquad (4.3b)$$

This pointer can also be converted into an integer vector with a corresponding directory. The storage required for this pointer structure is given by,

$$\begin{aligned} \text{Storage/Cell} &= (\text{Integers Pointers per Line})/(\text{Cells per Line}) \\ &= [3(2M+1)]/M \\ &= 6 \text{ Integers/cell} \end{aligned}$$

The last base structure to consider and the most basic is the cell itself. A typical cell is shown in figure 4-4, with primary and secondary nodes marked. Here a pointer vector P is constructed which contains 9 integer pointers per cell for every cell on every level of the multiple-grid structure. The cells are grouped by level, in any order on a given level, with a directory used to define the start and end of each level in the vector P. A detailed description of the pointer system will be presented later. It is clear that in this case the storage is simply

$$\begin{aligned} \text{Storage/Cell} &= \text{Integers Pointers per Cell} \\ &= 9 \text{ Integers/cell} \end{aligned}$$

The approximate storage requirements per cell are summarized in table 4-1. It is clear that, assuming a given multiple-grid mesh structure, the rectangular block structure is the most efficient in terms of storage.

Table 4-1: Comparison of Pointer Systems

BASE STRUCTURE	STORAGE/CELL	STRUCTURAL FLEXIBLITY
Block	4 integers	Highly Restrictive
Line	6 integers	Moderately Restrictive
Cell	9 integers	Least restrictive

While storage is an important factor there are many others which are equally if not more important depending on the application. Flexibility in terms of the ease in which multiple-grid structures, which include embedded regions, can be defined is very important. Using this criterion a pointer system based on the cell structure is the most flexible. Any number of embedded regions of any shape can easily be defined. Cells can be stored on any given level in any desired order allowing optimization of not only the accessing of the solution vector but also the pointer system. On the other hand the line and block formats are increasingly less flexible in this respect. For example, the definition of non-uniform embedded regions with the block format requires either the definition of many small blocks or the addition of unnecessary points to create larger blocks out of the irregular shapes. This flexibility is also important if the present scheme is to be extended in the future to adaptive mesh calculations. The

routines added for adaptive mesh calculation would be concerned with the addition and removal of cells from the pointer system to gain the resolution required. The more flexible the system the simpler the adaptive routines are, once again favoring the cell format. Two other factors, which may be important in choosing a pointer system are the amount of duplication of repetitive calculations and the ability to vectorize the system. Duplicate calculations, such as repeated calculations of cell dimensions could be reduced through the greater structure offered by the block structure, since it would be possible to construct the solver which operates on the complete block and not just cell by cell. For applications where vectorization is important either the line or block format may allow more vectorization of calculations than the cell format. This last point, if important, should be considered in more detail.

In light of the advantages and disadvantages presented, the cell format was chosen for having the maximum flexibility in the definition of the multiple-grid structure. The domain of a given level does not need to be simply connected, topologically restricted, or even defined in any order. It was also chosen in preparation of future possible adaptive mesh calculations, an area of growing importance. It must be kept in mind that this is not necessarily the best solution for every application, but the concept of a pointer system is very general and need not be limited to one particular solver or application.

Finally, while the present pointer system has been used simply as a means of overcoming the complexity of a multiple-grid structure with embedded mesh regions, it is possible to extend the pointer system to include any other information. In terms of a general modular approach not only mesh embedding but equation embedding may be required. Then the pointer system should not only define the cell structure but also the equations to be solved within the cell. In this manner the pointer system can change with the evolution of a general modular approach.

4.2 THE 2-D CELL POINTER SYSTEM

The current implementation of the cell pointer uses the following pointer definition. Nine pointers are required to define the nine nodes of each cell of every level of the grid structure (figure 4-4). They are stored in an integer matrix P defined as

$$P = [\, P_{ij} \,] \qquad (4.4a)$$

where

$$i = \text{Cell Node Number} \quad (1 \le i \le 9) \qquad (4.4b)$$
$$j = \text{Cell Number} \quad (1 \le j \le J)$$

Note that for the fine mesh cells the injection and interpolation points ($5 \le i \le 9$) are set to zero since those nodes don't exist. The value of point 5 is then the "switch indicator" as to whether the fine solver or the coarse mesh accelerator should be applied.

POINTER SYSTEM

For embedded mesh calculations nodes must be smoothed on different levels. This can be very efficiently handled by setting the sign of the corner pointers (1<=i<=4). If the node is to be smoothed by the cell it is positive, otherwise it is negative.

Finally there must be some way of knowing which level the cells belong to. By storing cells of the same level together (in any order), then only a pointer for the first and last cell of each level is required. The cell pointer matrix P combined with a level directory containing these level pointers completely defines the multiple-grid structure.

In addition to the basic grid structure, the location and type of boundary conditions must be defined for the solver. Boundary conditions are really exceptions to the general solver and can be problem dependent. Boundary conditions also tend to require different amounts of information and quite often access to domains larger than one cell. For these reasons, they are not included in the cell pointer matrix since, once defined, we would like this matrix definition to remain fixed.

At the present time there are two types of boundary pointers. As the need arises new forms can be added. Type 1 is used for solid wall, farfield boundaries, and any other boundary condition where pairs of cells are required. For each boundary node on the finest local mesh level the following information is stored in a 3 by K matrix called B1

$$Bl = [\ Bl_{ik}\ ] \tag{4.5a}$$

where

$$i = \begin{array}{ll} 1 & \text{Cell Number of Cell 1} \\ 2 & \text{Cell Number of Cell 2} \\ 3 & \text{Cell Orientation} \end{array} \tag{4.5b}$$

k = Boundary Point Number

The four possible cell orientations are shown in figure 4-5. For more than one boundary condition defined by this pointer all points of the same type are stored together alone with a starting and ending pointer for each boundary condition.

The second boundary pointer, type 2, is for boundary conditions that need only a cell side or string of 3 nodes as shown in fig 4-6. The embedded interface formulation is the only condition that uses this at this time. The definition of this pointer matrix B2 follows,

$$B2 = [\ B2_{ij}\ ] \tag{4.6a}$$

where

$$i = \begin{array}{ll} 1 & \text{For Node Number of Point 1} \\ 2 & \text{For Node Number of Point 2} \\ 3 & \text{For Node Number of Point 3} \end{array} \tag{4.6b}$$

j = Number of Interface Interpolation Point

This pointer is used to define the solution interpolation for point 2 along the embedded mesh interface before the embedded sweep and to zero the interface corrections at points 1,2, and 3 after the sweep. In B2 all sides on a given level are stored together from which a starting and ending pointer for each level is defined.

Clearly the pointer scheme described above provides a very flexible approach for dealing with complex grid structures. With an optimal grid structure the solution storage and computer time can be minimized. The price which must be paid for this flexibility appears in the total storage required and organization of the data base for vector computer architectures. While the solution storage is significantly reduced, the pointer system must also be stored. To illustrate the storage requirements let's compare the following two dimensional cases. In the first case we have a N*N global mesh with an embedded mesh (h/2) over one quarter of the domain as shown in figure 4-7a. For the second case we will consider a standard non-embedded mesh calculation where a global h/2 mesh refinement has been used to gain the same resolution as the first case, figure 4-7b. Storage of the solution in both cases requires storage of 10 real variables for each node of the finest mesh in each region (10 words(32 bit)/point); (10)(1.75)*N*N words for the first case and (10)(4)*N*N in the second. Neglecting the boundary pointers, the pointer system requires 9 integer variables for each cell of each level (9 half-words/cell or 4.5 words(32 bit)/cell), assuming two grid levels, the total pointer storage for the embedded case is (4.5)(2)(N-1)(N-1) words. A summary of the storage requirements for the two cases is presented in table 4-2. Comparing the total storage there is a reduction by using the pointer system but this reduction is less than might be expected if the pointers were not required. However, if

less than one quarter of the region is covered by an embedded mesh or if two embedded meshes are used, or more than 10 variables are stored at each node point the ratios will change in favor of the present approach.

Table 4-2: Storage Requirements for 2-D Example

	GLOBAL H/2 MESH	EMBEDDED H/2 MESH
NO. MESH POINTS	$4N^2$	$1.75N^2$
SOLUTION STORAGE	$40N^2$ Words	$17.5N^2$ Words
POINTER STORAGE	0 Words	$9(N-1)^2$ Words
TOTAL STORAGE	$40N^2$ Words	$26.5N^2 -18N+1$ Words

4.3 EXTENSION TO 3-D

All the problems of mesh resolution , storage, and computation times confronted in 2-D calculations tend to be amplified in the extension to three dimensions. Presently the limiting problems are large storage, long computing times and the ability to treat complex geometries. Fortunately, even though the problems are more severe, the payoff in adopting an embedded multiple-grid structure also increases dramatically. To illustrate the projected benefits of adopting a 3-D extension of the current cell based pointer system consider the corresponding base and coarse mesh accelerator cells (figure 4-8) for a three-dimensional version of Ni's scheme.

POINTER SYSTEM

As shown in figure 4-8b the number of integer pointers required per cell is three times that of the 2-D model. Both solvers use the 8 primary nodes (shown as solid dots), while the coarse mesh accelerator also requires the 19 secondary nodes (open nodes 9-27). Following the 2-D pointer system, we now define the cell pointer P as

$$P = [\, P_{ij} \,] \tag{4.7a}$$

where

$$\begin{aligned} i &= \text{Cell Node Number} \quad (1 <= i <= 27) \\ j &= \text{Cell Number} \quad (1 <= j <= J) \end{aligned} \tag{4.7b}$$

The cells are grouped in P by level in any order within the level and a level directory is created. In addition the corresponding boundary condition pointers can be constructed as required.

Now consider the storage requirements for two example cases. Beginning with a N*N*N global h mesh, consider a global h/2 mesh refinement using the standard global solver and, secondly, a embedded h/2 mesh refinement placed over 1/8 of the cube volume. The two mesh structures are illustrated in figure 4-9 with the corresponding storage requirements summarized in table 4-3 (columns 2 and 3). In both cases it has been assumed that there are two mesh levels used and the storage associated with the boundary condition pointers have been neglected. Note that the for 3-D solutions the baseline requirement for storage of the solution quantities at each node increases from 10 for the 2-D case to 13 due to the addition of the third coordinate z, conservation variable ρw, and it's change $d(\rho w)$. It is clear that for properly defined

embedded mesh regions there is a great potential for saving on storage by using the pointer system, even with the larger amount of pointer storage required. One reason for the large difference over the 2-D example is that in 3-D a global halving of the mesh results in a factor of 8 increase in the number of total nodes. It is important to note when trying to extrapolate this savings that the pointer storage is based on the assumption of half-word storage for integers. This places an upper limit on the total number of nodes at 32,768, the largest integer described by a half-word. Physically this corresponds to the number of nodes in a cube of mesh of dimensions 32*32*32. For larger systems of nodes full-word integers must be used which would double the amount of storage required for the pointer system. The fourth column in table 4-3 shows the storage for full-word integers.

Table 4-3: Storage Requirements for 3-D Example

	GLOBAL H/2 MESH	EMBEDDED H/2 MESH	FULL-WORD INTEGERS
NO. MESH POINTS	$8N^3$	$1.875N^3$	$1.875N^3$
SOLUTION STORAGE (words)	$104N^3$	$24.375N^3$	$24.375N^3$
POINTER STORAGE (words)	0	$27(N-1)^3$	$54(N-1)^3$
TOTAL STORAGE (words)	$104N^3$	$51.375N^3-81N^2$	$78.375N^3-162N^2$

CHAPTER 5

GENERAL EMBEDDED MESH FORMULATION

The ultimate objective of this research is the development of a general modular approach for solving complex flow problems. The proposed method of implementing such a modular approach is to cast the problem in terms of a general multiple grid structure, where local flow features are resolved through embedded mesh regions and embedded equation regions with higher approximation of the governing equations. Beginning with Ni's multiple-grid method for solution of the Euler equations as described in chapter 3, this solver will now be extended to general embedded mesh structures. The mesh structure considered is constructed through the combination of a coarse global mesh system, of one or more levels, and one or more local embedded mesh systems. The embedded mesh structures are really a continuation of the global mesh structure in local subdomains. Each additional level in the subdomain will be of the same grid topology as the coarse mesh, but of arbitrary shape and size as required to resolve the features of interest. The resulting solver for such general mesh structures represents the first step toward a

general modular approach.

The easiest way to illustrate the extension of the solver to general mesh structures is to follow the solver through a multiple-grid cycle for a simple model embedded grid structure. By following this process it will become clear what special problems must be considered for a proper formulation. Consider the addition of a local embedded mesh of half the mesh spacing h/2 into the global mesh as shown in figure 5-1. After renumbering the mesh levels, h/2 being level 1, h being level 2, and so on, it is noted that level 2 is now a coarse mesh within the embedded region and a fine mesh outside this region. It is desired to perform a control volume flux balance for all cells on the finest mesh in each region of the total domain in order that the fine mesh accuracy be obtained. However, it is also desired to couple the solution of the discrete equations throughout the total domain in order to achieve rapid convergence. The solution begins with a base solver sweep on level 1, which consists of only the embedded mesh region. Steps 1 to 3 of the basic solver are done for all cells in level 1. At this point the solution changes at all level 1 points internal to the embedded/global interface boundary are consistent with the changes for a standard Lax-Wendroff time step with the fine mesh accuracy. However, the changes for those points along the interface are incomplete since they lack the distribution of information from outside the interface. It is clear that a special treatment of the interface boundary points will be

required for a proper formulation of the solver. Noting this as a problem which will be addressed later, the base solver sweep is finished by updating the solution at all level 1 points. Proceeding to level 2, for those cells outside the embedded mesh region the base solver is used. For those cells within the embedded region the coarse mesh accelerator is used. Once again the treatment of the embedded/global interface points comes into question. In summary, on level 2 the solver must perform two functions, within the embedded mesh region it accelerates the convergence of the solution while outside this region it performs the standard base solver operations. For levels greater than level 2 it follows that the coarse mesh accelerator would be used everywhere. Beyond the basic framework just described, two special problems must be considered. One is the treatment of the boundary points already mentioned. In addition to the multiple-grid algorithm, the formulation of the solver at these points provides coupling of the embedded and global mesh solutions. The second question which arises is how the solver can be constructed to be independent of changes in the location, size, and shape of the embedded region. Key to this problem is how to determine whether the base solver, coarse mesh accelerator, and special interface formulation is required and what points are involved. With the solution of these two problems, the extension of the present approach to more than one embedded region or a progression of embedded meshes in a region is straight forward.

Points at the boundary must be carefully treated in order to maintain global conservation and computational stability. Consider the embedded mesh/global mesh interface shown in figure 5-2. A choice must be made as to whether points 1,6,2 are to be considered as members of the global grid or the embedded grid. That is, it must be decided as to whether the solution of the equations at these points is to be obtained to global or embedded grid accuracy. For the current work, the approach has been adopted that the boundary points are members of the global grid. The solution for points 1 and 2 is obtained on the level 2 sweep described above. Values at point 6, which are needed to compute the level 1 sweep, are obtained by linear interpolation from points 1 and 2. Linear interpolation is consistent with the trapezoidal integration used for the flux balances.

Treatment of the boundary cells proceeds as follows. Prior to the solution sweep on level 1, points such as 6 are initialized by linear interpolation from points 1 and 2. Steps 1 through 3 of the base solver are performed for all cells on level 1 including those bounded by points 1,6,2. Prior to the update sweep, all values of dU at boundary points between the embedded and global mesh are reset to zero (points 1,6,2). At this point the first step in the interface formulation is performed. For each of the level 2 cells along the embedded interface (such as the right hand part of figure 5-2) an order h/2 accurate flux balance is computed using all nine points of figure 5-2. This flux balance is defined as

follows,

$$(DU_5^{2h}) = \frac{h}{} \qquad (5.1a)$$

$$.5(Dt/DV)\{ [(F+F)(y-y)-(G+G)(x-x)]+[(F+F)(y-y)-(G+G)(x-x)]$$
$$\qquad 1\,6\;\;6\,1 \qquad\qquad 1\,6\;\;6\,1 \qquad\qquad 6\,2\;\;2\,6 \qquad\qquad 6\,2\;\;2\,6$$

$$-[(F+F)(y-y)-(G+G)(x-x)]-[(F+F)(y-y)-(G+G)(x-x)]$$
$$\qquad 3\,8\;\;3\,8 \qquad\qquad 3\,8\;\;3\,8 \qquad\qquad 8\,4\;\;8\,4 \qquad\qquad 8\,4\;\;8\,4$$

$$+[(G+G)(x-x)-(F+F)(y-y)]+[(G+G)(x-x)-(F+F)(y-y)]$$
$$\qquad 1\,9\;\;9\,1 \qquad\qquad 1\,9\;\;9\,1 \qquad\qquad 9\,4\;\;4\,9 \qquad\qquad 9\,4\;\;4\,9$$

$$-[(G+G)(x-x)-(F+F)(y-y)]-[(G+G)(x-x)-(F+F)(y-y)]\}$$
$$\qquad 2\,7\;\;7\,2 \qquad\qquad 2\,7\;\;7\,2 \qquad\qquad 7\,3\;\;3\,7 \qquad\qquad 7\,3\;\;3\,7$$

where

$$DV = -0.5[(x_5-x_1)(y_9-y_6)-(x_9-x_6)(y_5-y_1)] \qquad (5.1b)$$

$$-0.5[(x_7-x_6)(y_5-y_2)-(x_5-x_2)(y_7-y_6)]$$

$$-0.5[(x_3-x_5)(y_8-y_7)-(x_8-x_7)(y_3-y_5)]$$

$$-0.5[(x_8-x_9)(y_4-y_5)-(x_4-x_5)(y_8-y_9)]$$

This flux balance is then distributed to the level 2 interface points (such as points 1 and 2) using the distribution formula of equation 3.16. After the flux balance and distribution has been performed for all such cells the changes dU at these points (points 1 and 2) is saved in temporary storage for use in the level 2 sweep. Since the solution changes at points 1 and 2 are no longer zero, they are reset to zero completing the first step of the interface formulation. The level 1 sweep is then completed with the updating of the solution. As a result, no change of U has taken place at the boundary points.

After deletion of every other point on the embedded mesh, the level 2 solution outlined earlier proceeds except for boundary cells as shown in figure 5-2. The injected value from the fine mesh is used for coarse mesh accelerator updating of interior points such as 3 and 4. However the contribution to interface points 1 and 2 is determined by step 2 of the interface formulation rather than a coarse mesh distribution of the transferred change. Step 2 of the interface formulation is simply to recall the special changes saved in step 1 during the level 1 sweep. These distributed changes are added to the change at interface points 1 and 2. The net result of the two steps of the interface formulation is that an order h/2 flux balance has been performed for the level 2 cell using the solution at all nine points at the same time level. In fact, if the coarse mesh accelerator is switched off this formulation is essentially a patching method for coupling the embedded and global solutions. For all cells of level 2 outside the embedded region the standard base solver is used. This includes distribution to the level 2 interface points from outside the embedded mesh region. With completion of the level 2 sweep the change dU is interpolated back to the fine mesh and the solution is updated. For the present model problem of figure 5-1 all coarser levels use standard coarse mesh accelerator.

There remain two areas of the present formulation which must be clarified, the smoothing formulation and the transfer operator formulation near the interface. First recall that

the smoothing operator is only applied once for each point on the fine mesh for a global multiple-grid solver. Then since the present embedded mesh approach assumes that the interface points belong to the global mesh it is clear that the interface points should be smoothed on the global solver sweep. Therefore for the level 2 sweep smoothing is added into the distribution formula for points 1,2. This is also the only way the interface points can be smoothed normal to the boundary since on level 1 no information is known outside the embedded region. All other points are smoothed only on the finest level in each region.

The formulation of the transfer operator along the boundary presents a slightly different problem. If simple injection (eqn 3.9) is used no special operation is required for transfer of the change in cells along the boundary. However if a distribution type transfer operator (eqn. 3.13 plus 3.15) is used the solution changes at points 1 and 2 are required. Unfortunately since these points are updated on the level 2 sweep, they are not known at the time of transfer. The present approach has been to simply use the changes at point 5 (the cell center) in place of the unknown values at points 1 and 2 in the transfer operations. Essentially this approach results in a transfer operator which is somewhere between simple injection and a distribution type transfer operator but which is no worse than simple injection. Since this change is only used by the coarse mesh accelerator it has no effect on the level of approximation of the scheme,

although it may effect the rate of convergence.

The extension of the above procedure to multiple embedded domains and embedded regions with more than one level follows directly. The solution cycle always begins on the finest mesh. Under the above described formulation, the boundary of a h/4 mesh embedded in a h/2 mesh should be at least a distance h from the boundary of the h and h/2 mesh. Some changes in the formulation could remove this restriction. However, this is not an important restriction since the truncation error near an interface will be of the coarser h level order anyway.

Finally in order to implement the present embedded mesh formulation for general embedded mesh structures the solver was written in terms of the pointer system described in the previous chapter. Written in terms of this pointer system the definition of the grid structure is completely independent from the solver. The pointers for each cell completely define the operations which should be performed (i.e. base solver, coarse mesh accelerator, which nodes to smooth, etc.). The interface boundary pointer, also described in chapter 4, defines the location and orientation of internal mesh boundaries where the interface formulation must be applied.

5.1 SOLUTIONS WITH THE EMBEDDED MESH FORMULATION

The embedded mesh extension to Ni's multiple-grid method as presented in the preceding section provides a very flexible

structure in which resolution of local flow detail is possible while minimizing the storage and work required. In general the addition of a local embedded h/2 level mesh region results in very little change in the number of multiple-grid cycles over the solution on the h global mesh alone. The price for this higher resolution then only appears in the additional work performed within the embedded mesh. The following 2-D airfoil solutions demonstrate the performance and flexibility of the present formulation. Each of these cases has special flow details which must be resolved for a proper solution. They are often chosen for code comparisons (for example the first two were part of the GAMM workshop [49]). In each of the following solutions the far field vortex correction has been used with a mesh far field radius of 5 chords. Each of the following embedded mesh structures were created by first generating a global mesh and then removing the fine mesh cells in the global region. This mesh generation approach was used for the following two reasons. First, it provides the corresponding global fine mesh which can be run with basic multiple grid solver for accuracy comparisons. Secondly this approach required the least amount of mesh generator development. For production codes for use in design applications a much better approach to generation of such general mesh structure would be the formulation an interactive mesh generation routine. As envisioned the user would generate a coarse global mesh and then interactively define regions of desired mesh refinement.

To demonstrate the above embedded mesh formulation consider the NACA0012 test case of figures 3-13. Beginning with the global 65*17 mesh of figure 3-8, we now include embedded h/2 meshes around the leading and trailing edges of equal density to a 129*33 mesh as shown in figure 5-3. Note that since the embedded mesh was generated from the 129*33 mesh of figure 3-12 the grid resolution and node locations in the embedded region are the same. Using the embedded mesh formulation presented above with a total of 4 multiple-grid levels produced the solution shown in figure 5-4. Comparing this solution with the 129*33 and 65*17 global solutions (figures 3-10 and 3-13) shows that the embedded leading and trailing edge regions have resolved the same flow detail as the global 129*33 mesh (fig. 3-12). Total pressure loss for this case is generated by the smoothing of the rapid expansion around the leading edge and in the trailing edge region by the skewness of the mesh as discussed in chapter 3. Comparing the total pressure loss for the 129*33 mesh (fig. 3-13b) and the embedded mesh (fig. 5-4b) shows that the embedded mesh regions provide the same accuracy as the global mesh solution. The total pressure contour plots presented in figures 3-13d and 5-4d show that this is also true over the general region around the airfoil. It is important to note that there is no generation of total pressure losses at the embedded/global mesh interface boundaries as shown by figure 5-4d. Since the total pressure loss is very sensitive to errors in the solver formulation, this is a good indication that the interface

formulation is correct. A calculated lift coefficient for the embedded mesh solution of 0.331 is almost exactly the same as the global 129*33 mesh result.

The residuals presented for embedded mesh solutions are the average of the absolute value of $d(\rho u)/Dt$ for all points in the domain after the global level sweep. The global level was chosen for evaluation of the residual because this level includes all points in the domain. The spectral radius for all other levels have been found to be the same. Figure 5-5 presents the residual as defined above after each of the of the four levels of the embedded mesh solution. It is clear from this figure that while the absolute levels vary somewhat, the rates of convergence are the same. Figure 5-6 compares the embedded mesh residual for the global level with the convergence histories of global 129*33 solution and the global 65*17 solution. Note that the convergence rate is almost the same as the 65*17 global solution and twice as fast as the global 129*33 solution. Thus, we have gained the 129*33 mesh resolution with a convergence rate on the order of the 65*17 global solution. Since the total number of mesh points is much less than the number of global 129*33 mesh points, the work per cycle is also significantly reduced.

To illustrate the benefit of continued mesh refinement consider the NACA0012 embedded mesh structure of figure 5-3 but now include a second embedded h/4 mesh in the leading edge region as shown in figure 5-7. The solution for this double

embedded region is shown in figure 5-8. Table 5-1 compares the force coefficients for the 33*17 global mesh, 129*33 global mesh, single embedded mesh, and doubled embedded mesh solutions. As expected the higher resolution further reduced the total pressure loss and shows an improvement in the force coefficients. The convergence history on the global h mesh is similar to the previous embedded mesh case, with no loss in the rate of convergence with the addition of the new region embedded region as shown in figure 5-9.

Table 5-1: Comparison of Force Coefficients for NACA0012 Airfoil Using Different Mesh Structures (M = 0.63 and angle of attack of 2.0 degrees)

MESH STRUCTURE	C_L	C_D
Global Coarse (33*17) (figure 3-10)	0.326	0.0019
Embedded (figure 5-4)	0.331	0.0011
Global Fine (129*33) (figure 3-13)	0.328	0.0009
Double Embedded (figure 5-7)	0.334	0.0008

The second case considered is a NACA0012 airfoil at flow conditions of M = 0.85 and α= 1.0 deg.. This is a lifting case with strong shocks at 85% chord on the upper surface and 70% chord on the lower surface. The lift in this case is a strong function of the shock location making good shock resolution very important. The 65*17 global mesh and corresponding solution are shown in figures 3-27 and 3-30.

While both shocks are apparent in fig. 3-27 neither shock is very well defined due to the poor mesh resolution in the shock regions. Figure 5-10 shows the corresponding embedded mesh used for this case where four embedded regions have been added to resolve the leading and trailing edges and the two shock regions. Comparing the embedded mesh solution with 4 mesh levels as shown in figure 5-11 with the 65*17 global solution shows much better resolution of the two shocks and a reduction in the surface total pressure losses which occur in the expansion region around the leading edge. The embedded mesh solution also agrees very well with the 129*33 global solution of figure 3-30. Note in this example that the upper surface shock wave crosses the boundary of the embedded mesh region. Other than local loss of resolution (fig 5-11c and d) no difficulties are encountered. The convergence histories in terms of multiple-grid cycles for these three cases are compared in figure 5-12. Once again the embedded mesh solution provides the fine mesh resolution with a 65*17 global mesh convergence rate.

The next test case to be shown is the RAE2822 supercritical airfoil [49] at $M = 0.75$ and $\alpha = 3.0$ deg.. At these conditions there is a very rapid expansion around the leading edge and also a strong shock at 80% chord on the upper surface. The embedded mesh used is presented in fig. 5-13 using embedded regions around the leading and trailing edges and in the shock region. The corresponding embedded mesh solution is shown in fig. 5-14. Comparing this solution with

the 129*33 global mesh of figure 3-35 shown good resolution of both the leading edge and shock regions. The convergence rates for a global 65*17 mesh, global 129*33 mesh and the embedded mesh calculations are shown in fig. 5-15.

The final case to be presented is the Garabedian and Korn supercritical airfoil [51] with a design condition of M_∞=0.75 and α=0.12 deg. and a theoretical lift coefficient of 0.63. At design conditions the supersonic region extends over about 60% of the upper surface. The solution in this case is very sensitive to the location of the sonic line in the flow. The embedded mesh used for this case is shown in fig. 5-16. The corresponding embedded mesh solution is shown in fig. 5-17 with a lift coefficient of 0.607. Comparing the embedded mesh solution of figure 5-17 with the global 129*33 mesh solution of figure 3-40 clearly shows that the embedded meshes have resolved the flow features in the leading and trailing edge regions. This is also confirmed by the lift coefficient agreement between the two cases. Note however, that the drag coefficient for the embedded mesh solution (0.0033) is not the same as the 129*33 global mesh solution but falls midway between the values of the 65*17 mesh (0.0042) and the 129*33 mesh (0.0022) solutions. This poor agreement with the globally refined mesh is due to the fact that a majority of the supersonic region is being resolved with the global 65*17 mesh resolution, giving a much poorer resolution of the sonic line. The rise in total pressure error at the end of supersonic region (figure 5-17), which is similar to

that of the 65*17 mesh solution (figure 3-38b), is also due to the poor resolution of the sonic line. This total pressure error rise tends to suggest the formation of a very weak shock. The convergence histories the convergence histories for a global 65*17, global 129*33, and embedded mesh solutions are presented in figure 5-18.

CHAPTER 6

CONCLUSIONS

This work represents the first step in the development of a general modular approach to solving complex flow problems. In the current approach, the flow problem has been viewed in terms of a general multiple-grid structure consisting of a global mesh combined with one or more embedded mesh regions which provide local mesh refinement to resolve important flow features. Ni's method [14], a Lax-Wendroff type time marching scheme for multiple-grid solutions of the Euler Equations, has been extended to the solution of flows with general embedded mesh structures. While the present formulation uses a Lax-Wendroff type time marching scheme, the multiple-grid structure is a much more fundamental concept which need not be limited to this scheme. Adopting a multiple-grid formulation for embedded mesh calculations yields several important advantages over a simple patching approach. First, the coupling of the global and embedded mesh solutions takes place over the entire embedded mesh domain, rather than simply at the embedded mesh boundaries, resulting in accelerated convergence to steady state. Second, the

multiple grid structure provides a systematic way of describing general embedded mesh structures.

The main contributions of the present thesis are the following:

* Introduction of a pointer system for description of general embedded mesh structures.
* Formulation of a consistent treatment for embedded mesh regions in a multiple-grid formulation.
* Demonstration of the improved performance of the embedded mesh formulation.
* Improvements to the basic Ni scheme in the following areas:
  - Implementation of boundary conditions in characteristic form.
  - Development of a vortex far field boundary formulation for lifting airfoils.
  - Development of an improved transfer operator for the coarse mesh accelerator.

These will be summarized in the following paragraphs together with recommendations for future research.

One of the major problems in the formulation of a solver for general embedded mesh structures is the organization of the computational data base, comprised of the location and flow solution at node points. For conventional global solvers this data base organization is an integral part of the solver formulation, with the solver being written for a particular grid structure. While in principle this approach could be extended to embedded mesh calculations, this would result in a solver which must be rewritten for each new embedded mesh

structure. This thesis presents a much simpler and more flexible approach to this problem by constructing a pointer system which defines the organization of the computational data base for general embedded mesh structures. A general solver is then written in terms of this pointer system, making the solver independent from the organization of the grid structure.

Several different pointer system formulations have been considered, each constructed using a different base element (block of cells, line of cells, or single cell). Each pointer system has both advantages and drawbacks in terms of the storage required, flexibility in definition of grid structures and flexibility in the solver formulation. The pointer system chosen here uses the cell as the base element. Since the cell is the most fundamental element of general grid structures this pointer system provides the greatest flexibility in the type of grid structures which can be defined, allowing irregular embedded mesh regions, any number of embedded regions and multiple embedded regions. It is quite possible that for certain applications, such as vectorized algorithms, another pointer formulation might be more desirable. The addition of a pointer system does add to the total storage required per grid point of the system but for a proper distribution of mesh points, made possible by the general multiple-grid structure, the total number of grid points can now be minimized. This can then result in a significant reduction in storage over that required for an equivalent

global grid.

After adopting the pointer system, the extension of the global Ni scheme to embedded mesh structures is very straight forward. Embedded regions are viewed simply as a continuation of the global mesh in local regions. Of particular importance in this extension is the proper formulation of the algorithm at embedded/global interface boundaries. The formulation must both satisfy the conservation laws across these boundaries as well as resulting in a stable formulation. In the present formulation, points along these boundaries are viewed as part of the global mesh and therefore updated on the global mesh sweep. Conservation is satisfied across these boundaries by assuming a linear variation of the conservation variables along the global cell faces. This is consistent with the discrete finite volume approximation of the governing equations for any cell.

This embedded multiple-grid formulation has been demonstrated with the solution of the two-dimensional Euler equation for transonic flow over airfoils. In the cases presented, embedded mesh regions were used to resolve the flow features in the region of shocks, the leading edge stagnation point, and the trailing edge. Each of these cases used more than one embedded mesh region, with the location, size and shape of these regions being chosen to resolve the features of importance (see figures 5-3, 5-10, 5-13, and 5-16). The embedded regions need not be rectangular (in computational

space) as shown by figures 5-10 and 5-13. Multiple-embedded mesh regions are no more difficult to solve than a single embedded region, as demonstrated by the double embedded mesh used in the leading edge region of figure 5-7. These cases clearly demonstrate the great flexibility possible in the definition of general embedded mesh grid structures. For each case, the embedded mesh solution has been compared with the solutions for a global coarse mesh (the coarse global mesh without any embedded regions) and a global mesh refinement (the embedded mesh resolution over the complete domain). The algorithm formulation at embedded mesh boundaries does not generate errors in the flow (see figures 5-4 and 5-14). Comparing the embedded mesh solutions with the corresponding global refined mesh solutions demonstrates that the important flow features within the embedded mesh regions are accurately resolved (figures 3-13 and 5-4). This is also confirmed by the good force coefficient agreement between embedded and globally refined solutions. Even when a shock penetrates the embedded mesh boundary the only noticeable effect is the larger smearing of the shock outside the embedded mesh due to the lower mesh resolution (see figure 5-14). Comparing the convergence rates for global coarse, embedded, and global fine solutions, the embedded mesh solutions have been found to achieve the same convergence rates as the global coarse mesh solutions, which are much better than the global fine mesh solutions (see figures 5-6 and 5-15). In addition, since the computational work per multiple-grid cycle is proportional to

the number of mesh points used, the embedded mesh solutions require far less work per cycle than the equivalent fine mesh solution. Combining both the reduction in cycles required to converge with the lower computational work per cycle and a great reduction in computational work is possible by using embedded mesh structures. For the cases presented, the embedded mesh solutions have generally been 3-4 times faster than the equivalent global fine mesh solutions.

In summary the following conclusions may be made about the present embedded mesh formulation:

* Embedded mesh solutions always achieved the convergence rate of the coarse global grid, resulting in a significant reduction in work (a factor of 3-4 over the equivalent global fine mesh solution).

* The embedded mesh formulation in combination with the pointer system allows great flexibility in the definition of embedded mesh structures, including the capability of:

  - Any number of embedded regions.

  - Embedded regions of arbitrary shape and size.

  - Multiple embedded regions.

* Always achieved the virtually same accuracy as the globally refined mesh.

In addition to the formulation of a general embedded mesh approach to solving flow problems, the basic Ni's scheme for global multiple-grid solutions of the Euler equation has been studied and improved in the areas of boundary conditions and residual transfers. Of particular importance in the formulation of this scheme is the implementation of boundary

conditions. Unlike cell centered finite volume methods, the present scheme requires the calculation of the density and velocities in addition to the pressure along solid wall boundaries. A characteristic boundary condition formulation for solid wall boundaries has been employed to provide the additional information. As shown by the cases studied, this boundary formulation has been found to be quite accurate. The farfield boundary conditions are also implemented through a characteristic boundary condition formulation. By modeling the farfield flow in terms of the superposition of uniform flow with a compressible point vortex, whose strength is determined by the lift on the airfoil, the location of the farfield boundary can be placed much closer to the airfoil than that permitted by commonly used uniform flow boundary conditions (see table 4-3). This represents a large savings in the number of mesh points required for equivalent resolution of the flow about lifting airfoils.

Proper formulation of the residual transfer operator for multiple-grid solutions has been shown to be very important for highly stretched meshes. For moderately stretched grids simple injection works quite well but as the stretching increases the convergence rate for the solver deteriorates and finally fails altogether. Algebraic and area weighting transfer operators have been found to also fail for highly stretched meshes. A transfer operator which is based on the distribution formula of the base solver has been presented which corrects this problem. This distribution type transfer

operator has been found to work where all other formulations have failed (see table 3-1).

The effects of artificial smoothing on the solution of two-dimensional airfoils has also been studied. While the addition of smoothing causes the generation of total pressure errors in high gradient regions of the flow, as well as errors in the drag coefficient, it has little effect on the calculated lift coefficient (figures 3-15 and 3-22). For reasonable levels of smoothing these errors can be kept to an acceptable level. While required for transonic flows, smoothing has also been shown to greatly improve the convergence rate of the multiple-grid solver for all types of flows. For subsonic flows on O-type meshes smoothing is always required due to the mesh singularity at the trailing edge.

Based on the present results it is clear that there are many areas where future work is possible. In terms of the basic multiple-grid solver these areas are the following. First, the present smoothing formulation is only spatially first order accurate. Since smoothing is required and the algorithm itself is second order, a second order smoothing should be formulated which is consistent with the accuracy of the Ni algorithm. A switch to a first order smoothing (such as the present) could then be made in the region of shocks where the higher smoothing is required for stability. Such a duel smoothing formulation would make the algorithm second

order over a majority of the flow and therefore reduce the total pressure errors generated by the smoothing. Second, while some progress has been made in understanding how the coarse mesh accelerator works, there remains a great deal to learn. An in-depth study of the coarse mesh acceleration process would be helpful in formulation of special boundary conditions for the coarser levels as well as improvements in the basic accelerator formulation. A third area where some work has already been done is in the choice of the base solver used. Johnson [39] has demonstrated that the coarse mesh accelerator may be used with other base solvers. An in-depth study of these and other possible solvers should be made at this point to determine which are best for different applications. In particular while the present embedded mesh formulation is considered to be relatively independent of the base solver formulation, the choice of base solver may require different interface formulations from those given here. It is therefore important at this point to decide which solvers are the most promising before this approach progresses much farther.

The embedded mesh formulation in combination with the pointer system presented here opens the door to a whole range of future developments. These include the addition of embedded viscous regions as formulated in Appendix D, leading to an embedded equation approach as well as a embedded mesh approach. Viewing the multiple-grid structure in terms of the pointer system leads directly to adaptive solution problems

where routines are developed to manipulate the pointer structure during the solution process to add and remove embedded mesh regions as required. Since the mesh structure is completely defined by the pointer system, the solver algorithm would remain unchanged with changes in the grid. This adaptive mesh approach is currently being developed by Dannenhoffer and Baron [53]. The next step toward a general modular approach to solving complex flow problems will require the extension of the present embedded mesh approach to include embedded mesh regions where the embedded mesh is of different topology from the global mesh. This would allow the generation of simple, locally body fitted meshes around complex bodies. One possibility for such an approach is to consider patching meshes of different topologies together on the global level resulting in a single global mesh. The present embedded mesh procedure could then be used to resolve important flow features on such a mesh. Norton, Thompkins and Haimes [54] have demonstrated such a technique for turbine cascade calculations without embedded meshes.

REFERENCES

1. Rubbert, P.E. and Tinoco, E.N., "Impact of Computational Methods on Aircraft Design," AIAA Paper No. 83-2060, August 1983.

2. Henne, P., "Computational Aerodynamics-Application to Transport Aircraft Design," AIAA Paper No. 83-2061, August 1983.

3. Tunlinius, J.R., Bonner, E. and Shankar, V., "Impact of Computational Aerodynamics on Aircraft Design," AIAA Paper No. 83-2062, August 1983.

4. Bradley, R.G. and Bhateley, I.C., "Computational Aerodynamics Design of Fighter Aircraft Progress and Pitfalls," AIAA Paper No. 83-2063, August 1983.

5. Murman, E.M. and Cole, J.D., "Calculation of Plane Steady Transonic Flows," AIAA Journal, Vol. 9, 1971, pp 114-121.

6. South Jr., J.C. and Brandt, A. "Application of a Multi-level Grid Method to Transonic Flow Calculations," Transonic Flow Problems In Turbomachinery, Hemisphere Publishing Co., 1977, pp 180-207.

7. Ballhaus, W.F., Jameson, A. and Albert, J., "Implicit Approximate Factorization Schemes for the Efficient Solution of Steady Transonic Flow Problems," AIAA Paper No. 77-634, AIAA 3rd Computational Fluid Dynamics Conference, Albuquerque, NM, June 1977.

8. Boppe, C.W., "Computational Transonic Flow About Realistic Aircraft Configurations," AIAA Paper No. 78-104, January 1978.

9. Boppe, C.W., "Computational Transonic Flow about Realistic Aircraft Configurations," NASA CR 3243, May 1980.

10. Jameson, A., "Iterative Solution of Transonic Flows Over Airfoils and Wings, Including Flows at Mach 1," Comm. Pure Applied Math., Vol. 27, 1974, pp 283-309.

11. Caspar, J.R., Hobbs, D.E., and Davis, R.L., "The Calculation of Two-dimensional Compressible Potential Flow in Cascades Using Finite Area Techniques," AIAA Paper No. 79-0077, January 1979.

12. Jameson, A. and Caughey, D.A., "A Finite Volume Method for

Transonic Potential Flow Calculations," AIAA Paper No. 77-635, AIAA 3rd Computational Fluid Dynamics Conference Proceedings, June 1977.

13. Atta, E.H. and Vadyak, J., "A Grid Overlapping Scheme for Flowfield Computations About Multicomponent Configurations," AIAA Journal, Vol. 21, No. 9, September 1983, pp 1271-1277.

14. Ni, R.H., "A Multiple-Grid Scheme For Solving the Euler Equation," AIAA Journal, Vol. 20, No. 11, November 1982, pp 1565-1571.

15. Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solutions to the Euler Equation by Finite Volume Methods Using Runge-Kutta Time Stepping," AIAA Paper 81-1259, June 1981.

16. Jameson, A., Schmidt, W. and Whitfield, D., "Finite Volume Solution for the Euler Equations for Transonic Flow Over Airfoils and Wings Including Viscous Effects," AIAA Paper No. 81-1265, June 1981.

17. Rizzi, A., "Damped Euler Equation Algorithms to Compute Transonic Flow Around Wing-Body Configurations," AIAA Journal, Vol. 20, No. 10, October 1982, pp 1321-1328.

18. Jameson, A., "Solution of the Euler Equation for Two Dimensional Transonic Flow by a Multigrid Method," Princeton University, MAE Report No. 1613, June 1983.

19. Eriksson, L.E., "Generation of Boundary-Conforming Grids Around Wing-Body Configurations Using Transfinite Interpolation," AIAA Journal, Vol. 20, No. 10, October 1982, pp 1313-1320.

20. Jameson, A. and Baker, T.J., "Multigrid Solution of the Euler Equations for Aircraft Configurations," AIAA Paper No. 84-0093, January 1984.

21. Atta, E.H., "Component-Adaptive Grid Embedding," NASA CP2166, October 1980, pp 157-174.

22. Forester, C.K., "Body-Fitted 3-D Full-Potential Flow Analysis of Complex Ducts and Inlets," AIAA Paper No. 81-0002, January 1981.

23. Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," Mathematics of Computation, Vol. 31, No. 138, April 1977, pp 333-390.

24. Brandt, A., "Multi-Level Adaptive Computations in Fluid Dynamics," AIAA Journal, Vol. 18, No. 10, October 1980, pp 1165-1172.

25. Brown, J.J., "A Multigrid Mesh-Embedding Technique for Three-Dimensional Transonic Potential Flow Analysis," NASA CP 2202, October 1981.

26. Stienhoff, J. and Jameson, A., "Multiple Solutions of the Transonic Potential Flow Equation," AIAA Journal, Vol. 20, No. 11, November 1982, pp 1521-1525.

27. Salas, M.D., Jameson, A. and Melnik, R.E., "A Comparative Study of the Nonuniqueness Problem of the Potential Equation," AIAA Paper No. 83-1888, AIAA Computational Fluid Dynamics Conference Proceedings, Danvers, MA, July 1983.

28. Johnson, G.M., "Convergence Acceleration of Viscous Flow Computation," NASA TM 83039, October 1982.

29. Chima, R.V. and Johnson, G.M., "Efficient Solution of the Euler and Navier-Stokes Equations With a Vectorized Multiple-Grid Algorithm," AIAA Paper No. 83-1893, AIAA Computational Fluid Dymanics Conference Proceedings, Danvers, MA, July 1983.

30. Davis, R.L., "The Prediction of Compressible, Laminar Viscous Flows Using a Time-Marching Control Volume and Multigrid Technique," AIAA Paper No. 83-1896, AIAA Computational Fluid Dymanics Conference Proceedings, Danvers, MA, July 1983.

31. Hindman, R.G., "Generalized Coordinate Forms of Governing Fluid Equations and Associated Geometrically Induced Errors," AIAA Journal, Vol. 20, No. 10, October 1982, pp 1359-1367.

32. Viviand, H., "Conservation Forms of Gas Dynamic Equations," Recherche Aerospatiale, No. 1, January 1974, pp 65-68.

33. Warsi, Z.U.A., "Conservation Form of the Navier-Stokes Equations in General Nonsteady Coordinates," AIAA Journal, Vol. 19, No. 2, February 1981, pp 240-242.

34. Thompkins Jr., W.T., Tong, S.S., Bush, R.H., Usab Jr., W.J., and Norton, R.J.G., "Solution Procedures for Accurate Numerical Simulations of Flow in Turbomachinery Cascades," AIAA Paper No. 83-0257, January 1983.

35. Cebeci, T. and Smith, A.M.O., Analysis of Turbulent Boundary Layers, Academic Press, New York, 1974.

36. Baldwin, B.S. and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper No. 78-257, January 1978.

37. Whitfield, D.L., Thomas, J.L., Jameson, A., and Schmidt, W., "Computation of Transonic Viscous-Inviscid Interacting Flow," Proceedings of Second Symposium of Numerical and Physical Aspects of Aerodynamic Flows, California State University, January 1983.

38. Ludford, G.S.S., "The Behavior at Infinity of the Potential Function of a Two Dimensional Subsonic Compressible Flow," Journal Math. Physics, Vol. 30, 1951, pp 117-130.

39. Johnson, G.M., "Multiple-grid Acceleration of Lax-Wendroff Algorithms," NASA TM 82843, March 1982.

40. Dannenhoffer III, J., private communication.

41. Brandt, A., "Multi-Level Adaptive Techniques (MLAT), I. the Multi-grid Method," IBM Research Report RC 6026, June 1976.

42. Ni, R.H., private communication.

43. Abbett, M.J., "Boundary Condition Calculation Procedures for Inviscid Supersonic Flow Fields," AIAA Computational Fluid Dynamics Conference Proceedings, Palm Springs, California, July 19-20, 1973.

44. McCartin, B., "Theory, Computation, and Application of Exponential Splines," Courant Mathematics and Computing Laboratory, U.S. Dept. of Energy Report No. DOE/ER/03077-171, October 1981.

45. Eriksson, L.E. and Rizzi, A., "Computation of Vortex Flow Around Wings Using the Euler Equations," Proceedings of the IVth GAMM Conference on Numerical Methods in Fluid Mechanics, Paris, October 1981, Vieweg Verlag.

46. Loyd, B., "Calculating C-Grids with Fine and Embedded Mesh Regions," M.I.T. CFDL TR83-7, December 1983.

47. Lock, R.C., "Test Cases for Numerical Methods in Two-Dimensional Transonic Flows," AGARD Report No. 575, November 1970.

48. Roache, P.J., Computational Fluid Dynamics, Hermosa Publishers, Albuquerque, N.M., pg 249.

49. Rizzi, A., "Numerical Methods for the Computation of Invisid Transonic Flows with Shock Waves," A GAMM Workshop, Vieweg und Sohn, Wiesbaden, 1981.

50. Lerat, A. and Sides, J., "A New Finite Volume Method for the Euler Equations with Applications to Transonic Flows," Numerical Methods in Fluid Dynamics, edited by P.L. Roe,

Academic Press, New York, NY, 1982, pp 245-288.

51. Kacprzynski, J.J. and Ohman, L.H., "Analysis of the Flow Past a Shockless Lifting Airfoil in Design and Off-Design Conditions," National Research Council of Canada, Aeronautical Report LR-554, November 1971.

52. Jameson, A., "Transonic Flow Calculations," Numerical Methods in Fluid Dynamics, edited by H.J. Wirz and J.J. Smolderen, Hemisphere Publishing Corporation, Washington DC, 1978, ppl-88.

53. Dannenhoffer III,J.F. and Baron, J.R., "Adaptive Solution Procedure for Steady State Solution of Hyperbolic Equations," AIAA Paper No. 84-0005, January 1984.

54. Norton, R.J.G., Thompkins Jr., W.T. and Haimes, R., "Implicit Finite-Difference Scheme With Non-Simply Connected Grids... A Novel Approach," AIAA Paper No. 84-0003, January 1984.

55. Abarbanel, S.S., private communication.

APPENDIX A

NON-DIMENSIONALIZATION OF THE GOVERNING EQUATIONS

The two-dimensional Navier-Stokes equations for unsteady compressible laminar flow may be expressed in conservation form for a cartesian coordinate system as

$$U_t + F_x + G_y + R_x + S_y = 0 \qquad \text{(A.1a)}$$

where

$$U = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ e \end{vmatrix} \qquad F = \begin{vmatrix} \rho u \\ \rho uu + p \\ \rho uv \\ \rho uH \end{vmatrix} \qquad G = \begin{vmatrix} \rho v \\ \rho uv \\ \rho vv + p \\ \rho vH \end{vmatrix} \qquad \text{(A.1b)}$$

$$R = \begin{vmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xx} u + \tau_{yx} v - (\mu/(\gamma-1)Pr)\, T_x \end{vmatrix}$$

$$S = \begin{vmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yy} v + \tau_{xy} u - (\mu/(\gamma-1)Pr)\, T_y \end{vmatrix}$$

and where

$$\begin{aligned} \tau_{xx} &= -\mu[\ (4/3)u_x - (2/3)v_y\ ] \\ \tau_{yy} &= -\mu[\ (4/3)v_y - (2/3)u_x\ ] \\ \tau_{xy} &= \tau_{yx} = -\mu[\ u_y + v_x\ ] \end{aligned} \qquad \text{(A.1c)}$$

in terms of density ρ, cartesian (x,y) velocity components (u,v), temperature T, total internal energy per unit volume e, viscosity coefficient μ, and Prandtl number Pr. The pressure p and total enthalpy H are then defined for a perfect gas as

$$p = (\gamma - 1)[\ e - 0.5\rho(\ uu + vv\ )] \qquad \text{(A.2)}$$

$$H = (\ e + p\ )/\rho$$

where γ is the ratio of specific heats.

In addition, the viscosity μ is defined by Sutherlands law, an emperical relation describing the viscosity μ as a function of temperature T is given as

$$\mu = \mu_o (T/T_o)^{1.5} (\ T + 110^{\circ}K\ )/(\ T + 100^{\circ}K\ ) \qquad \text{(A.3)}$$

where μ_o is a reference viscosity and T_o is the reference temperature.

The following reference quantities have been chosen to non-dimensionalize the governing equation:

l_o the reference length

ρ_o the reference density

T_o the reference temperature

a_o the reference speed of sound

μ_o the reference viscosity

Note that by choosing a_o as the reference speed of sound, rather than simply a reference speed, the following relation between a_o and T_o results.

$$a_o = \sqrt{\gamma R T_o} \tag{A.4}$$

The above reference quantities are then used to define a set of non-dimensional variables, which will be denoted here as primed variables.

$$x' = x/l_o \qquad y' = y/l_o \qquad u' = u/a_o \qquad v' = v/a_o \tag{A.5}$$

$$\rho' = \rho/\rho_o \qquad T' = T/T_o \qquad \mu' = \mu/\mu_o \qquad p' = p/(\rho_o a_o^2)$$

$$H' = H/a_o^2 \qquad e' = e/(\rho_o a_o^2)$$

Substituting these non-dimensional variables into equations (A.1-A.3) and dropping the prime notation results in the equations presented in chapter 2 (eqn. 2.1-2.3). Note that with this particular scaling the non-dimensional expressions for U, F, and G are identical to the original dimensional expressions. Therefore, with this scaling the non-dimensional Euler equations are the same as the dimensional Euler equations. For the full Navier-Stokes equations the only difference between dimensional and non-dimensional forms is

the appearance of the reference Reynolds number Re_o in R and S. This reference Reynolds number is defined as follows,

$$Re_o = \rho_o a_o l_o / \mu_o \qquad (A.6)$$

This non-dimensionalization also yields the following useful relations:

$$\begin{aligned} p' &= \rho' T'/\gamma \quad \text{(equation of state)} \\ a' &= \sqrt{T'} \\ M &= M' = u'/a' \end{aligned} \qquad (A.7)$$

Throughout the present work the freestream stagnation conditions have been used to determine the reference quantities of equation A.4. The advantage of using stagnation conditions over the many other possible reference conditions is that the non-dimensional stagnation quantities reduce to

$$p'_T = 1/\gamma \qquad \rho'_T = 1 \qquad T'_T = 1 \qquad (A.8)$$

and therefore, are independent of the actual flow conditions.

APPENDIX B

STABILITY ANALYSIS OF THE 2-D WAVE EQUATION

An important step in both the development of new algorithms and the application of existing algorithms is performing a stability analysis of the chosen scheme. Such an analysis determines the stability limit for the scheme, from which the time step restriction for a stable solution is defined. Even for well established schemes a stability analysis can provide important insight into how and why an algorithm performs as it does.

To gain such an understanding of the Ni scheme a Von Neuman stability analysis has been performed for the 2-D scalar wave equation. The wave equation was chosen over the Euler equations for this analysis for the following reasons. The wave equation is of the same form as the Euler equations but since it is a scalar equation, rather than a system of equations, the analysis is much easier to perform. In addition since the Euler equations are nonlinear they must be linearized for such an analysis to be possible. This adds a further level of complexity without any additional insight into the algorithm. Finally, as will be shown in appendix C,

the stability limit for the linearized Euler equations can be inferred directly from the results of the present analysis.

The 2-D scalar wave equation may be expressed for a Cartesian system as

$$U_t + aU_x + bU_y = 0 \qquad \text{(B.1)}$$

where a and b are constants. Now consider a discrete approximation of equation B.1 on a uniform mesh of constant mesh spacing Dx and Dy, as shown in figure B-1, using the base solver discribed in chapter 3. Note for the present analysis the conventional (i,j) node indexing has been used. Performing the flux balance (eqn. 3.1) and distribution (eqn. 3.2) steps results in the following expression for the total change at point (i,j) at time step n

$$dU_{i,j} = U^{n+1}_{i,j} - U^{n}_{i,j} \qquad \text{(B.2)}$$

$$= C_x \left[U^{n}_{i-1,j-1} + 2U^{n}_{i-1,j} + U^{n}_{i-1,j+1} - U^{n}_{i+1,j+1} - 2U^{n}_{i+1,j} - U^{n}_{i+1,j-1}\right]/8$$

$$+ C_y \left[U^{n}_{i+1,j-1} + 2U^{n}_{i,j-1} + U^{n}_{i-1,j-1} - U^{n}_{i+1,j+1} - 2U^{n}_{i,j+1} - U^{n}_{i-1,j+1}\right]/8$$

$$
\begin{aligned}
&+C_x^2\,[\,U^{n}_{i-1,j-1} + 2U^{n}_{i-1,j} + U^{n}_{i-1,j+1} + U^{n}_{i+1,j-1} + 2U^{n}_{i+1,j} + U^{n}_{i+1,j+1} \\
&\qquad -2(U^{n}_{i,j-1} + 2U^{n}_{i,j} + U^{n}_{i,j+1})\,]/8 \\
&+C_x C_y\,[\,U^{n}_{i+1,j+1} + U^{n}_{i-1,j-1} - U^{n}_{i+1,j-1} - U^{n}_{i-1,j+1}\,]/4 \\
&+C_y^2\,[\,U^{n}_{i-1,j-1} + 2U^{n}_{i,j-1} + U^{n}_{i+1,j-1} + U^{n}_{i-1,j+1} + 2U^{n}_{i,j+1} + U^{n}_{i+1,j+1} \\
&\qquad -2(U^{n}_{i-1,j} + 2U^{n}_{i,j} + U^{n}_{i+1,j})\,]/8
\end{aligned}
$$

where the CFL numbers in the two coordinate directions are defined as

$$C_x = a\Delta t/\Delta x \qquad C_y = b\Delta t/\Delta y \tag{B.3}$$

If the solution U at point (i,j) and time level n is now assumed to be of the form

$$U^{n}_{i,j} = r^{n}\exp\{I[k_x i\Delta x + k_y j\Delta y]\} \tag{B.4}$$

then the amplification factor G is defined as

$$G = r^{n+1} / r^{n} \tag{B.5}$$

Substituting Equation B.4 into the discrete scalar wave equation of B.2 and rearranging gives the following expression for the amplification factor G.

$$G = 1 - 0.5I[C_x \sin\alpha(1+\cos\beta) + C_y \sin\beta(1+\cos\alpha)]$$

$$+0.5C_x^2(\cos\alpha-1)(\cos\beta+1) - C_x C_y \sin\alpha \sin\beta$$

$$+0.5C_y^2(\cos\beta-1)(\cos\alpha+1) \qquad \text{(B.6)}$$

where

$$\alpha = ik_x Dx \qquad \beta = jk_y Dy$$

For stability the magnitude of G must be less than or equal to one for all values of α and β. That is

$$|G|^2 \leq 1 \qquad \text{for } 0 \leq \alpha \leq 2\pi, \; 0 \leq \beta \leq 2\pi \qquad \text{(B.7)}$$

This inequality determines the relation and range of C_x and C_y for which the solution scheme is stable. To this point in time the author has been unable to find a closed form solution for C_x and C_y which satisfies this inequality. In view of this two alternative approached have been taken. First is to constrain C_x and C_y to certain values which simplify the inequality to the point where it can be solved. While this approach gives some indication of the stabilities boundaries is does not give a full picture. The second approach which has been taken is to numerically map out the stability boundary by evaluating the inequality over a large range of values of C_x and C_y.

In the first approach there are three special cases for which an analytic solution is possible. The first two are the trivial 1-D limits corresponding to the two Cartesian

coordinates x and y.

$$\text{If } C_y = 0 \text{ then } C_x \leq 1 \qquad \text{(B.8)}$$

$$\text{If } C_x = 0 \text{ then } C_y \leq 1$$

The third special was found with the help of Abarbanel [55]. Assuming $C_x = C_y = C$ equation B.6 may be rewritten as

$$G = 1-2C^2\left[\xi\sqrt{1-\eta^2}+\eta\sqrt{1-\xi^2}\right]^2 - I2C\left[\xi\sqrt{1-\eta^2}+\eta\sqrt{1-\xi^2}\right]\sqrt{1-\xi^2}\sqrt{1-\eta^2} \qquad \text{(B.9)}$$

where $\xi = \sin(\alpha/2)$ and $\eta = \sin(\beta/2)$

then for stability the following inequality must be satisfied.

$$1 \geq |G|^2 = 1-4C^2\left[\xi\sqrt{1-\eta^2}+\eta\sqrt{1-\xi^2}\right]^2+4C^4\left[\xi\sqrt{1-\eta^2}+\eta\sqrt{1-\xi^2}\right]^4$$
$$+4C^2\left[\xi\sqrt{1-\eta^2}+\eta\sqrt{1-\xi^2}\right](1-\xi^2)(1-\eta^2) \qquad \text{(B.10)}$$

$$\text{If } \left[\xi\sqrt{1-\eta^2}+\eta\sqrt{1-\xi^2}\right]^2 \neq 0 \quad (\text{ie. } \xi \neq 0,\ \eta \neq 0)$$

then after simplification

$$(1-2C^2)\xi^2\eta^2 - (1-C^2)(\xi^2+\eta^2) \leq -2C^2\xi\eta\sqrt{1-\xi^2}\sqrt{1-\eta^2} \qquad \text{(B.11)}$$

For $C \leq 1$ the most restrictive case is when $\xi\eta > 0$. Finally, squaring both sides

$$0 \leq (1-4C^2)\xi^4\eta^4 - (2-6C^2)\xi^2\eta^2(\xi^2+\eta^2) + (2-4C^2-2C^4)\xi^2\eta^2$$
$$+ (1-2C^2+C^4)(\xi^4+\eta^4) \qquad \text{(B.12)}$$

The above inequality is satisfied for all values of ξ and η if the follow stability criterion for C is met

$$C \leq 1/\sqrt{2} \qquad \text{(B.13)}$$

To prove this requires the following two steps. First direct

substitution of $C=1/\sqrt{2}$ shows that the inequatity is satisfied. Second, by demonstrating that if $C>1/\sqrt{2}$ this inequality is violated for some combination of ξ and η, then the above limit is proven. By substitution of $\xi = \eta = 0$ the second point is show. In summary, the most restrictive condition shown analytically is that of eqn B.13. Based on this, the following stability criterion results,

$$C_x = C_y \leq 1/\sqrt{2} \qquad \text{(B.14)}$$

While special analytical solutions with constrained values of C_x and C_y give some indication of the stability limit for the Ni scheme, this approach does not define the stability boundary completely. An alternate method, which is often used in cases such as this, is to solve the inequality of B.6 numerically. This involves testing the inequality, with fixed values of C_x and C_y, for all α and β. By repeating this process for a large number of combinations of (C_x, C_y) values the complete stability boundary can be mapped out. The disadvantage of this approach is that it doesn't result in a closed form solution defining the boundary as a function of C_x and C_y. In addition such calculations involve a large amount of computing time to be done accurately.

A numerical analysis has been performed for the inequality of equation B.6. The results are presented graphically in figure B-2. The line in the figure is the locus of points (C_x, C_y) for which $|G|=1$. The inequality is satisfied and therefore, the scheme is stable for all points

inside this line. As expected the three analytical solutions agree with the numerical solution. While it can't be shown analytically, Figure B-2 leads one to believe that the stability boundary for the 2-D wave equation is a circle in the (C_x, C_y) plane. This results in the following general stability criterion for the Ni scheme,

$$\left(C_x^2 + C_y^2 \right) \leq 1 \qquad \text{(B.15)}$$

APPENDIX C

CFL NUMBERS FOR THE EULER EQUATIONS

The Von Neuman stability analysis of the Ni scheme for the 2-D scalar wave equation presented in Appendix B leads to a time step restriction expressed in terms of the CFL numbers in the two Cartesian coordinate directions. Unfortunately it is not possible to perform the same analysis for the 2-D Euler equations (eqns. 2.9) since they are a nonlinear system of equations. To gain some insight into the stability of the present Euler solver, the Euler equations must first be linearized. Once linearized the preceding Von Neuman analysis can again be performed. Since the linearized form of the Euler equations is the same as the scalar 2-D wave equation (the only difference being a system of equations rather than a single equation), it is much easier to simply relate the wave equation results directly to the present system of equations. The key to extending the wave equation analysis to the present system of equations is the proper definition of the CFL numbers coresponding to the two computational coordinate directions. The derivation which follows will determine these numbers.

The non-orthogonal form of the 2-D Euler equations (eqn 2.9) may be expressed on quasilinear form through the chain rule as

$$(U/J)_t + A\,(U/J)_\xi + B\,(U/J)_\eta = 0 \qquad \text{(C.1a)}$$

where the Jacobian matrix A is defined as

$$\begin{aligned} A &= \left[\, y_\eta F - x_\eta G \,\right]_{(U/J)} \\ &= \left[\, U \,\right]_{(U/J)} \left[\, y_\eta F - x_\eta G \,\right]_U \\ &= J \left[\, y_\eta F - x_\eta G \,\right]_U \\ &= J\, y_\eta F_U - J\, x_\eta G_U \end{aligned} \qquad \text{(C.1b)}$$

and similarly,

$$B = -J\, y_\xi F_U + J\, x_\xi G_U \qquad \text{(C.1c)}$$

$(\partial F/\partial U)$ and $(\partial G/\partial U)$ are the Jacobian matrices of the quasilinear form of the Euler equations expressed in Cartesian coordinates. While the form of equation C.1 resembles that of the scalar wave equation, this system of equations is still nonlinear since both A and B are functions of U.

Equation C.1 is linearized by freezing the values of matrices A and B. Once frozen, the equations reduce to a set of four constant coefficient linear equations of the same form as the scalar wave equation. Recall that for the scalar wave equation, the CFL numbers corresponding to Cartesian coordinate directions x and y are

$$CFL_x = Dt|a|/Dx \qquad (C.2)$$

$$CFL_y = Dt|b|/Dy$$

where a and b are constants which determine the characteristic propagation speeds in respective coordinate directions as

$$Dt/Dx = 1/a \qquad Dt/Dy = 1/b \qquad (C.3)$$

The corresponding characteristic propagation speeds for a linear system of equations, such as equation C.1, are determined by the eigenvalues of the coefficient matrices, A and B. For the present system then

$$Dt/D\xi = 1/\lambda_A \qquad Dt/D\eta = 1/\lambda_B \qquad (C.4)$$

where λ_A and λ_B are the eigenvalues of A and B, respectively. Note that since A and B are 4*4 matrices, there are 4 eigenvalues for each matrix and therefore, 4 propagation speeds for each. The CFL number corresponding to each coordinate direction is determined by the maximum propagation speed or eigenvalue of the respective Jacobian matrix.

$$CFL_\xi = Dt\,|(\lambda_A)_{max}|/D\xi \qquad (C.5)$$

$$CFL_\eta = Dt\,|(\lambda_B)_{max}|/D\eta$$

To complete the above expressions the actual eigenvalues of A and B must be determined. Noting that both A and B are of the form

$$P = k_1 F_U + k_2 G_U \tag{C.6}$$

where k_1 and k_2 are constants, then if the eigenvalues of matrix P are determined once in terms of these constants, the eigenvalues of both A and B can be found through substitution. While the derivation of these eigenvalues is not difficult, it is extremely long and tedious, and therefore will not be presented here. Rather, the reader is referred to a very clear derivation of the eigenvalues of P presented by McCartin [11]. The eigenvalues of matrix P are

$$\lambda_1 = \lambda_2 = k_1 u + k_2 v \tag{C.7}$$

$$\lambda_3 = k_1 u + k_2 v + c (k_1^2 + k_2^2)^{0.5}$$

$$\lambda_4 = k_1 u + k_2 v - c (k_1^2 + k_2^2)^{0.5}$$

where u and v are Cartesian velocity components and c is the speed of sound. Substituting the correct values of k_1 and k_2 for Jacobian matrices A and B into equation C.7 gives the following maximum eigenvalues,

$$|(\lambda)_{A\,max}| = J [| y_\eta u - x_\eta v | + c (x_\eta^2 + y_\eta^2)^{0.5}]$$

$$|(\lambda)_{B\,max}| = J [| y_\xi u - x_\xi v | + c (x_\xi^2 + y_\xi^2)^{0.5}] \tag{C.8}$$

With substitution of these expressions into C.5 the CFL numbers are

$$CFL_\xi = J\,Dt\,[\,|\,y_\eta\,u - x_\eta\,v\,|\,+\,c\,(\,x_\eta^2 + y_\eta^2\,)^{0.5}\,]\,/\,D\xi$$
$$CFL_\eta = J\,Dt\,[\,|\,y_\xi\,u - x_\xi\,v\,|\,+\,c\,(\,x_\xi^2 + y_\xi^2\,)^{0.5}\,]\,/\,D\eta \qquad (C.9)$$

Finally, the metrics ($x_\xi, y_\xi, x_\eta, y_\eta$) and Jacobian J must be evaluated for a typical cell, such as shown in figure 3-2. Using second order accurate, cell centered differences then

$$x_\eta = Dx^l / D\eta \qquad y_\eta = Dy^l / D\eta$$
$$x_\xi = Dx^m / D\xi \qquad y_\xi = Dy^m / D\xi \qquad (C.10)$$
$$J = D\xi\,D\eta / DV$$

where Dx^l, Dy^l, Dx^m, Dy^m, and DV are defined by equations 3.2. This choice of differencing is consistent with the differencing used for the higher order time terms of the presented Ni scheme. It also follows through equations 3.3 and C.10 that

$$(\,x_\eta^2 + y_\eta^2\,)^{0.5} = Dl / D\eta \qquad (C.11)$$
$$(\,x_\xi^2 + y_\xi^2\,)^{0.5} = Dm / D\xi$$

Substituting the metric definitions into C.9 then gives

$$CFL_\xi = \frac{Dt[\,|uDy^l - vDx^l|\,+aDl\,]}{DV} \qquad CFL_\eta = \frac{Dt[\,|uDy^m - vDx^m|\,+aDm\,]}{DV} \qquad (C.12)$$

The above expressions define the CFL numbers corresponding to

non-orthogonal directions ξ and η for a typical cell of figure 3-2.

APPENDIX D

EXTENSION TO THE NAVIER-STOKES EQUATIONS

For high Reynold's number flows viscosity plays an important role only in thin layers near the body and small wake regions. A majority of the flow is inviscid. Viewing the Navier-Stokes equations as a combination of convective terms which propagate information and viscous terms which smooth information, it is clear that a majority of the flow is dominated by the convective terms.

An extension of the Ni multiple-grid method seems well suited as a solver for these flows since it takes advantage of the convective nature of the equations to accelerate the solution convergence to steady state. With the Ni scheme the two parts of the solver, base solver and coarse mesh accelerator, perform two different roles. The base solver satisfies the physics of the problem by solving the governing equations. The coarse mesh accelerator then models the convective terms to rapidly propagate errors in the solution out of the flow field for fast convergence. In addition, as discussed in the conclusions, one of the next steps in the direction of a general modular approach to solving transonic

flow problems is the addition of embedded viscous regions into the embedded multiple-grid approach. With the formulation of a viscous version of Ni's Multiple-grid method such embedded equation solutions would be possible.

In this appendix a formulation of the Ni multiple-grid method is presented for solution of the 2-D Navier-Stokes equations. In this formulation a global mesh has been assumed, removing the need for a special embedded mesh boundary formulation. This should not be viewed as a limitation, rather the embedded mesh formulation has been left for future research. The approach which has been taken is to include the viscous terms into the base solver. With this modification the base solver correctly models the physics for viscous flows. The inviscid coarse mesh accelerator, as presented in section 3.3, is then used to propagate the fine mesh residuals accelerating the convergence to steady state. Since the convective terms dominate over a majority of the flow field, a coarse mesh accelerator based only on the convective terms captures the principle physics of the flow and will efficiently propagate the solution errors.

The present formulation was initially tested with the calculation of 2-D laminar flow in a duct with a circular arc bump on one wall. While these calculations were made on a very coarse mesh, resulting in unrealistic cell Reynolds numbers, and for a flow field which can not be considered very severe, the approach showed a great deal of promise as a

method for accelerated solutions of the Navier-Stokes equations. Chima and Johnson [29] independently adopted a similar approach, using a fine mesh solver based on MacCormack's method combined with the Ni coarse mesh accelerator, and also demonstrated accelerated convergence rates for duct type flows. Based on the success with duct type flows, preliminary calculations with the present formulation were made for laminar flow over 2-D airfoils. Unfortunately, these calculations proved to be much more difficult that initially expected. As formulated, the present approach failed to yield the expected acceleration of solution convergence. Repeated attempts to correct the present formulation failed to improve the rate of convergence. At this point due to time limitations, the current research had to be brought to a close, leaving the extension of Ni's multiple-grid method to viscous flows as an area of future research. In the author's opinion, this breakdown in the formulation can be corrected and therefore, the current approach will be presented as originally formulated. In the paragraphs which follow the base solver presented in section 3.2 will be extended to the solution of the 2-D Navier-Stokes equations.

The two dimensional Navier-Stokes equations were presented in strong conservation form (equation 2.1) as

$$U_t + F_x + G_y + R_x + S_y = 0 \qquad (D.1)$$

The time differencing proposed for solution of this equation is a combination of a Lax-Wendroff differencing of the inviscid terms (terms containing F and G) and a forward time-center space (FTCS) differencing of the dissipative terms (those containing R and S).

$$dU^{n}_{i,j} = -(Dt/DV)(F_x + G_y)^{n}_{i,j} \qquad (D.2)$$

$$+0.5(Dt^2/DV)\{[F_U (F_x + G_y)_x] + [G_U (F_x + G_y)_y]\}^{n}_{i,j}$$

$$-(Dt/DV)(R_x + S_y)^{n}_{i,j}$$

While the differencing of the inviscid terms is second order accurate in time, this semi-discrete equation is only first order accurate in time due to the FTCS differencing of the dissipative terms. Since only the steady state solution is of interest this lower accuracy is of little importance. The discrete spatial approximation used for the first two terms on the right hand side of equation D.2 is the same as that presented in equations 3.1 and 3.2, the flux balance and distribution steps for the Euler equations. The correct discrete approximation of the last term is found through a finite volume integration over the cell shown with a dashed line in figure D-1.

$$-(Dt/DV)\int_V (R_x + S_y)\,dV = .5(Dt/DV)\{ [Dy^{l}_{A} R^{l}_{A} - Dx^{l}_{A} S^{l}_{A}] + [Dx^{m}_{A} S^{m}_{A} - Dy^{m}_{A} R^{m}_{A}] \}$$

$$+.5(Dt/DV)\{ [Dy^{l}_{B} R^{l}_{B} - Dx^{l}_{B} S^{l}_{B}] - [Dx^{m}_{B} S^{m}_{B} - Dy^{m}_{B} R^{m}_{B}] \}$$

(D.3)

$$+.5(Dt/DV)\{-[Dy^{l}_{C}R_{C} - Dx^{l}_{C}S_{C}] - [Dx^{m}_{C}S_{C} - Dy^{m}_{C}R_{C}]\}$$

$$+.5(Dt/DV)\{-[Dy^{l}_{D}R_{D} - Dx^{l}_{D}S_{D}] + [Dx^{m}_{D}S_{D} - Dy^{m}_{D}R_{D}]\}$$

where R and S are defined at cell centers A,B,C and D as will be explained later. Cell centered values of Dx, Dy and DV are defined by equations 3.1 and 3.2. The particular form in which the above dissipative terms have been presented makes it easy to include into the distribution step of the solver.

To complete the discrete approximation of the dissipation terms the cell-centered values of R and S must be found. For convenience R and S will be defined using the base solver cell notation shown in figure 3-2. Recalling the definition of R and S from equation 2.1 as

$$R_{C} = (1/Re_{o}) \begin{vmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xx}u + \tau_{yx}v - (\mu/(\gamma-1)Pr)\,T_{x} \end{vmatrix}$$

$$S_{C} = (1/Re_{o}) \begin{vmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yy}v + \tau_{xy}u - (\mu/(\gamma-1)Pr)\,T_{y} \end{vmatrix}$$

it should be noted that both involve first derivatives of u and v with respect to Cartesian coordinates x and y. In order to approximate these derivatives with cell-centered differencing of nodes 1-4, for a general nonorthogonal cell,

these Cartesian derivatives must be transformed to derivatives with respect to computational coordinates ξ and η. This transformation is performed as follows

$$( )_x = J[\, y_\eta ( )_\xi - y_\xi ( )_\eta \,] \qquad (D.5)$$

$$( )_y = J[\, y_\xi ( )_\eta - y_\eta ( )_\xi \,]$$

In addition to a change of coordinates a large number of computational operations can be saved by rewriting derivatives of primitive variables u and v in terms of derivatives of conservation variables U. For example,

$$u_\xi = [\, (\rho u)_\xi - u\rho_\xi \,]/\rho \qquad (D.6)$$

$$u_\eta = [\, (\rho u)_\eta - u\rho_\eta \,]/\rho$$

Performing the above transformations results in the following expressions for the shear stresses

$$\tau_{xx} = -(\mu J/\rho)\{(4/3)[y_\eta(\rho u)_\xi - y_\xi(\rho u)_\eta - u(y_\eta\rho_\xi - y_\xi\rho_\eta)]$$
$$-(2/3)[x_\xi(\rho v)_\eta - x_\eta(\rho v)_\xi - v(x_\xi\rho_\eta - x_\eta\rho_\xi)]\}$$

$$\tau_{yy} = -(\mu J/\rho)\{(4/3)[x_\xi(\rho v)_\eta - x_\eta(\rho v)_\xi - v(x_\xi\rho_\eta - x_\eta\rho_\xi)]$$
$$-(2/3)[y_\eta(\rho u)_\xi - y_\xi(\rho u)_\eta - u(y_\eta\rho_\xi - y_\xi\rho_\eta)]\}$$

$$\tau_{xy} = -(\mu J/\rho)\{\, x_\xi(\rho u)_\eta - x_\eta(\rho u)_\xi + y_\eta(\rho v)_\xi - y_\xi(\rho v)_\eta$$
$$+(ux_\eta - vy_\eta)\rho_\xi - (ux_\xi - vy_\xi)\rho_\eta \,\} \qquad (D.7)$$

In addition,

$$T_x = J\,[\, y_\eta T_\xi - y_\xi T_\eta \,]$$
$$T_y = J\,[\, x_\xi T_\eta - x_\eta T_\xi \,] \qquad \text{(D.8)}$$

At this point the cell-centered discrete approximation for the first derivatives is defined using information at cell corner points 1-4. For example,

$$U_\eta = DU^{l} = 0.5(\, U_2 + U_3 - U_1 - U_4 \,) \qquad \text{(D.9)}$$
$$U_\xi = DU^{m} = 0.5(\, U_3 + U_4 - U_1 - U_2 \,)$$

where $D\xi$ and $D\eta$ are assumed to be equal to 1. This differencing and notation is consistent with the cell-centered metrics definitions of equation 3.2c.

Discretizing equation D.7, the final cell-centered form of R and S may be summarized for a typical cell C as

$$R_C = (1/Re_o) \begin{vmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xx}\,u + \tau_{yx}\,v - (\mu/(\gamma-1)Pr)\,T_x \end{vmatrix}_C$$

$$S_C = (1/Re_o) \begin{vmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yy}\,v + \tau_{xy}\,u - (\mu/(\gamma-1)Pr)\,T_y \end{vmatrix}_C$$

where

$$\tau_{xx} = -(\mu/\rho DV)\{(4/3)[Dy^{l}_{c} D(\rho u) - Dy^{m}_{c} D(\rho u) - u_{c}(Dy^{l}_{c} D\rho - Dy^{m}_{c} D\rho)]$$
$$-(2/3)[Dx^{m}_{c} D(\rho v) - Dx^{l}_{c} D(\rho v) - v_{c}(Dx^{m}_{c} D\rho - Dx^{l}_{c} D\rho)]\}$$

$$\tau_{yy} = -(\mu/\rho DV)\{(4/3)[Dx^{m}_{c} D(\rho v) - Dx^{l}_{c} D(\rho v) - v_{c}(Dx^{m}_{c} D\rho - Dx^{l}_{c} D\rho)]$$
$$-(2/3)[Dy^{l}_{c} D(\rho u) - Dy^{m}_{c} D(\rho u) - u_{c}(Dy^{l}_{c} D\rho - Dy^{m}_{c} D\rho)]\}$$

$$\tau_{xy} = -(\mu/\rho DV)\{Dx^{m}_{c} D(\rho u) - Dx^{l}_{c} D(\rho u) + Dy^{l}_{c} D(\rho v) - Dy^{m}_{c} D(\rho v)$$
$$+(u_{c} Dx^{l}_{c} - v_{c} Dy^{l}_{c})D\rho - (u_{c} Dx^{m}_{c} - v_{c} Dy^{m}_{c})D\rho\} \qquad (D.10a)$$

In addition,

$$T_{x} = [Dy^{l}_{m} DT - Dy^{m}_{l} DT]/DV$$

$$T_{y} = [Dx^{m}_{l} DT - Dx^{l}_{m} DT]/DV \qquad (D.10b)$$

and

$$D(\ )^{l} = 0.5[(\ )_{2} + (\ )_{3} - (\ )_{1} - (\ )_{4}]$$

$$D(\ )^{m} = 0.5[(\ )_{3} + (\ )_{4} - (\ )_{1} - (\ )_{2}] \qquad (D.10c)$$

This completes the formulation of the dissipation terms for the Navier-Stokes equations. These terms will now be included into the base solver presented in chapter 3 resulting in a formulation of Ni's multiple-grid method for solution of the Navier-Stokes equations.

Following the inviscid formulation of the base solver, the viscous base solver is performed with three sweeps over the fine mesh. First, the mesh is swept node by node, initializing the node point corrections dU to zero. After initialization, the second sweep is made, cell by cell, performing a inviscid flux balance, a calculation of R and S, and a modified distribution for each cell. For the typical cell shown in figure 3-2 this involves the following 3 steps.

STEP 1: Inviscid Finite Volume Approximation

This step involves calculation of DU as defined by equation 3.1. Note that this is no longer the discrete approximation to the governing integral equation since it does not include the dissipative components.

STEP 2: Calculation of R and S

R and S are calculated at the cell center using equations D.10.

STEP 3: Modified Distribution

$$dU_1 = dU_1 + 0.25[\, DU_c - Df_c - Dg_c - Dr_c - Ds_c \,] \quad \text{(D.11a)}$$

$$dU_2 = dU_2 + 0.25[\, DU_c - Df_c + Dg_c - Dr_c + Ds_c \,]$$

$$dU_3 = dU_3 + 0.25[\, DU_c + Df_c + Dg_c + Dr_c + Ds_c \,]$$

$$dU_4 = dU_4 + 0.25[\, DU_c + Df_c - Dg_c + Dr_c - Ds_c \,]$$

where

$$Df_c = (Dt/ DV)[DF_c \, Dy^l - DG_c \, Dx^l] \qquad (D.11b)$$

$$Dg_c = (Dt/ DV)[DG_c \, Dx^m - DF_c \, Dy^m]$$

$$Dr_c = (2Dt/ DV)[DR_c \, Dy^l - DS_c \, Dx^l]$$

$$Ds_c = (2Dt/ DV)[DS_c \, Dx^m - DR_c \, Dy^m]$$

and

$$DF_c = (\partial F/ \partial U) DU_c \qquad DG_c = (\partial G/ \partial U) DU_c \qquad (D.11c)$$

$$Dx^l = 0.5(x_2 + x_3 - x_1 - x_4) \qquad Dy^l = 0.5(y_2 + y_3 - y_1 - y_4)$$

$$Dx^m = 0.5(x_3 + x_4 - x_1 - x_2) \qquad Dy^m = 0.5(y_3 + y_4 - y_1 - y_2)$$

$$U_c = 0.25(U_1 + U_2 + U_3 + U_4)$$

Note that the control volume integration for the dissipative terms is actually being performed in the distribution step.

Once the solution sweep has been performed over each cell on the fine mesh, the required boundary conditions are applied to the boundary nodes. It is important to note that with this formulation the changes predicted at all boundaries are incorrect due to the way the dissipation terms are approximated. For the dissipation terms to be correct, changes must be distributed from all four surrounding cells, only then is the finite volume integration of the dashed cell

in figure D-1 complete. This point is not important for solid wall boundary points, since the boundary condition is imposed by extrapolating the pressure from the flow and setting the velocity to zero and temperature to the wall temperature. However, the far field boundary conditions require these predicted boundary changes. This problem is corrected by dropping the dissipative part of the distribution formula for distributions to the far field boundaries. This is equivalent to a inviscid flow assumption at the far field boundary, an appropriate approximation in this region. With this correction the inviscid far field boundary conditions presented in equations 3.24 and 3.25 may be used.

After application of the boundary conditions the solution is updated as

$$U_i^{n+1} = U_i^n + dU_i \qquad \text{(D.12)}$$

This completes the formulation of the viscous base solver. After a pass over the fine mesh the inviscid coarse mesh accelerator described in section 3.3 is applied on the 2h mesh, the 4h mesh, and so on. The coarse mesh accelerator, which is now operating on residuals of the Navier-Stokes equations, rapidly propagates errors in the solution by modeling only the dominate convective terms.

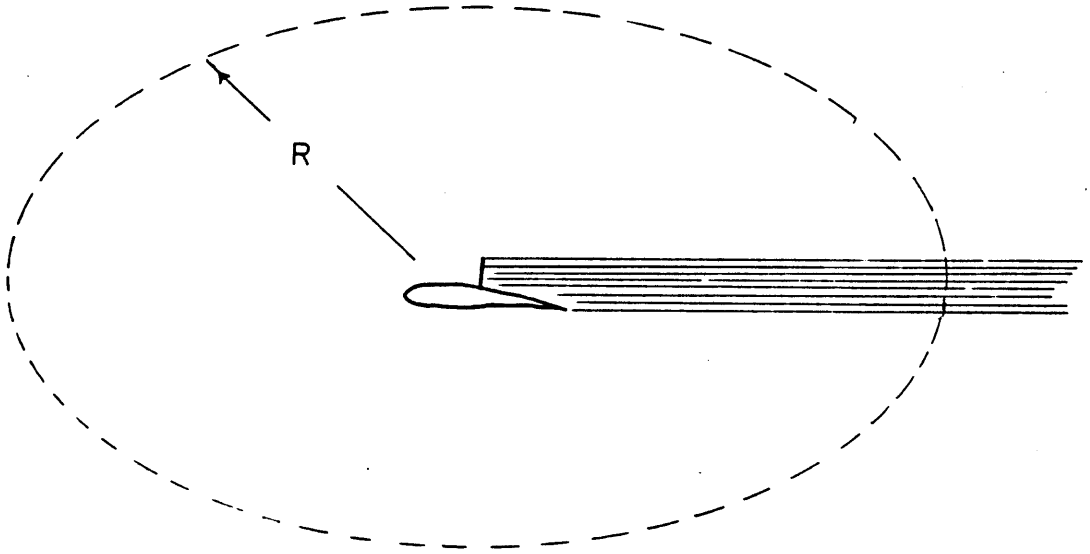

Figure 2-1. Typical two dimensional transonic airfoil flow.

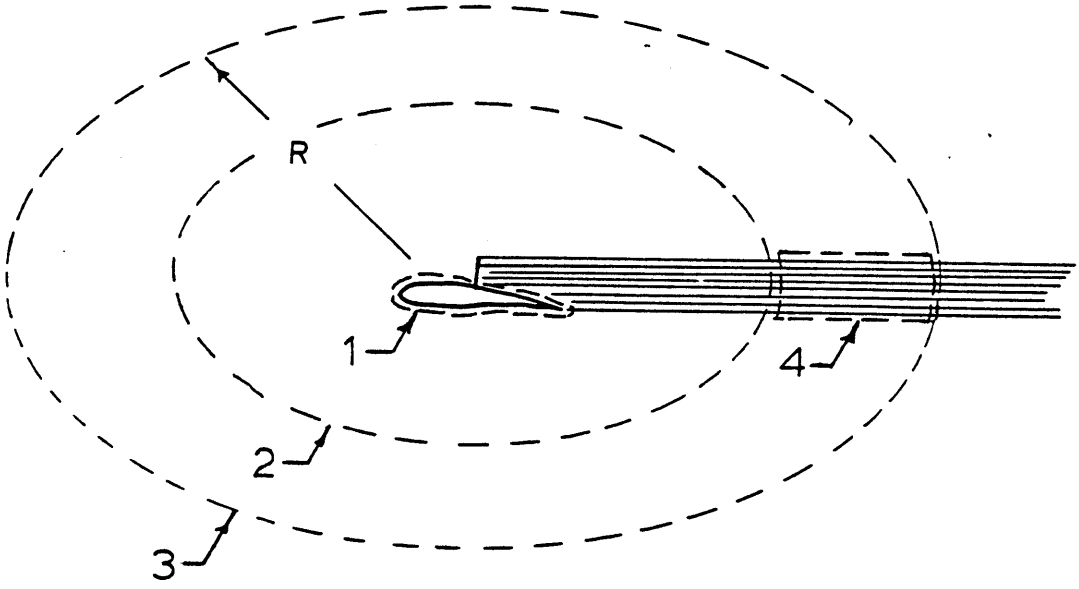

Figure 2-2. Three contours for far field boundary condition discussion.

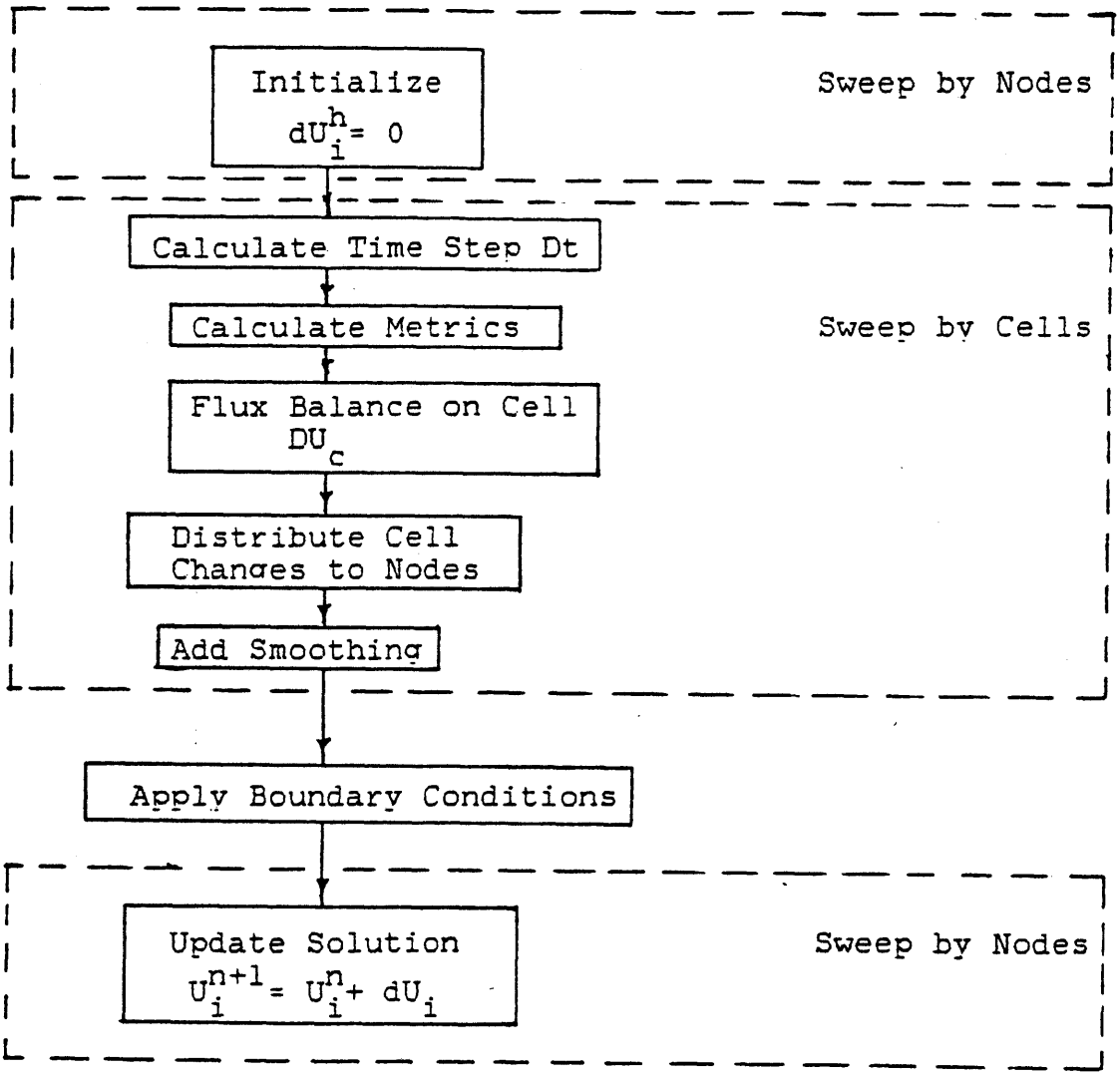

Figure 3-1. Flow chart for base solver.

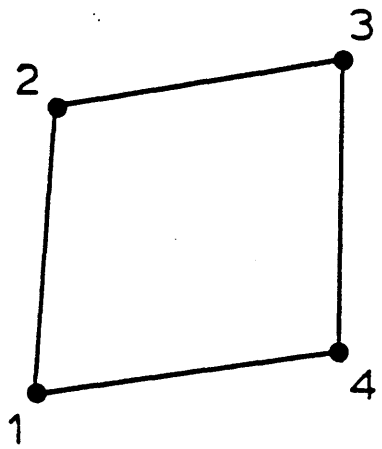

Figure 3-2. Base solver cell notation.

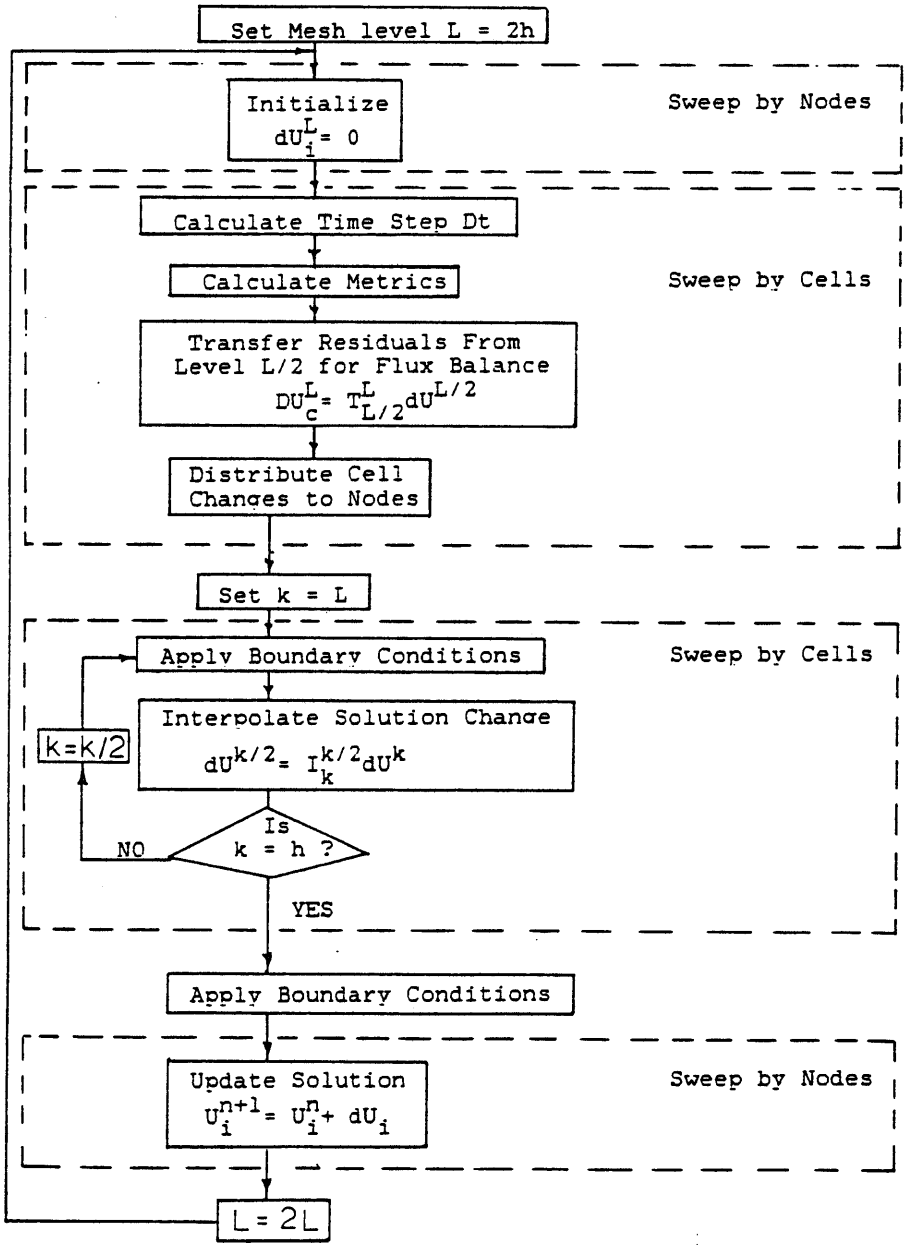

Figure 3-3. Coarse mesh accelerator flow chart.

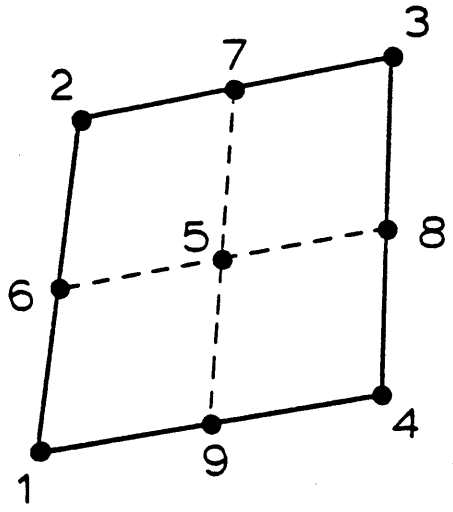

Figure 3-4. Coarse mesh accelerator cell notation.

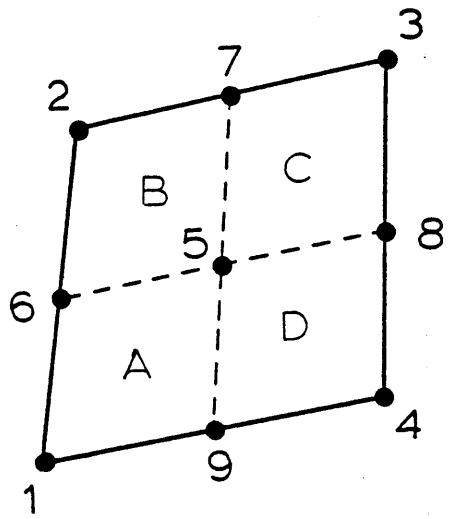

Figure 3-5. Cell notation for transfer operator discussion.

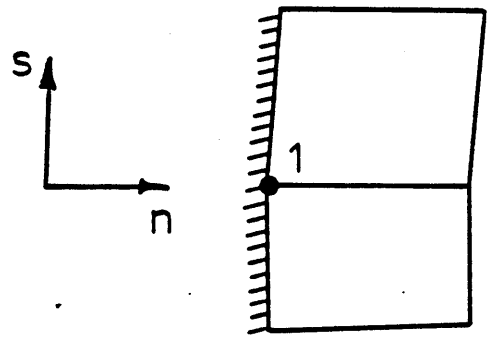

Figure 3-6. Boundary cell notation.

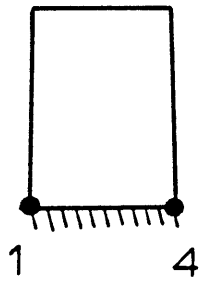

Figure 3-7. Boundary cell notation for boundary cell smoothing discussion.

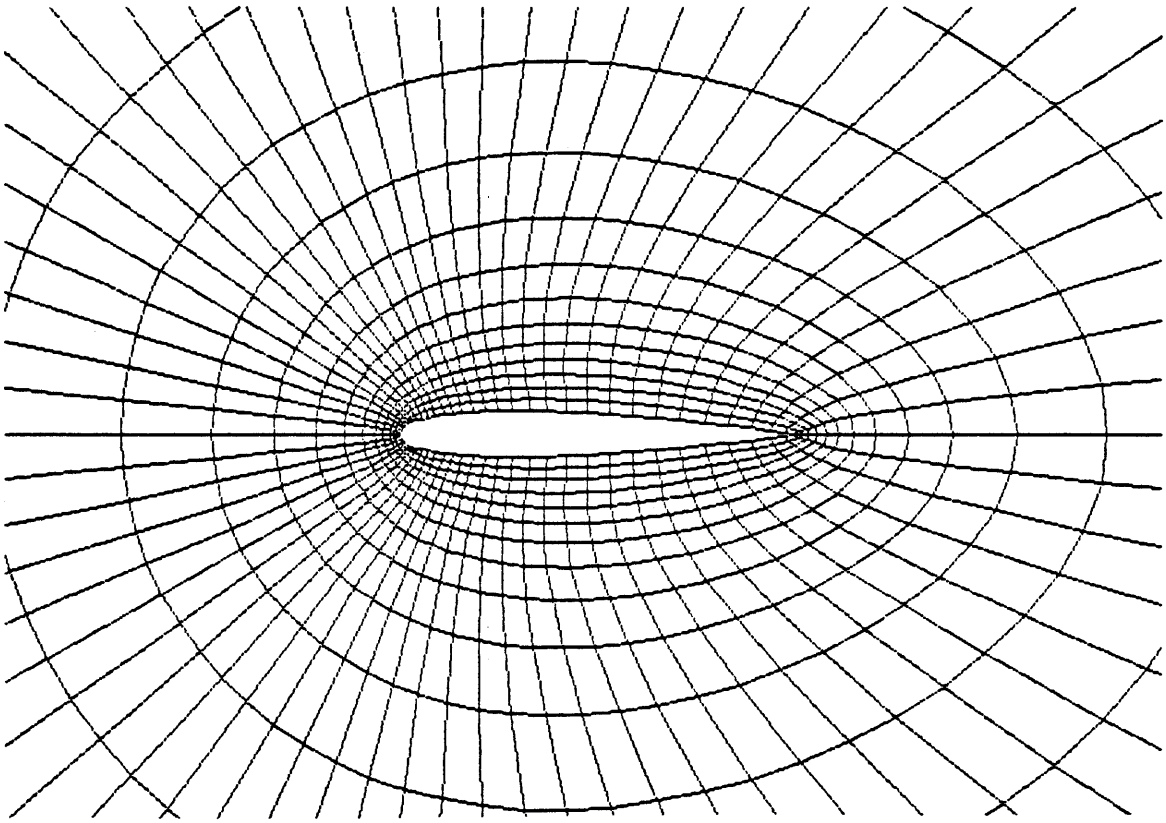

Figure 3-8. Near field of NACA0012 airfoil for 65*17 global O-type mesh. [run 179]

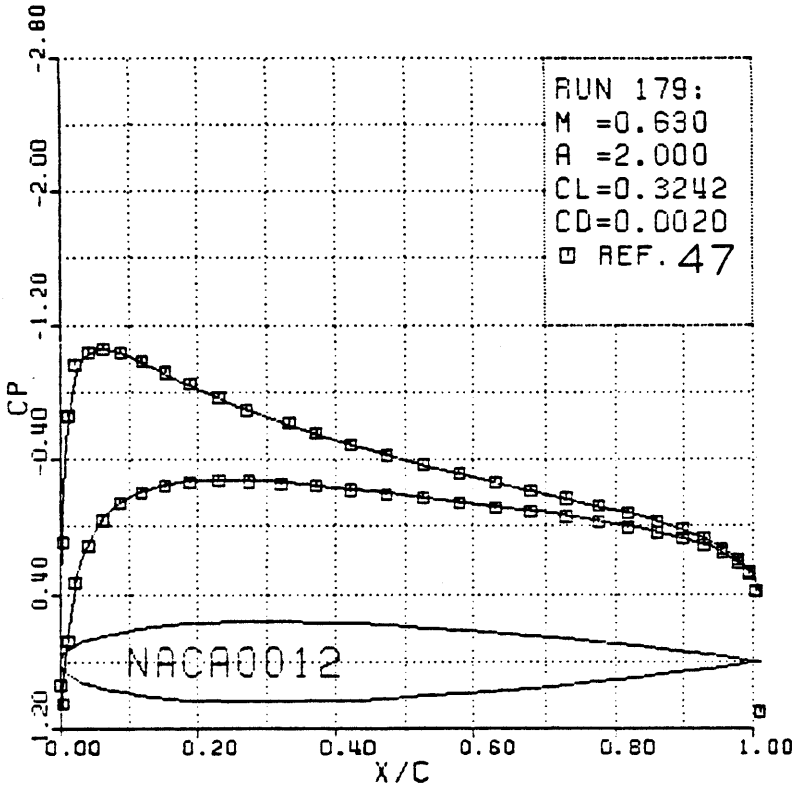

Figure 3-9a. Surface pressure coefficient.

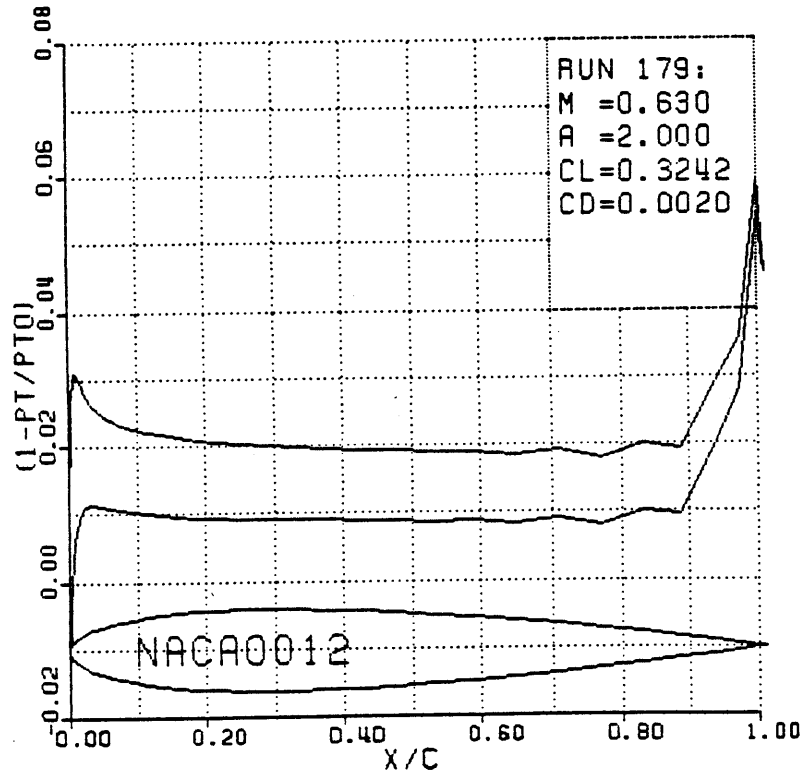

Figure 3-9b. Surface total pressure loss.

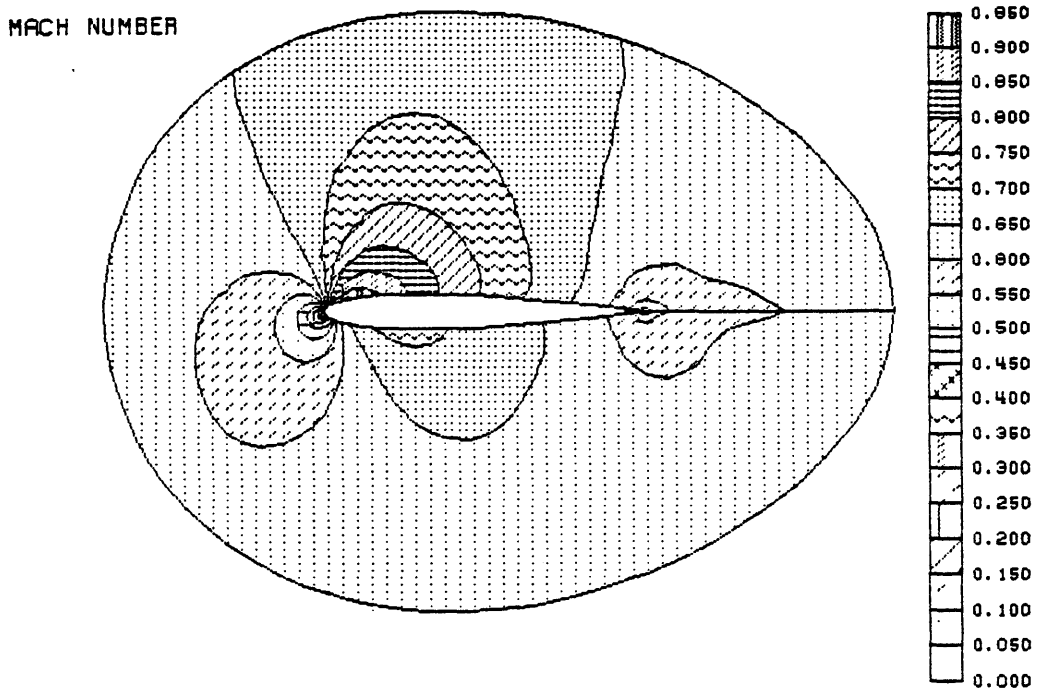

Figure 3-9c. Mach number contours.

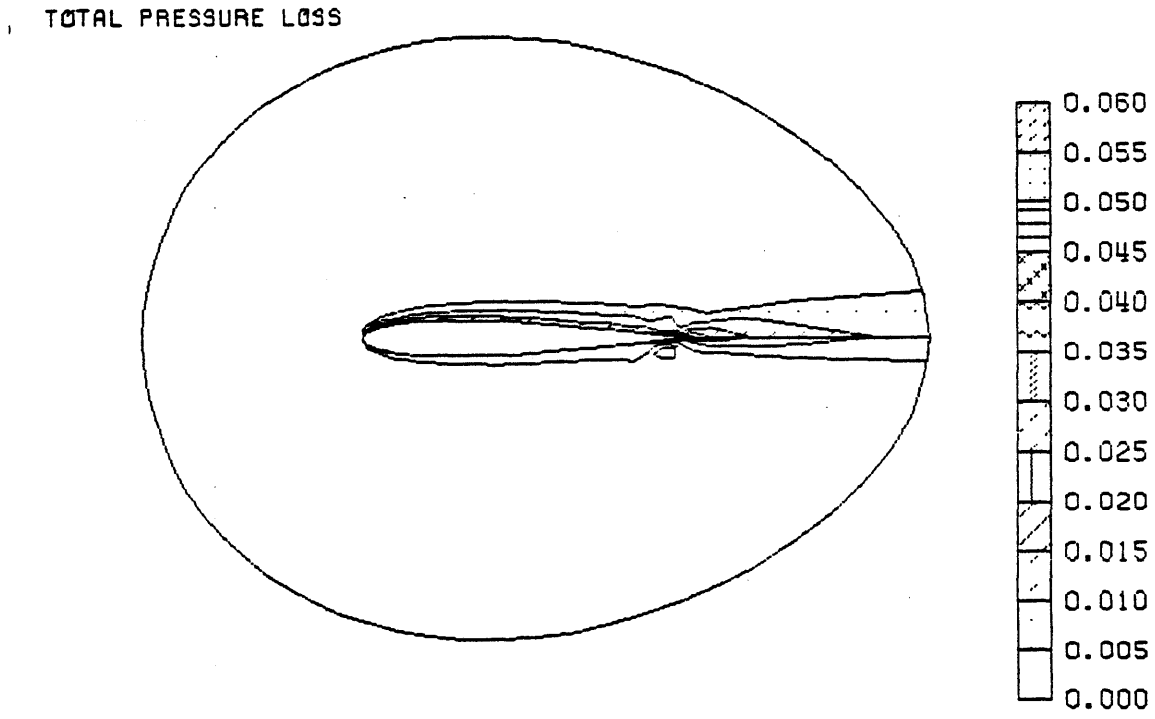

Figure 3-9d. Total pressure loss contours.

Figure 3-9. NACA0012 airfoil for M = 0.63 and angle of attack of 2.0 degrees. Base solver solution on 65*17 O-type mesh. [run 179]

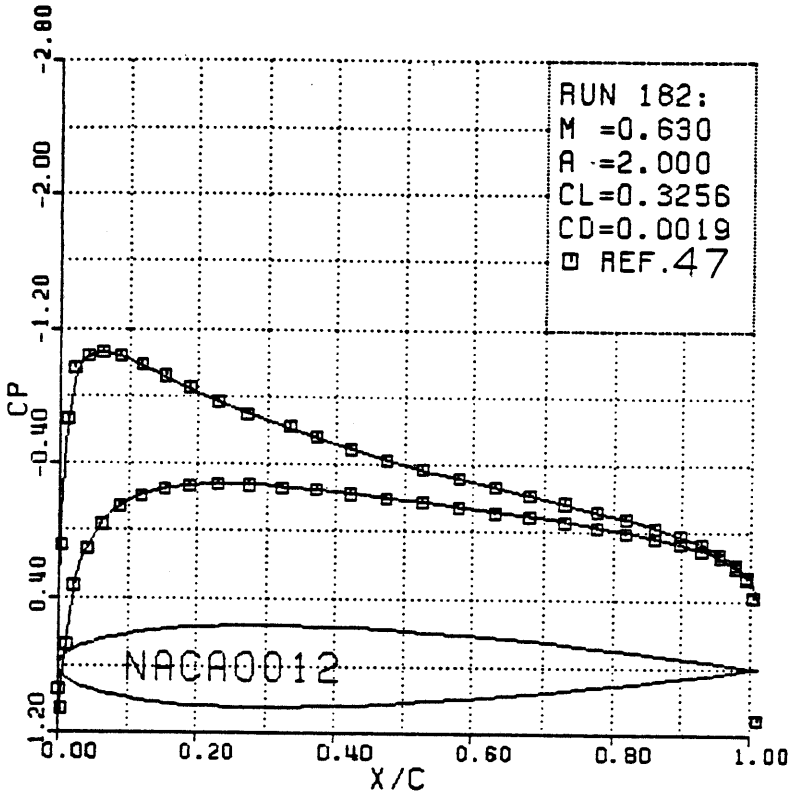

Figure 3-10a. Surface pressure coefficient.

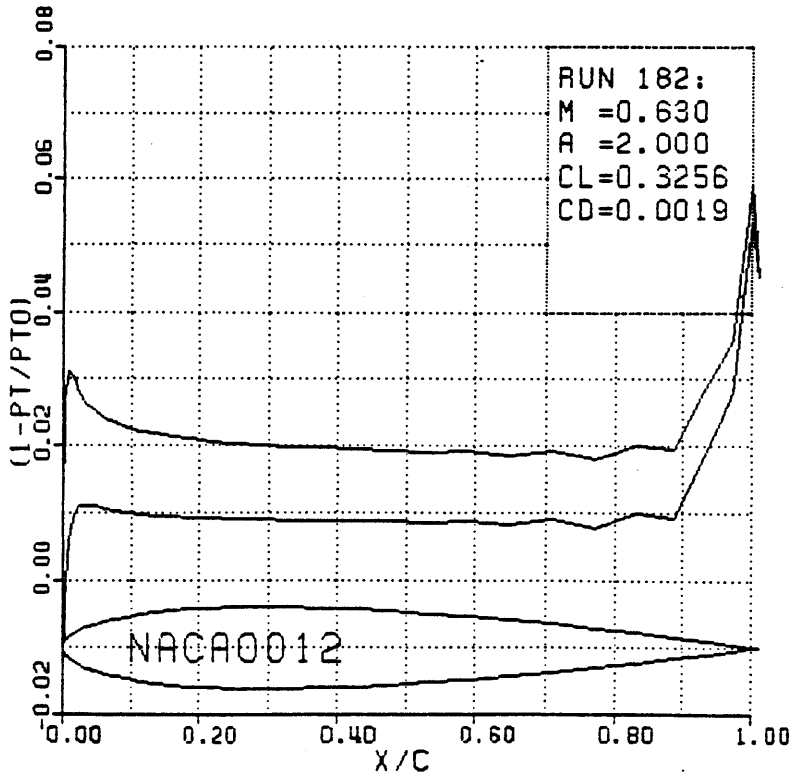

Figure 3-10b. Surface total pressure loss.

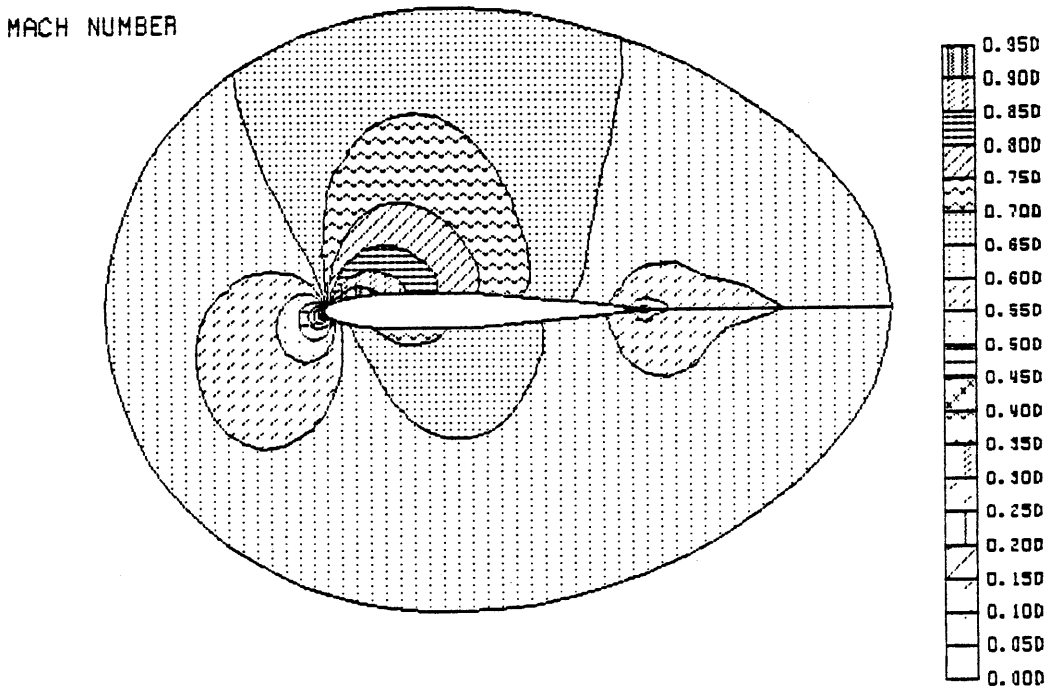

Figure 3-10c. Mach number contours.

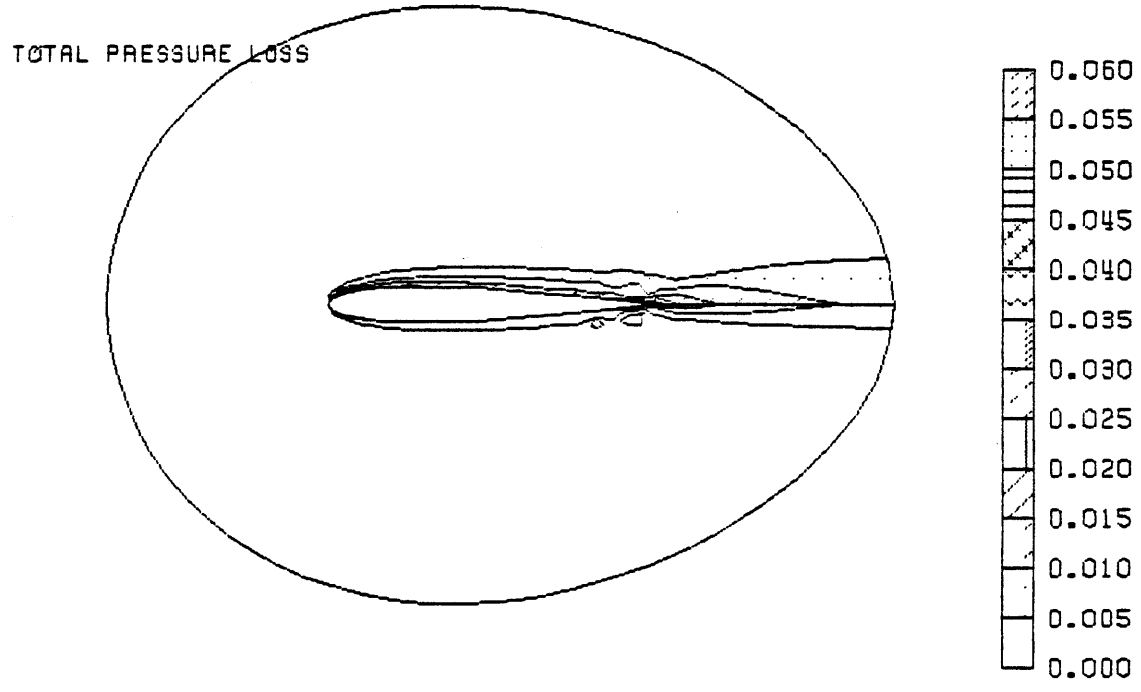

Figure 3-10d. Total pressure loss contours.

Figure 3-10. NACA0012 airfoil for M = 0.63 and 2.0 degree angle of attack. Multiple-grid solution on 65*17 O-type mesh with 3 global levels. [run 182]

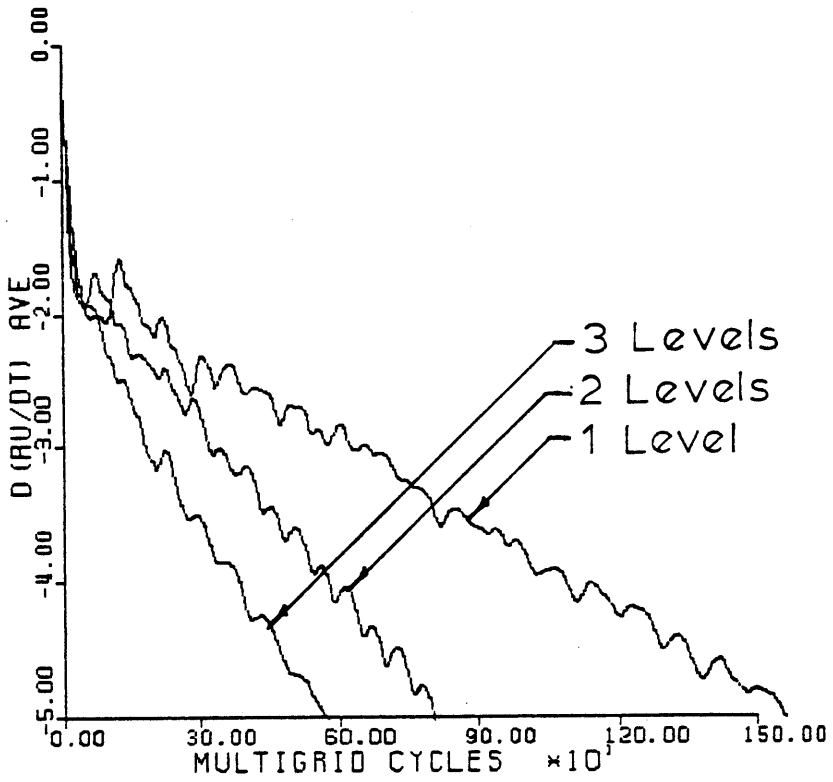

Figure 3-11a. Comparison of Convergence histories for NACA0012 airfoil for M = 0.63 and 2.0 degrees angle of attack with 1,2, and 3 global mesh levels. [runs 179,180,182]

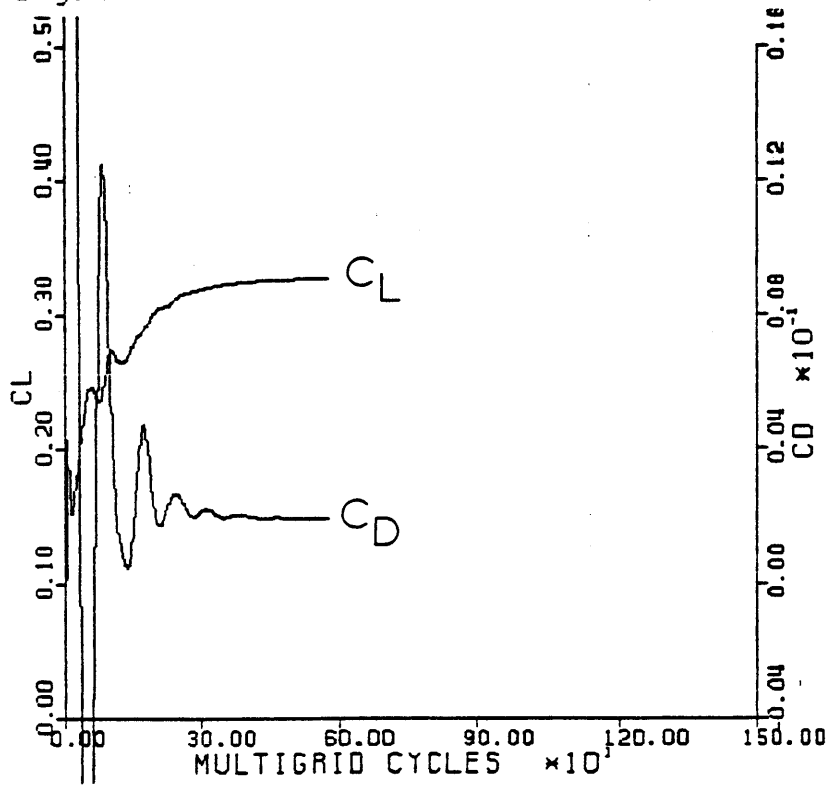

Figure 3-11b. Lift and drag coefficient histories as a function of multiple-grid cycle for NACA0012 airfoil at M = 0.63 and 2.0 degrees angle of attack with 3 global mesh levels.

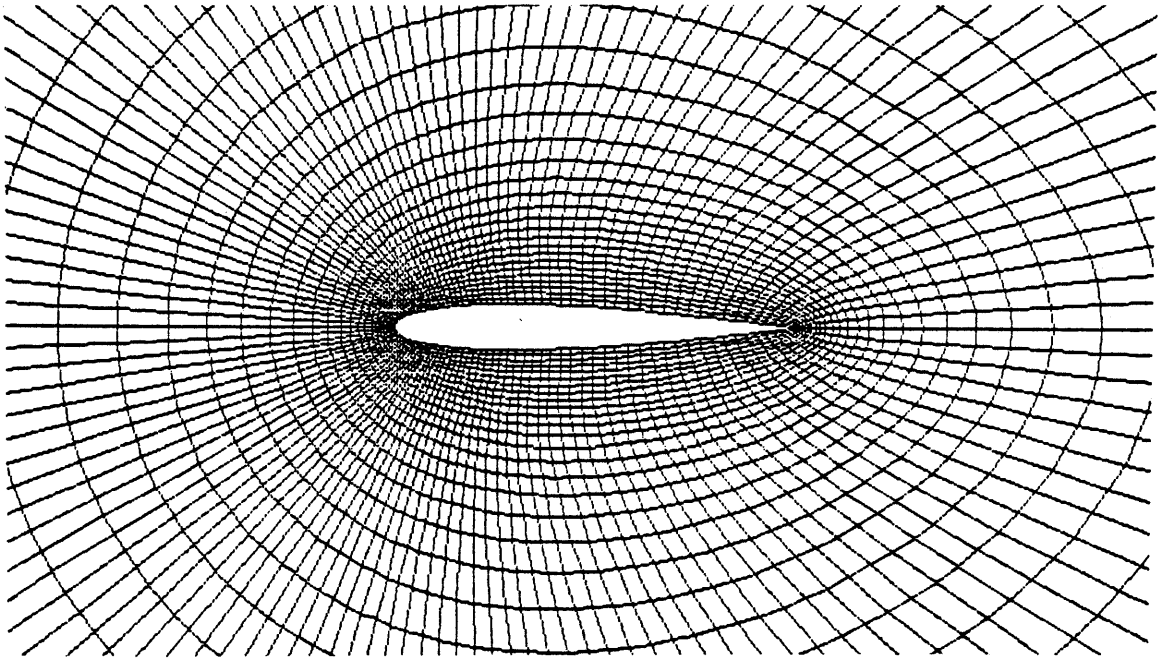

Figure 3-12. Near field of NACA0012 airfoil for 129*33 global O-type mesh. [run 181]

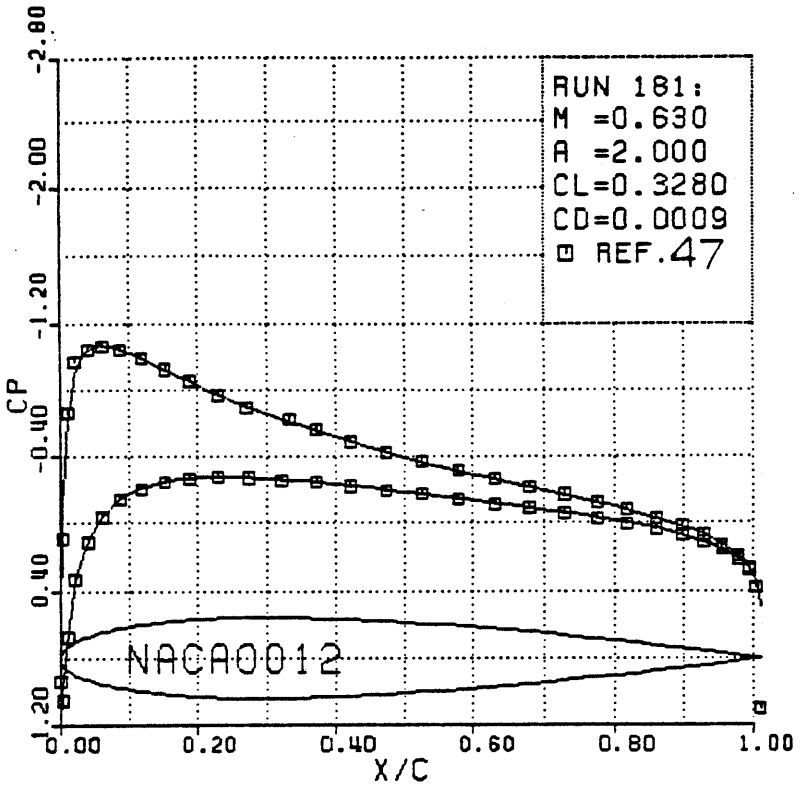

Figure 3-13a. Surface pressure coefficient.

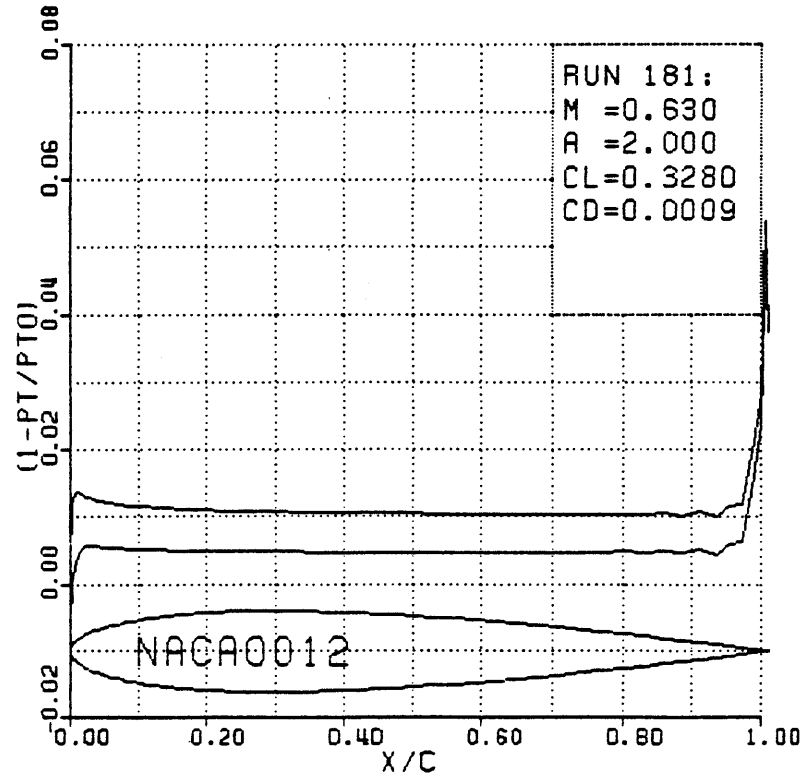

Figure 3-13b. Surface total pressure loss.

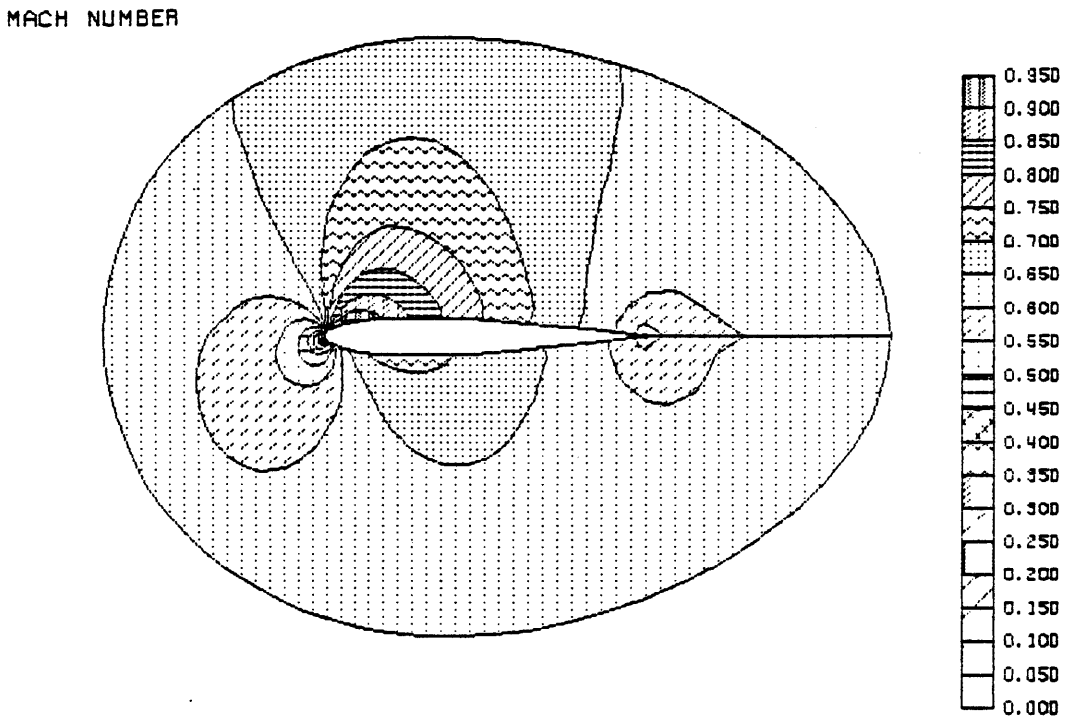

Figure 3-13c. Mach number contours.

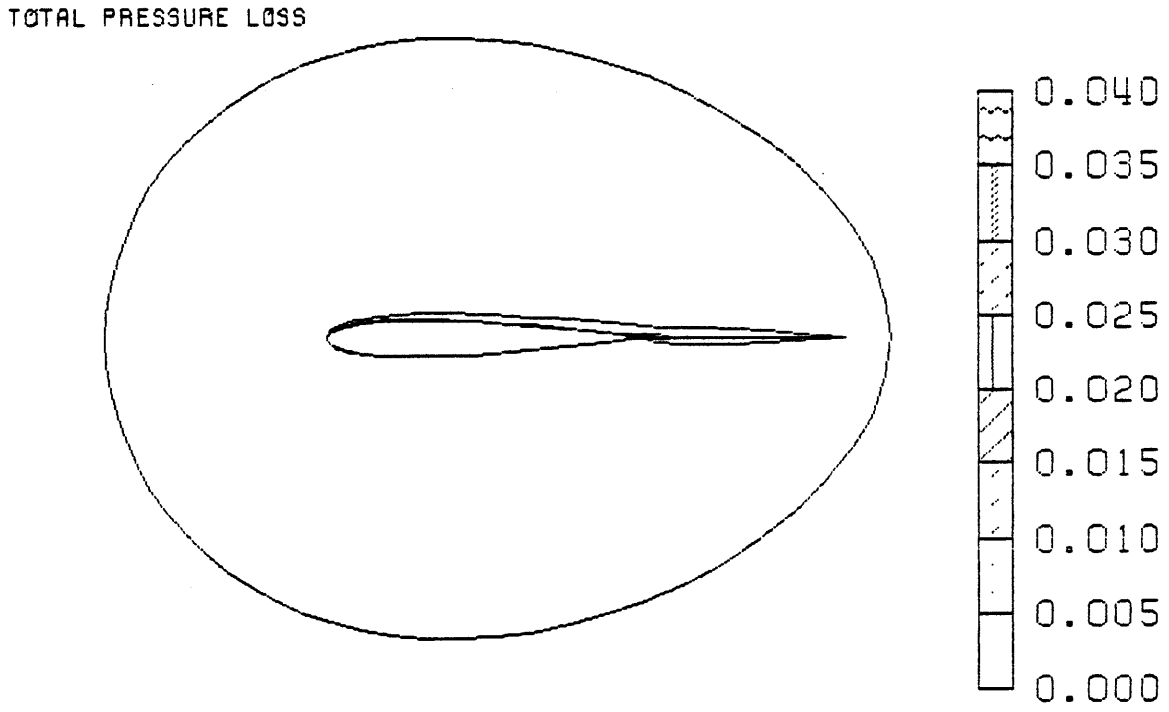

Figure 3-13d. Total pressure loss contours.

Figure 3-13. NACA0012 airfoil for M = 0.63 and 2.0 degrees angle of attack. Multiple-grid solution on 129*33 O-type mesh with 4 global levels. [run 181]

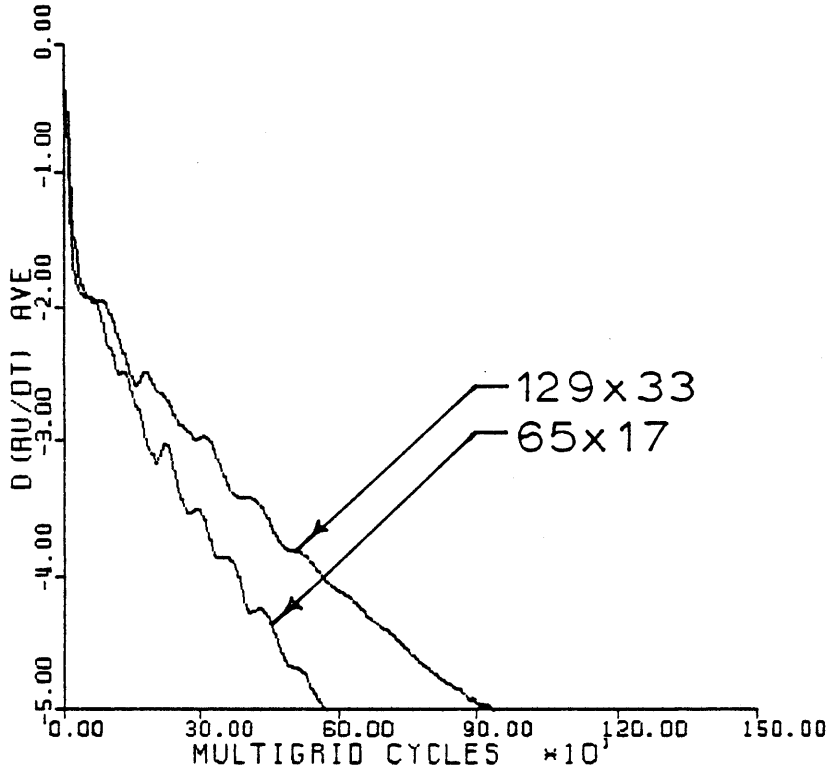

Figure 3-14. Comparison of Convergence histories for NACA0012 airfoil for M = 0.63 and 2.0 degrees angle of attack for 65*17 global solution with 3 levels and 129*33 global solution with 4 levels. [runs 182,181]

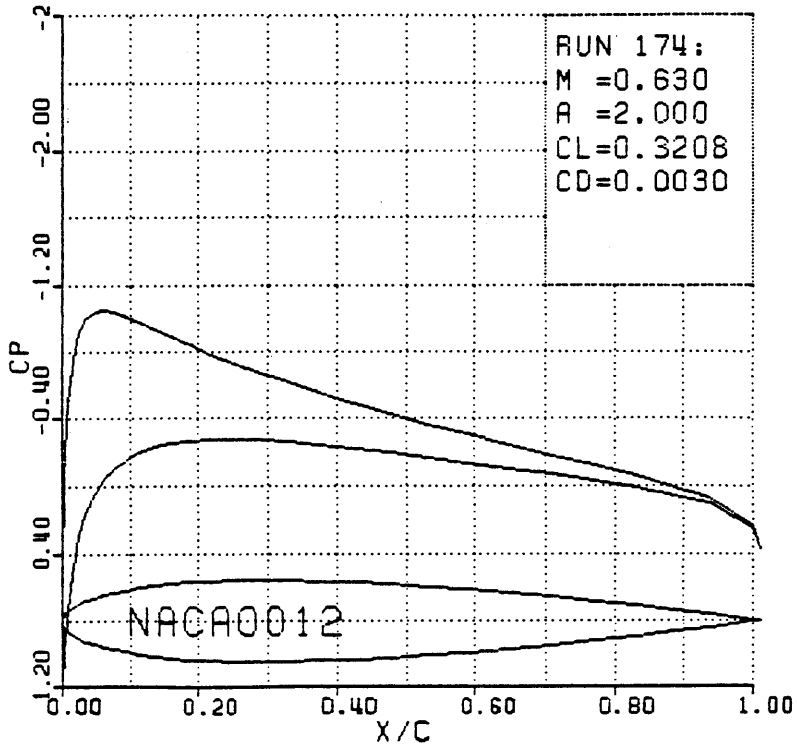

Figure 3-15a. Smoothing coefficient of 0.08.

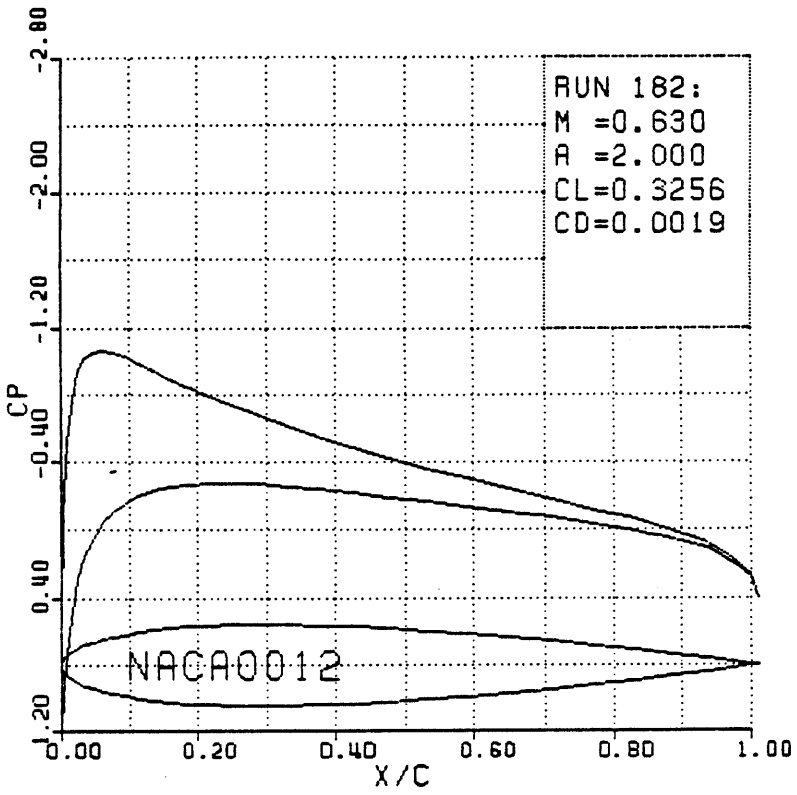

Figure 3-15b. Smoothing coefficient of 0.05.

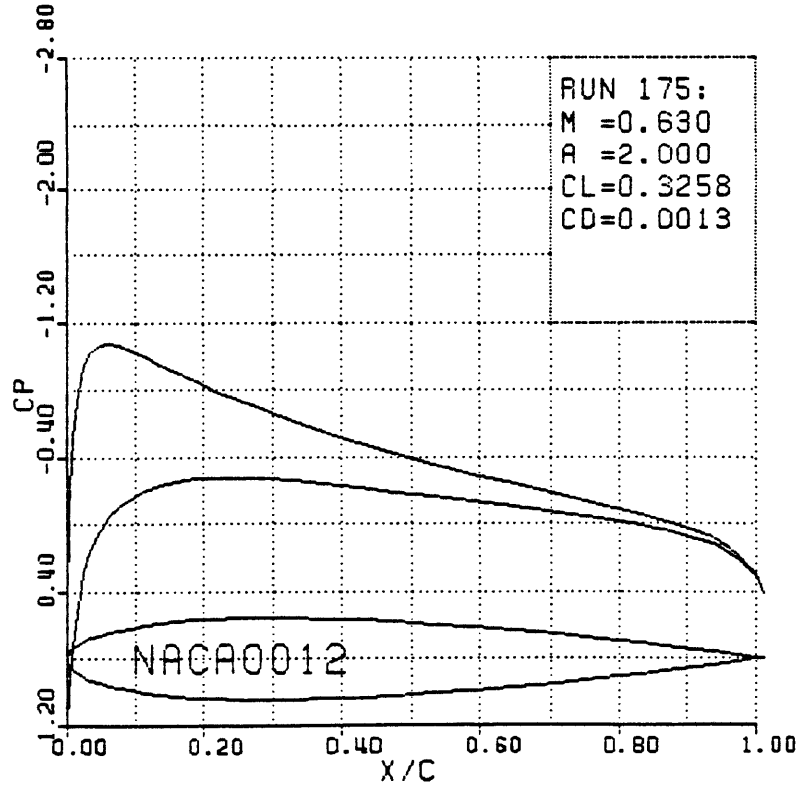

Figure 3-15c. Smoothing coefficient of 0.03.

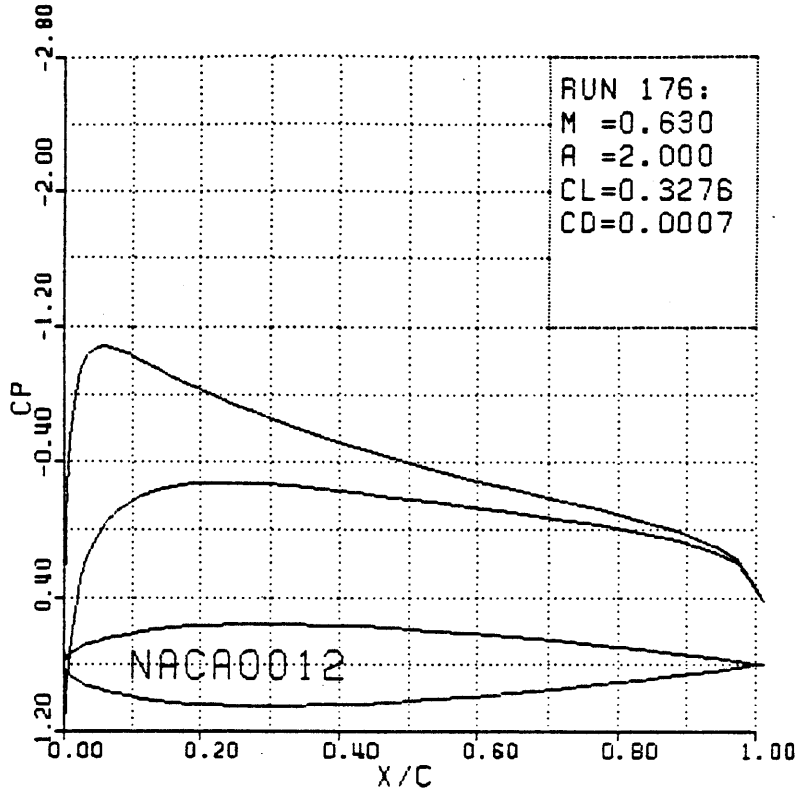

Figure 3-15d. Smoothing coefficient of 0.01.

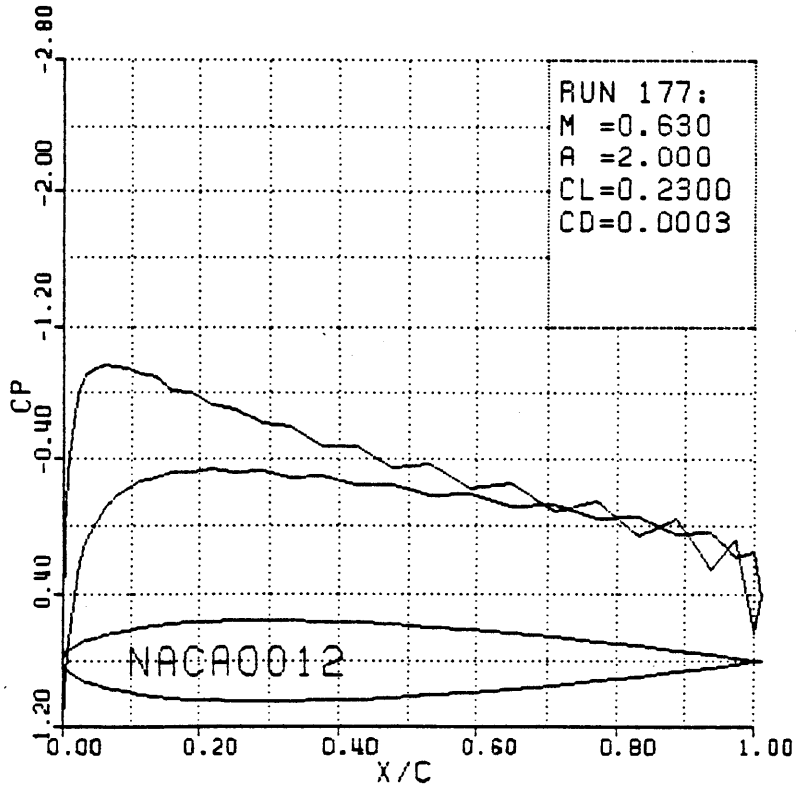

Figure 3-15e. Smoothing coefficient of 0.0.

Figure 3-15. Comparison of surface pressure coefficient for various smoothing coefficient values for O-mesh. [runs 174,182,175,176,177]

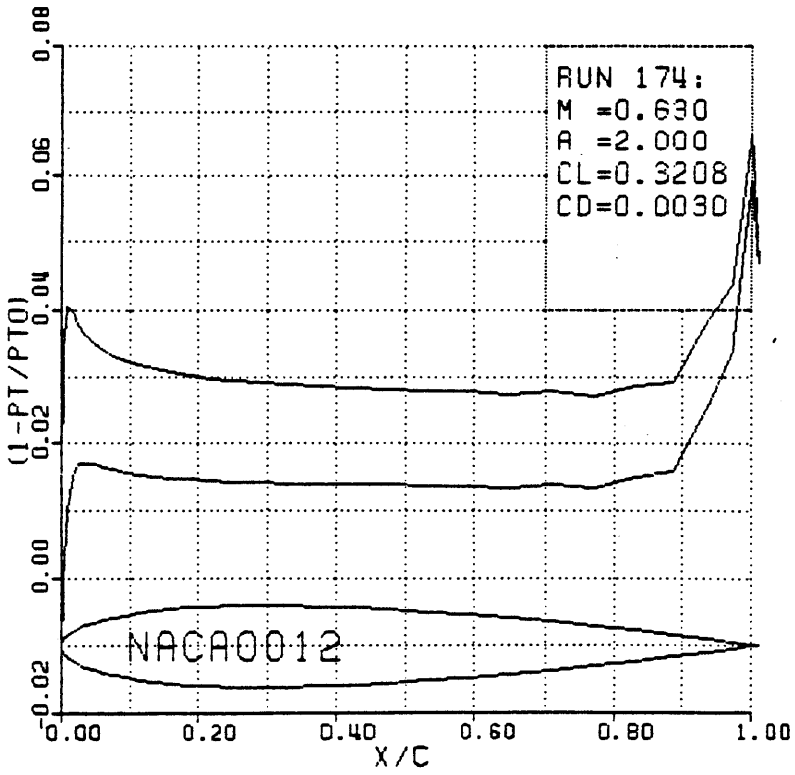

Figure 3-16a. Smoothing coefficient of 0.08.

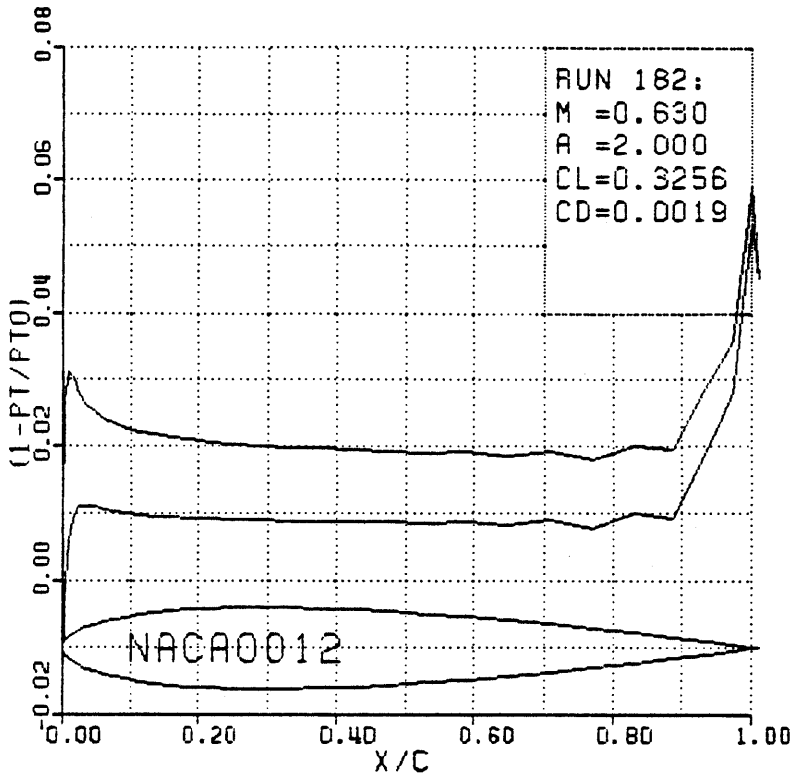

Figure 3-16b. Smoothing coefficient of 0.05.

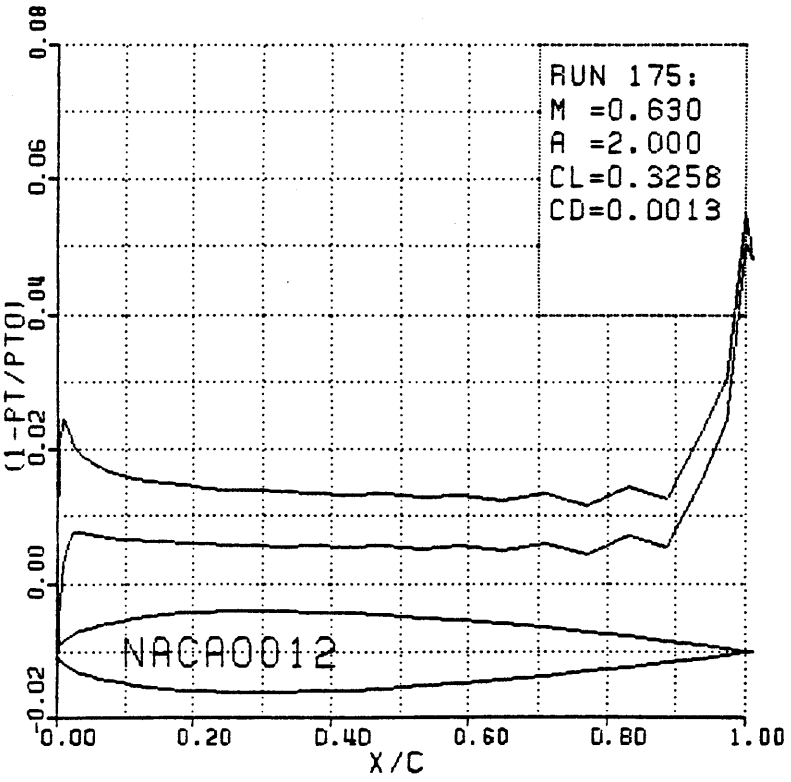

Figure 3-16c. Smoothing coefficient of 0.03.

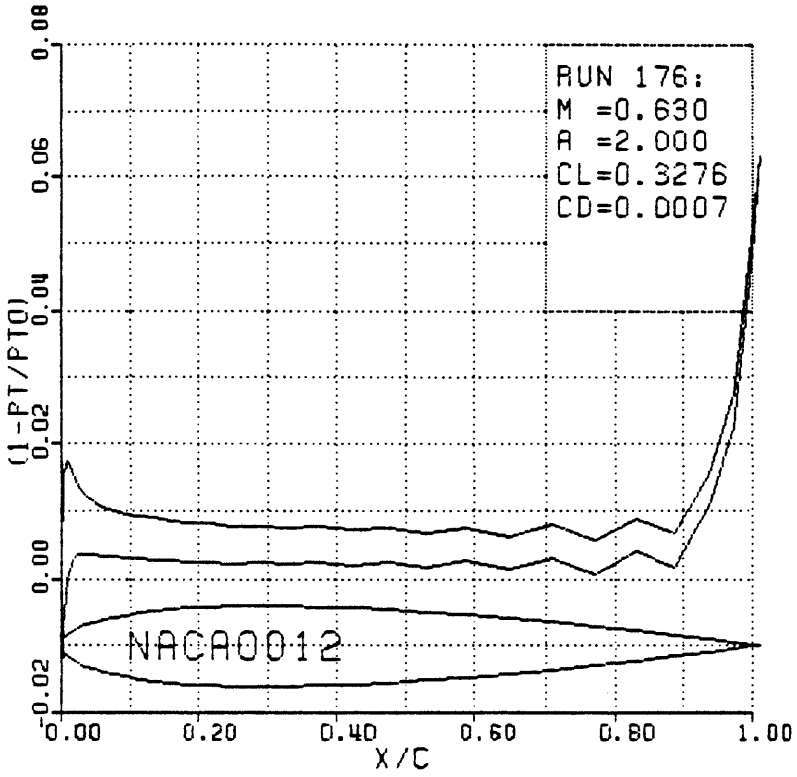

Figure 3-16d. Smoothing coefficient of 0.01.

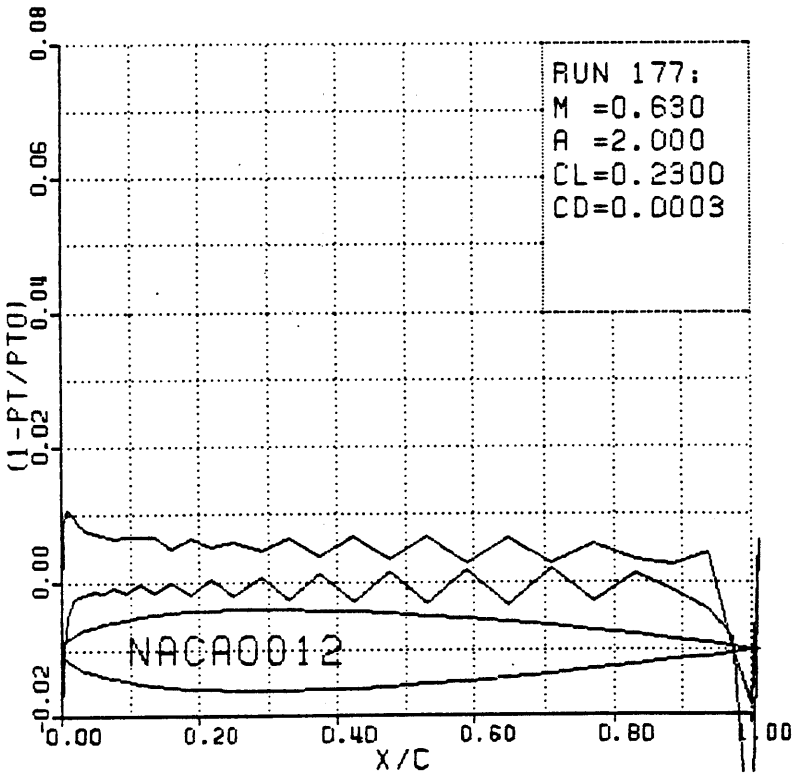

Figure 3-16e. Smoothing coefficient of 0.0.

Figure 3-16. Comparison of surface total pressure loss for various smoothing coefficient values for O-mesh. [runs 174,182,175,176,177]

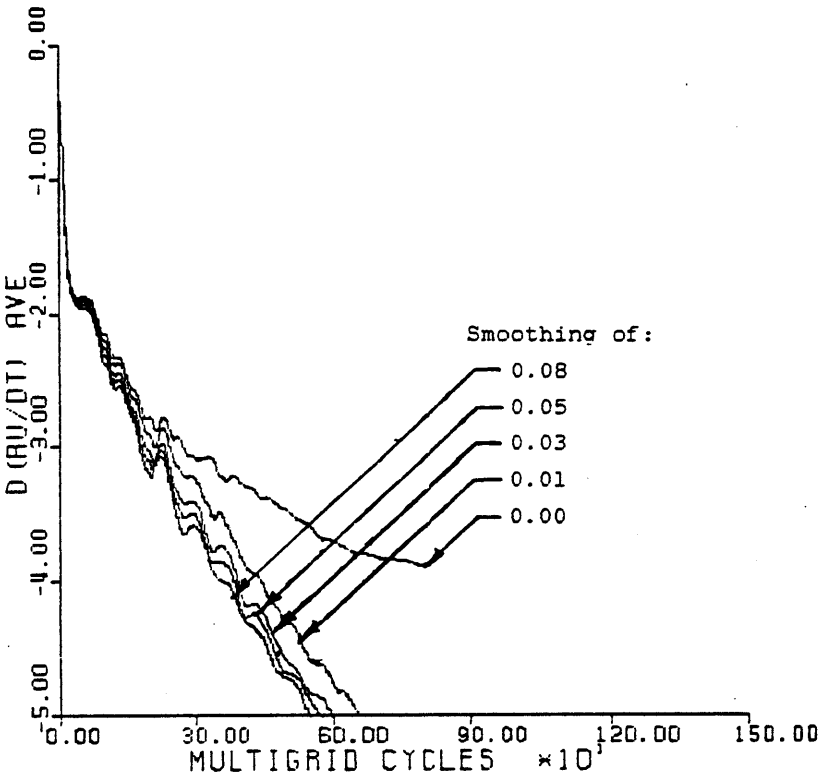

Figure 3-17. Comparison of convergence rates as a function of the smoothing coefficient of the O-type mesh. [runs 174,182,175,176,177]

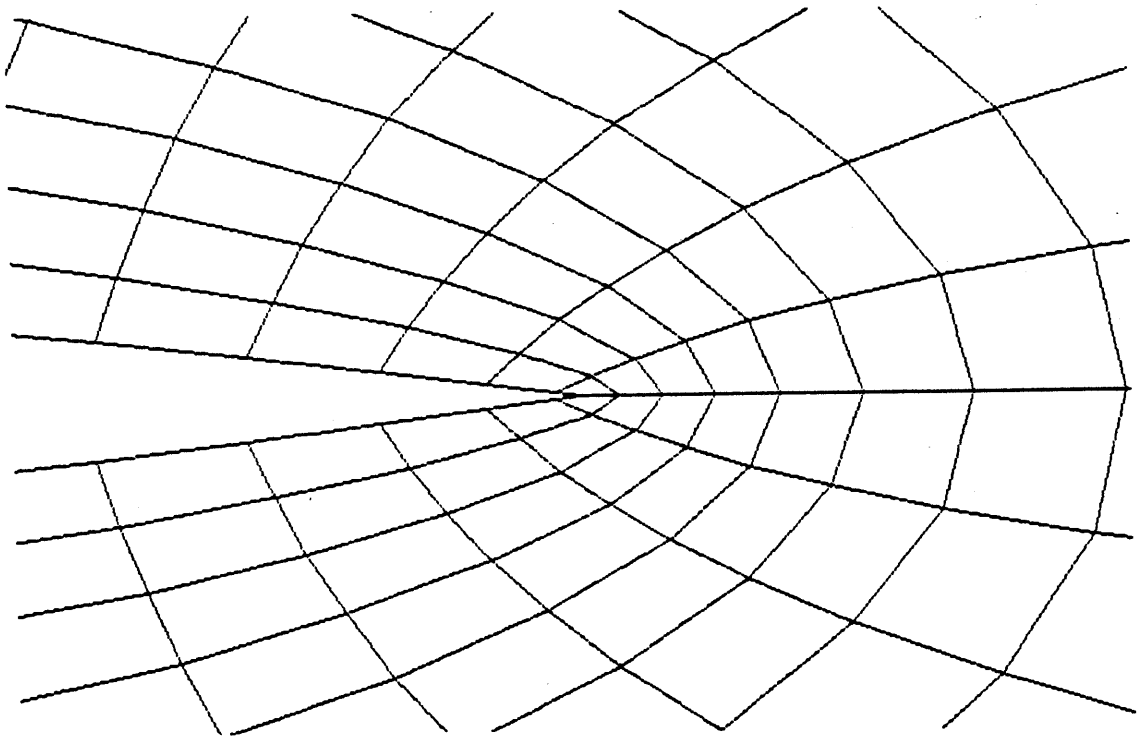

Figure 3-18. Blowup of mesh in the trailing edge region for NACA0012 for 65*17 O-type mesh. [run 182]

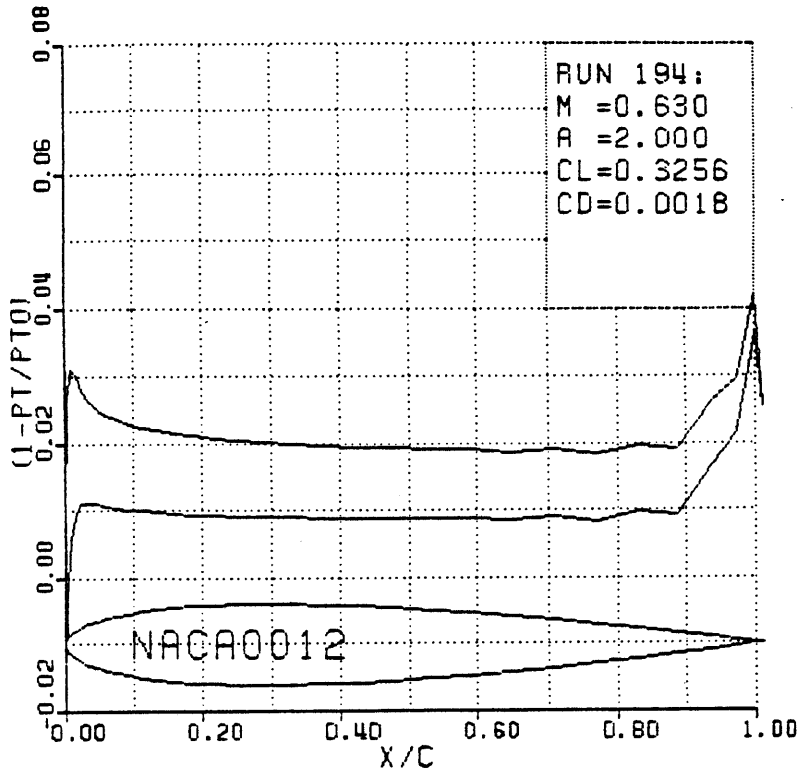

Figure 3-19. Total pressure loss for 65*17 O-type mesh with reduced skewness in trailing edge region. [run 194]

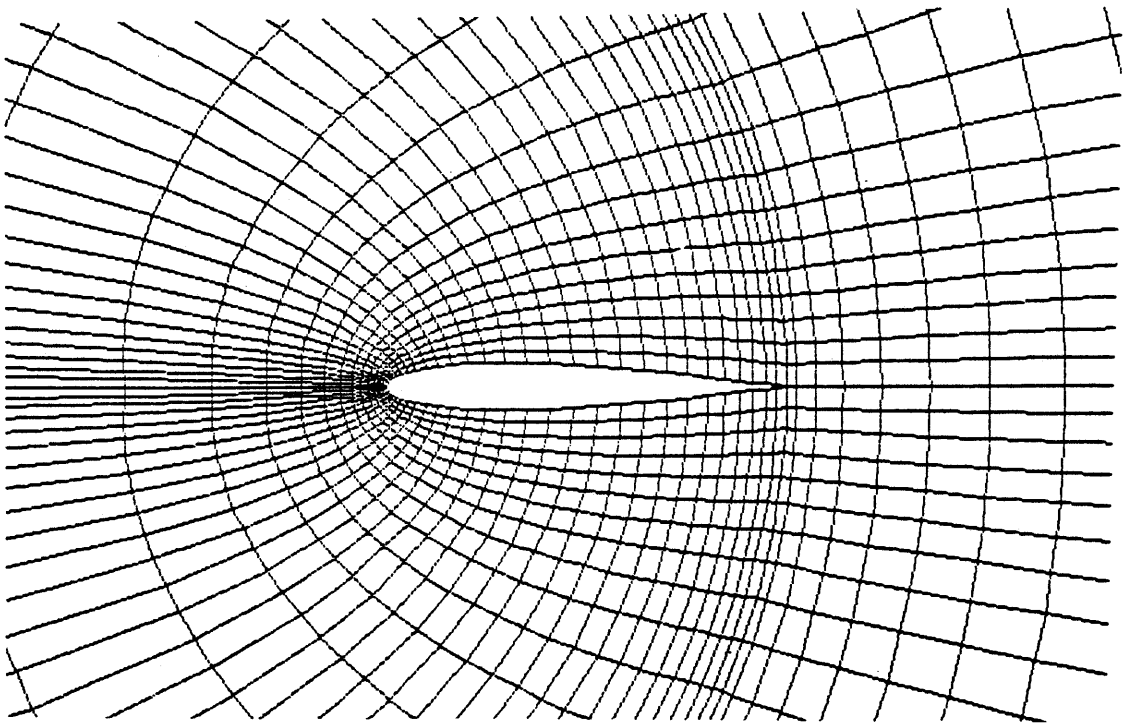

Figure 3-20. Near field of NACA0012 airfoil for 97*17 global C-type mesh. [run 189]

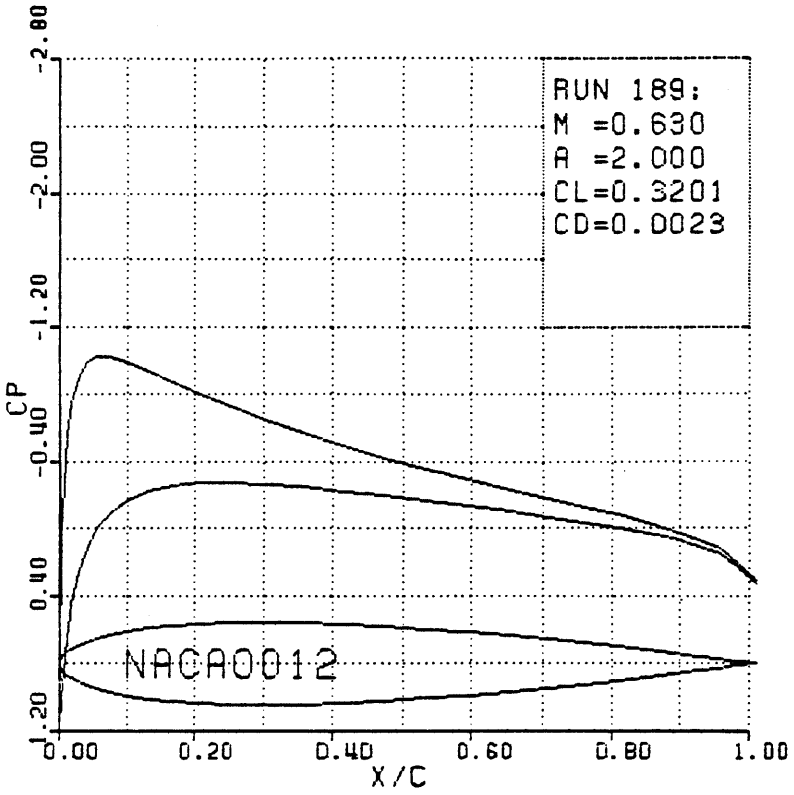

Figure 3-21a. Surface pressure coefficient.

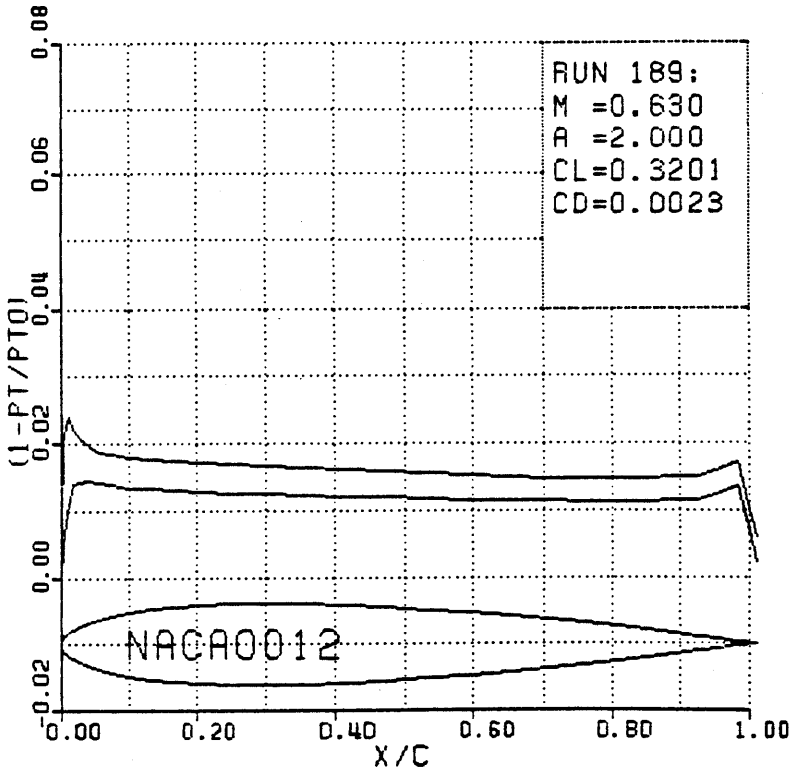

Figure 3-21b. Surface total pressure loss.

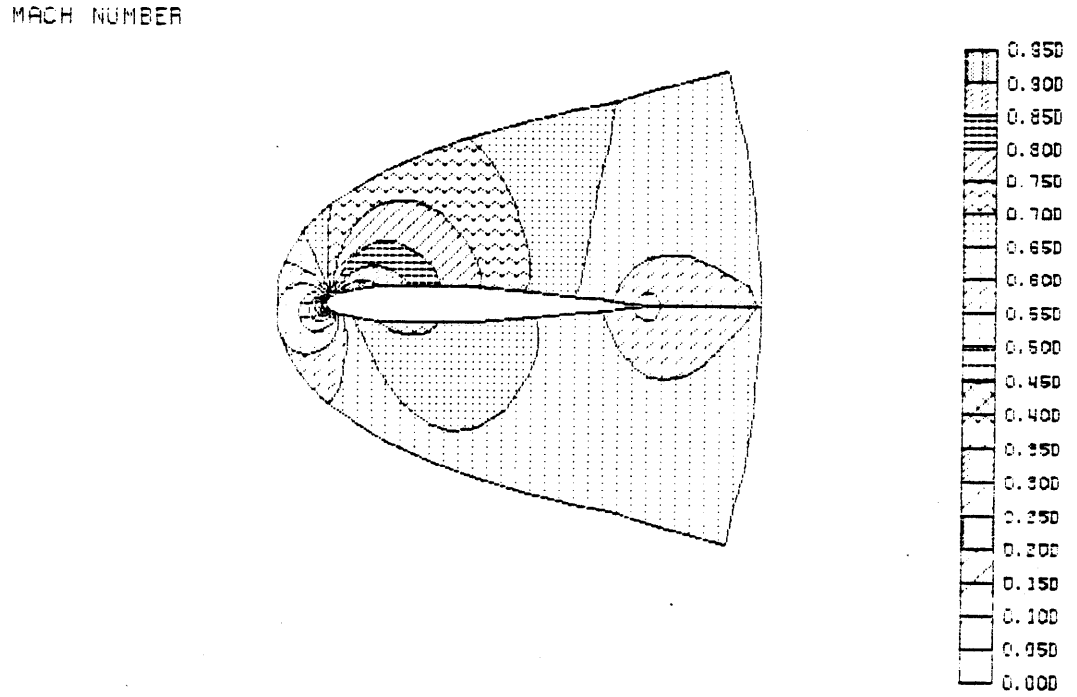

Figure 3-21c. Mach number contours.

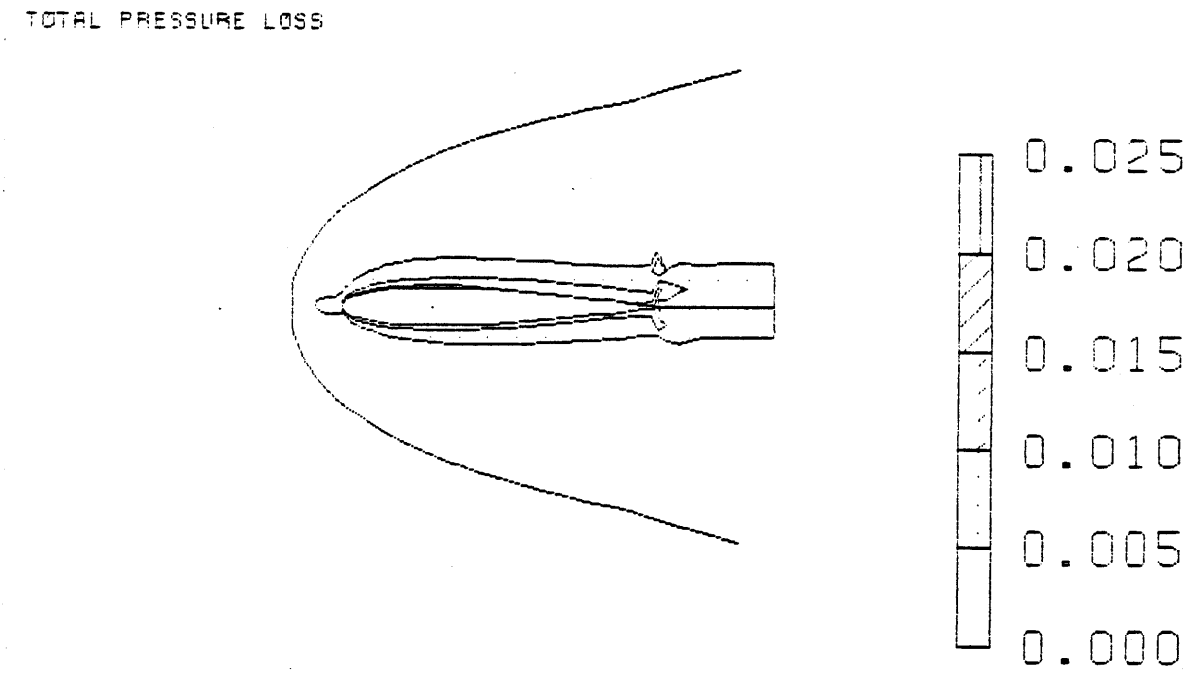

Figure 3-21d. Total pressure loss contours.

Figure 3-21. NACA0012 airfoil for M = 0.63 and 2.0 degree angle of attack. Multiple-grid solution on 97*17 C-type mesh with 3 global levels. [run 189]

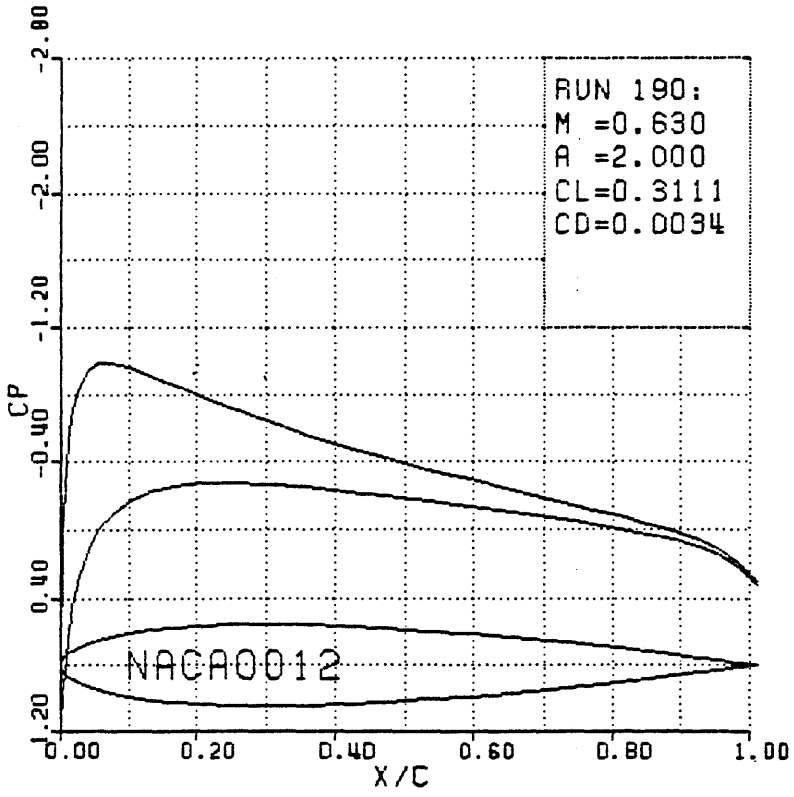

Figure 3-22a. Smoothing coefficient of 0.08.

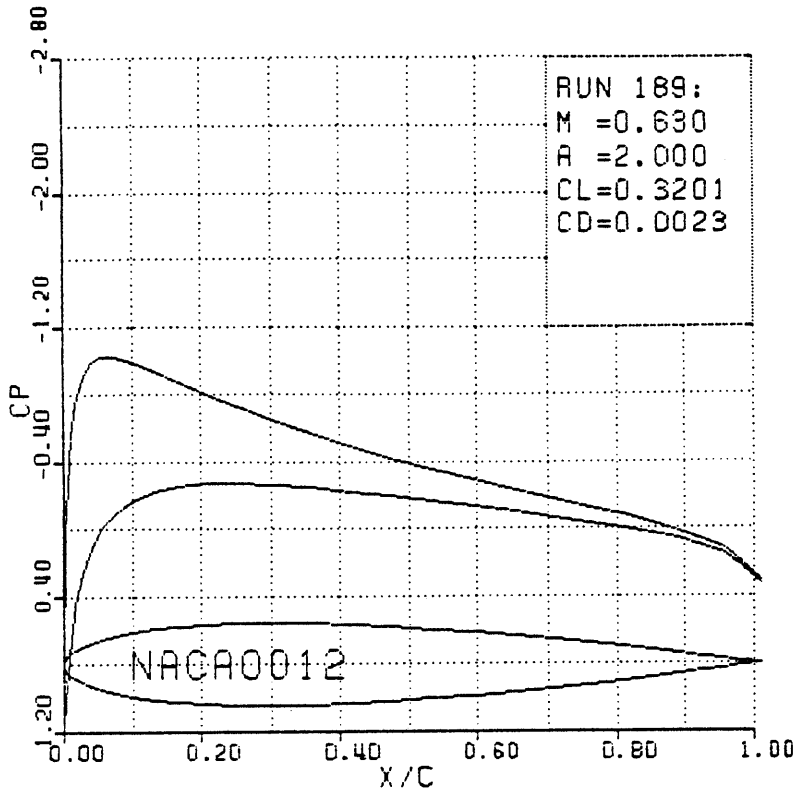

Figure 3-22b. Smoothing coefficient of 0.05.

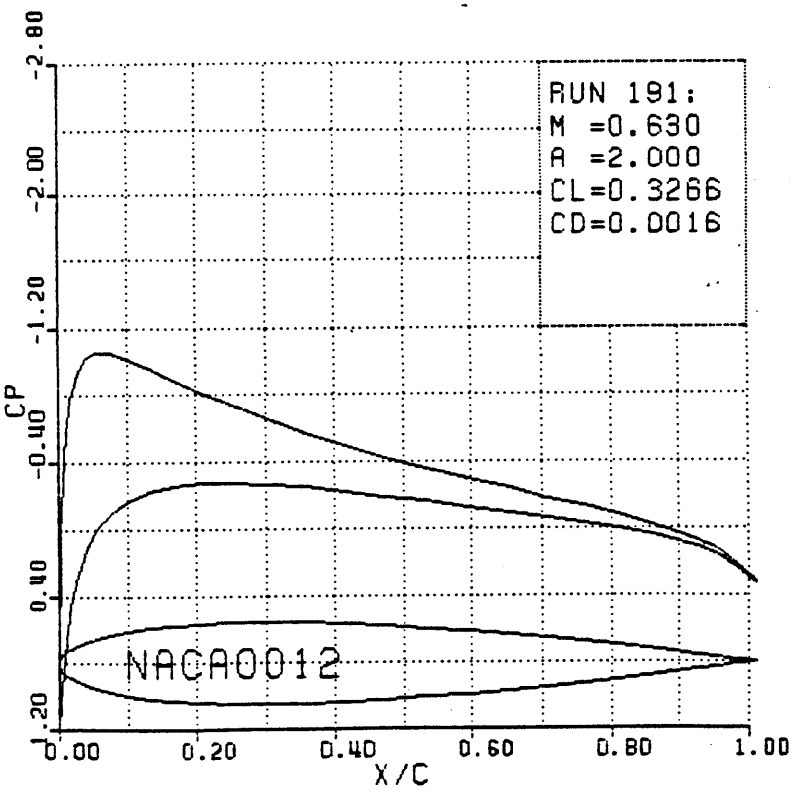

Figure 3-22c. Smoothing coefficient of 0.03.

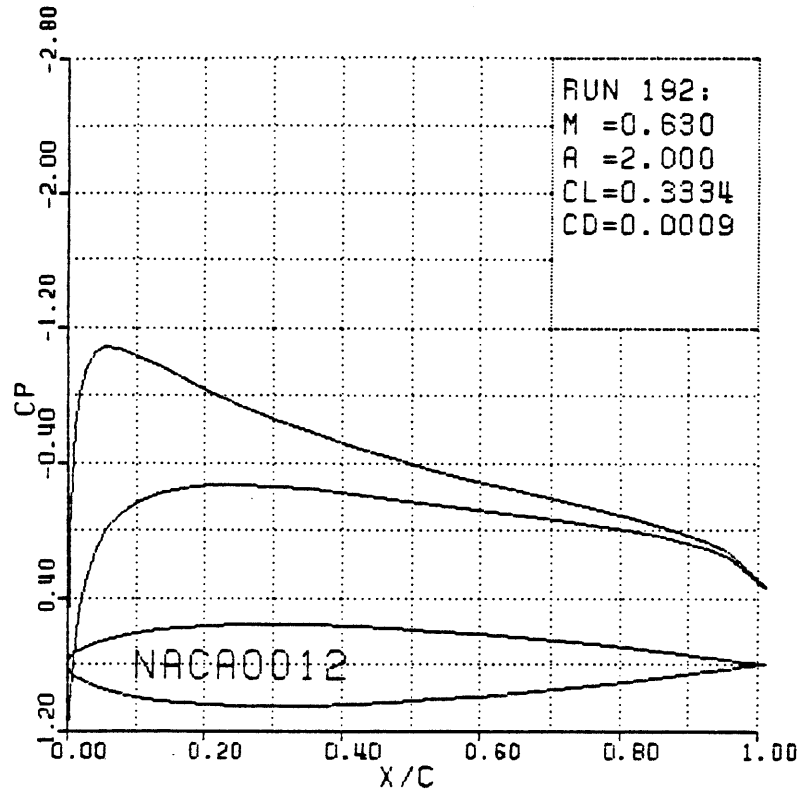

Figure 3-22d. Smoothing coefficient of 0.01.

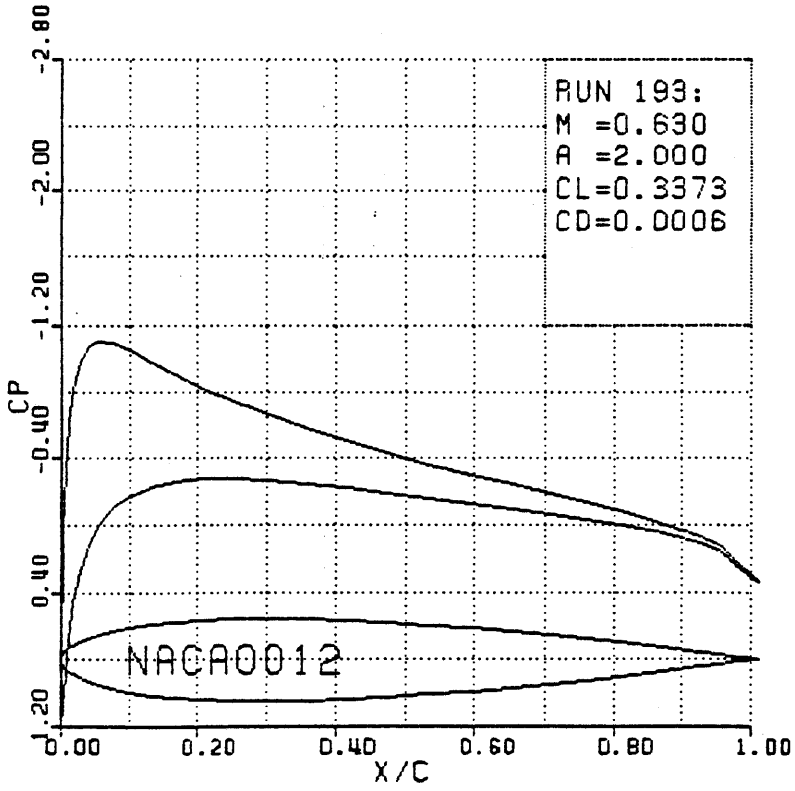

Figure 3-22e. Smoothing coefficient of 0.0.

Figure 3-22. Comparison of surface pressure coefficient for various smoothing coefficient values for C-mesh. [runs 190,189,191,192,193]

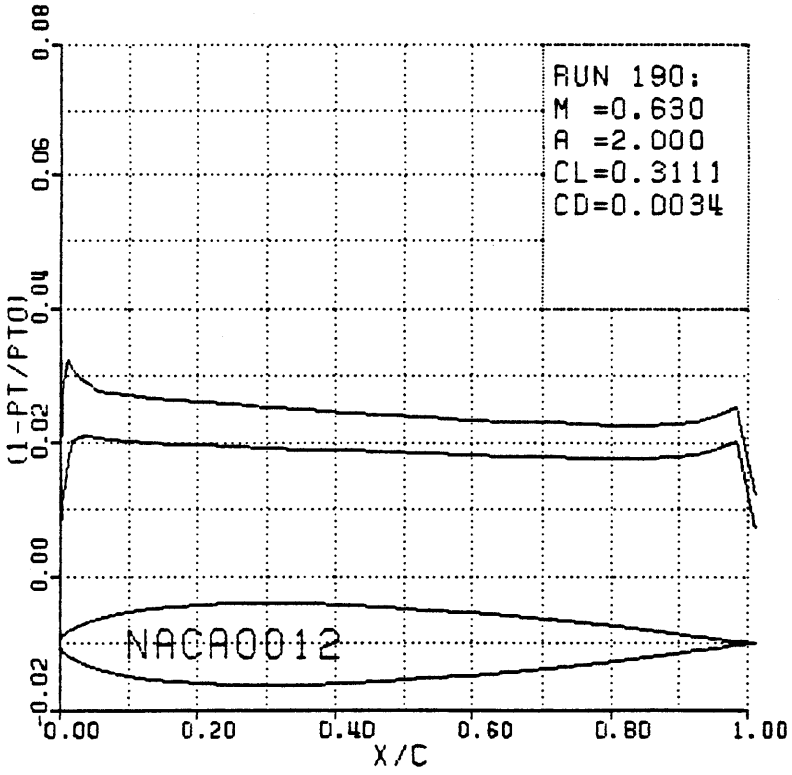

Figure 3-23a. Smoothing coefficient of 0.08.

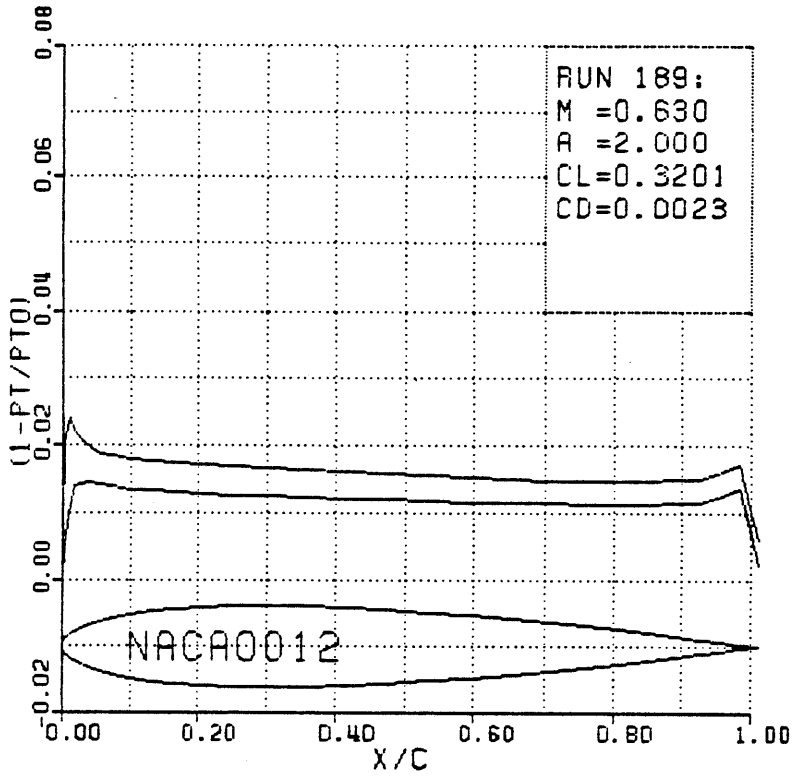

Figure 3-23b. Smoothing coefficient of 0.05.

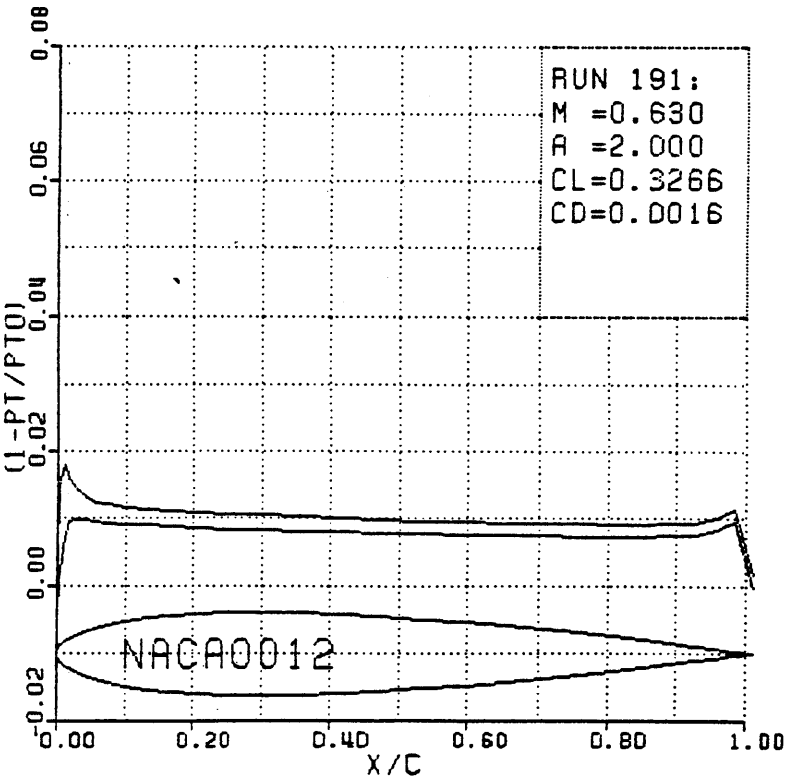

Figure 3-23c. Smoothing coefficient of 0.03.

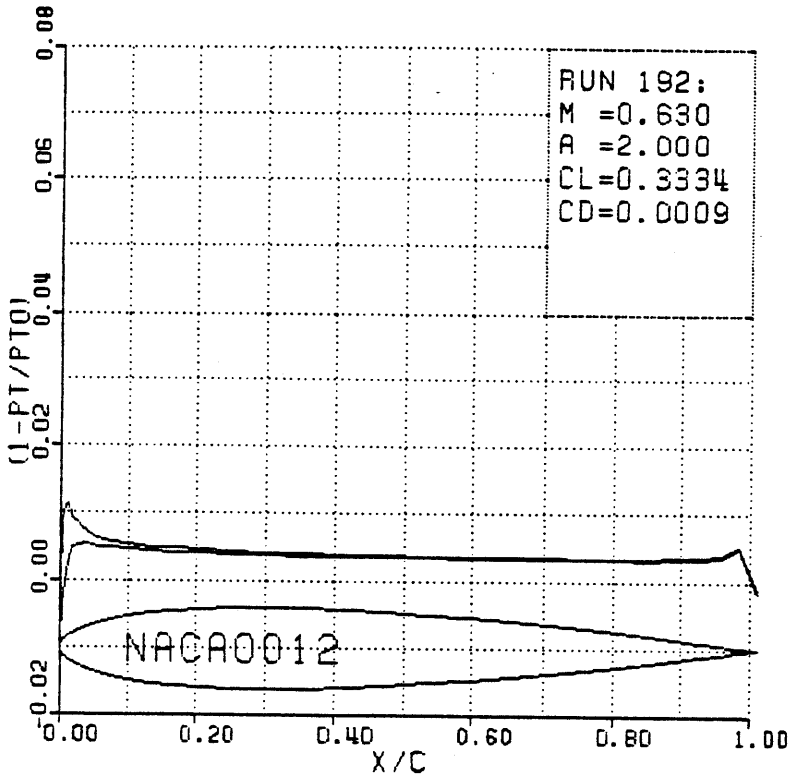

Figure 3-23d. Smoothing coefficient of 0.01.

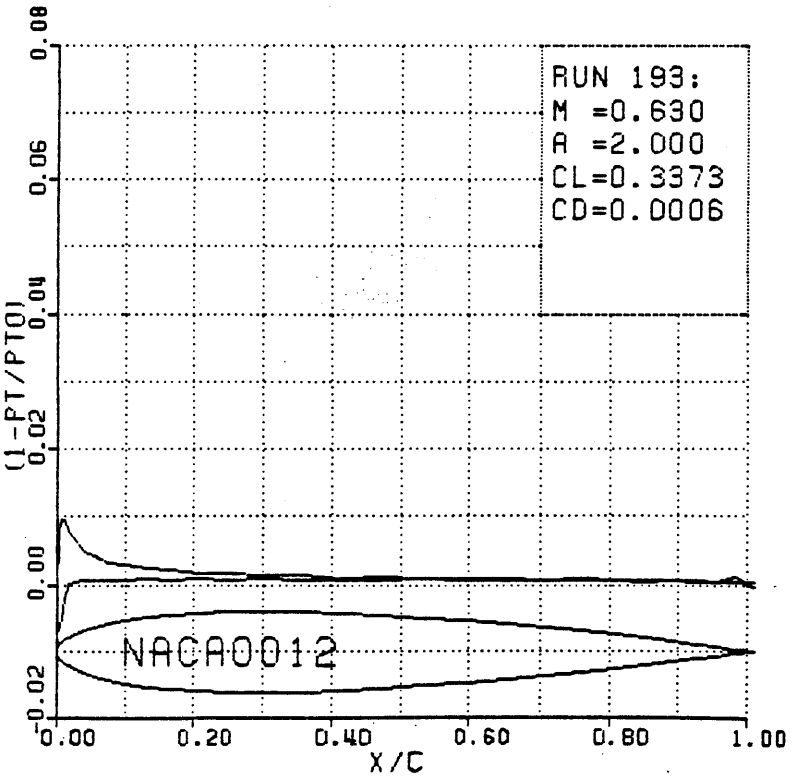

Figure 3-23e. Smoothing coefficient of 0.0.

Figure 3-23. Comparison of surface total pressure loss for various smoothing coefficient values for C-mesh. [runs 190,189,191,192,193]

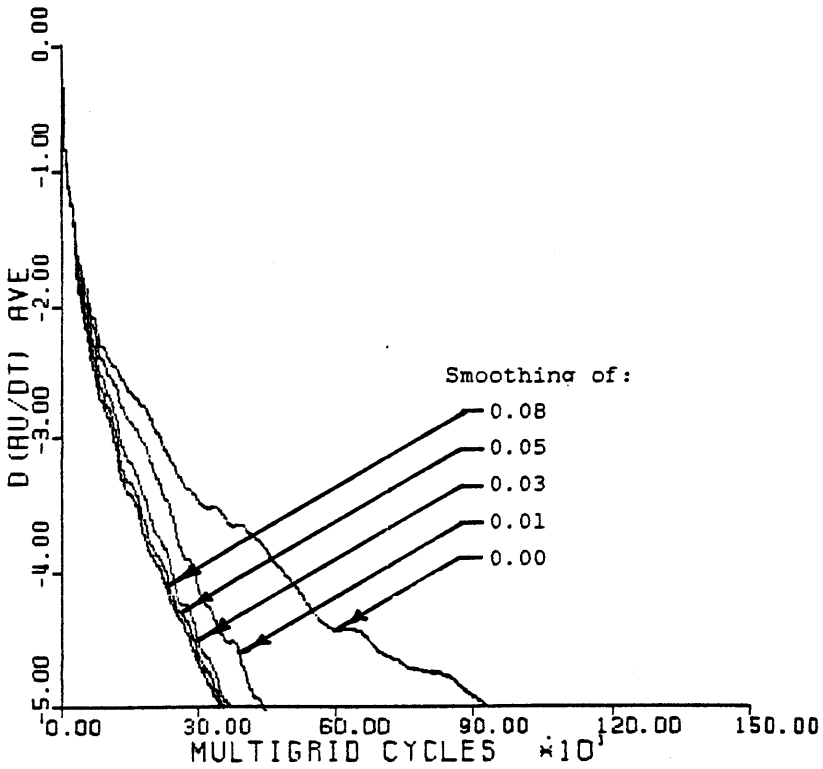

Figure 3-24. Comparison of convergence rates as a function of the smoothing coefficient for the C-type mesh. [runs 190,189,191,192,193]

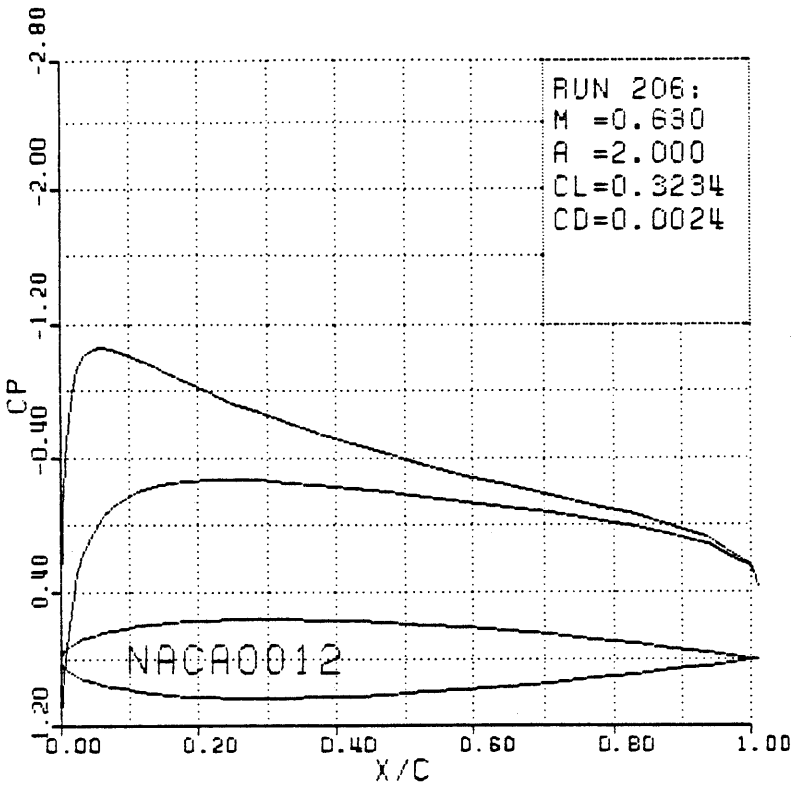

Figure 3-25a. Surface pressure coefficient.

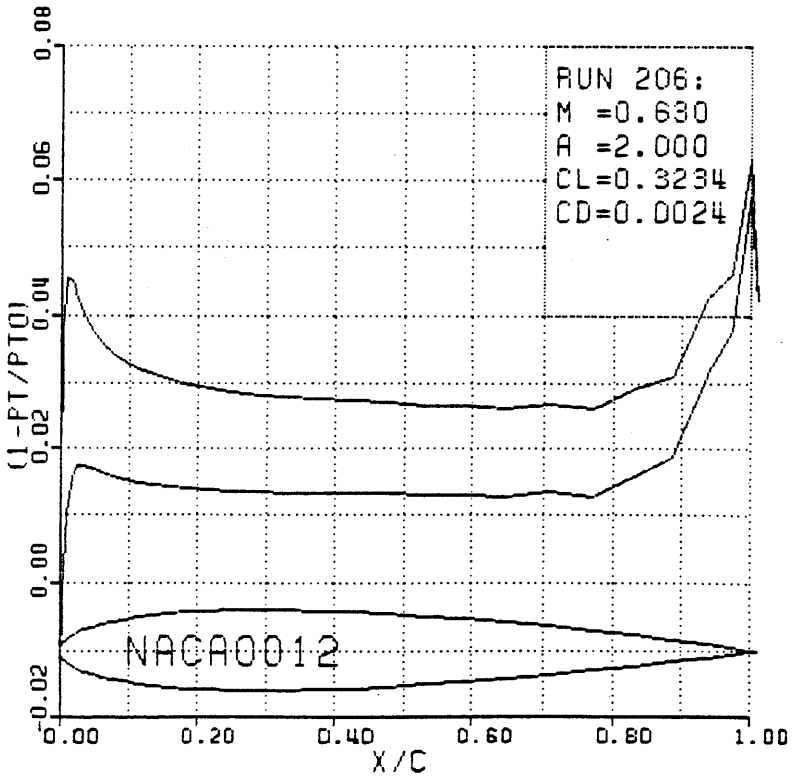

Figure 3-25b. Surface total pressure loss.

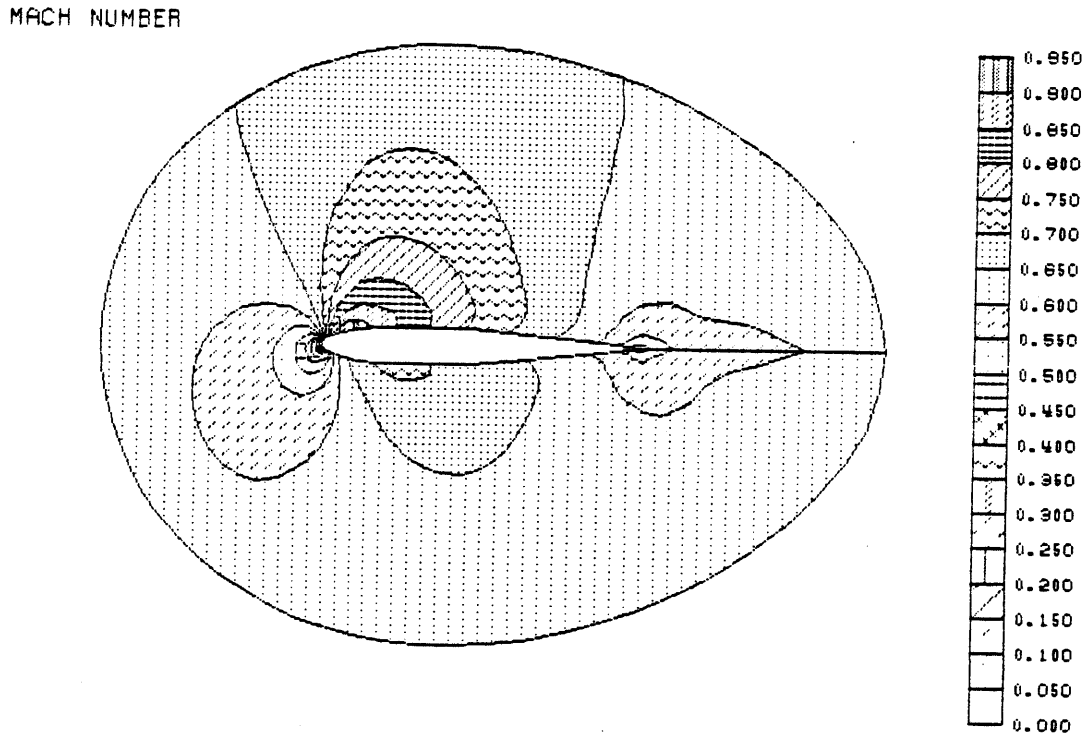

Figure 3-25c. Mach number contours.

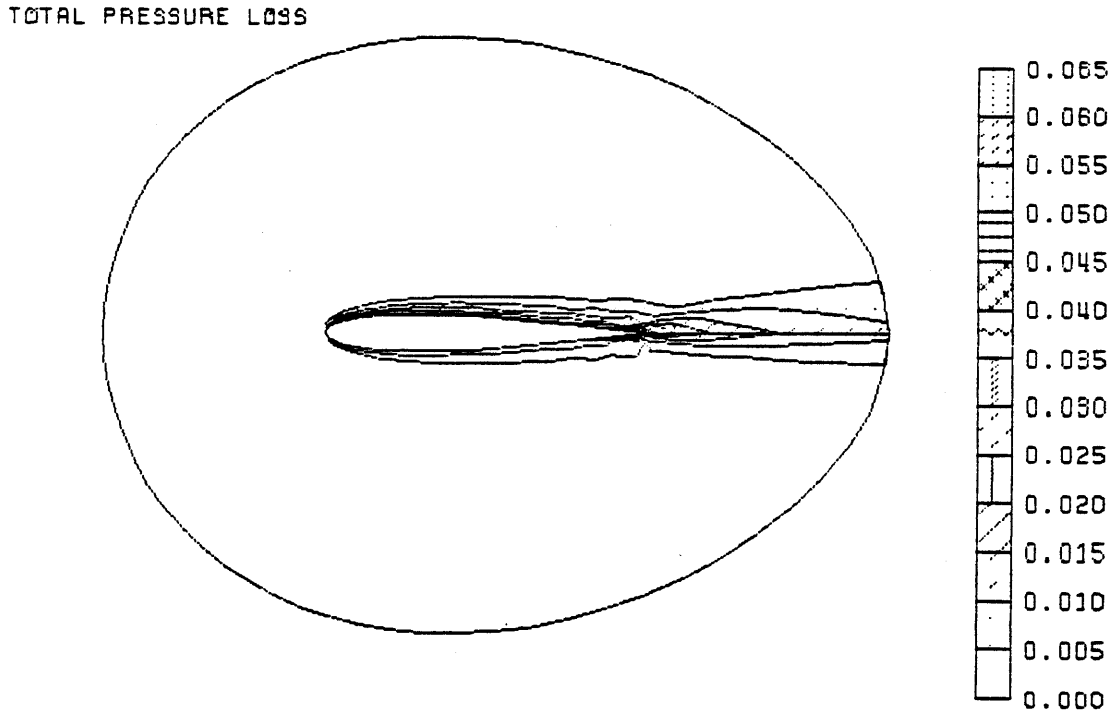

Figure 3-25d. Total pressure loss contours.

Figure 3-25. NACA0012 airfoil for M = 0.63 and 2.0 degrees angle of attack. Multiple-grid solution on 65*17 O-type mesh with 3 global levels. No doubling of predicted wall changes.

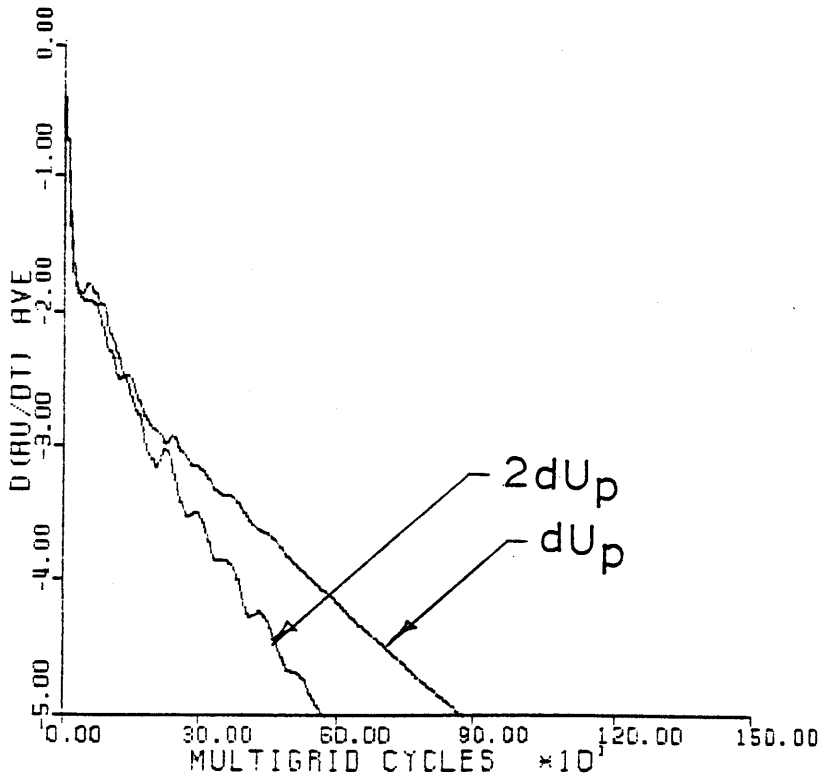

Figure 3-26. Comparison of convergence histories for solutions of figures 3-10 and 3-25.

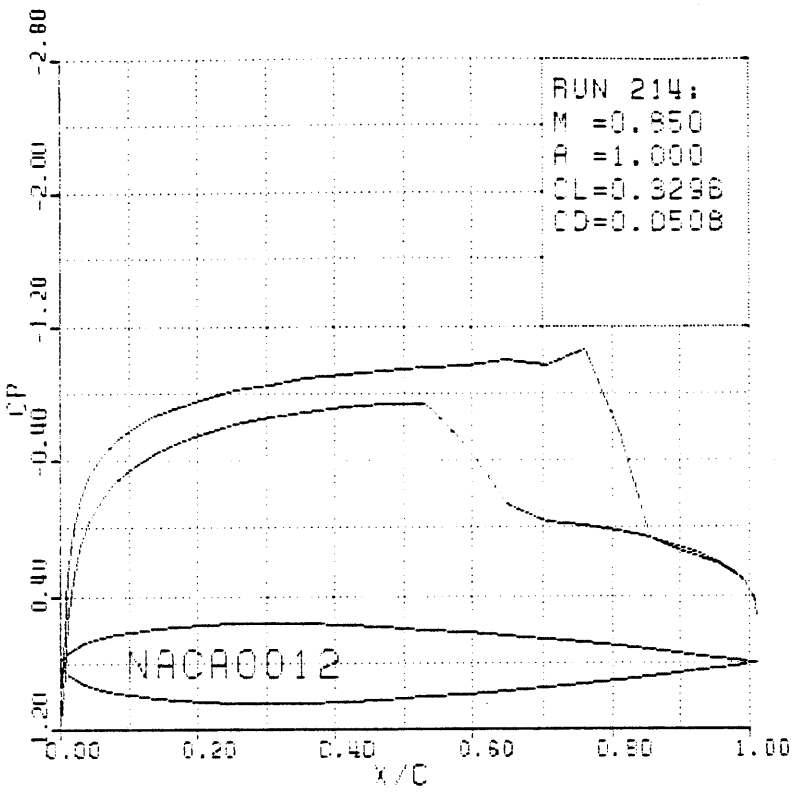

Figure 3-27a. Surface pressure coefficient.

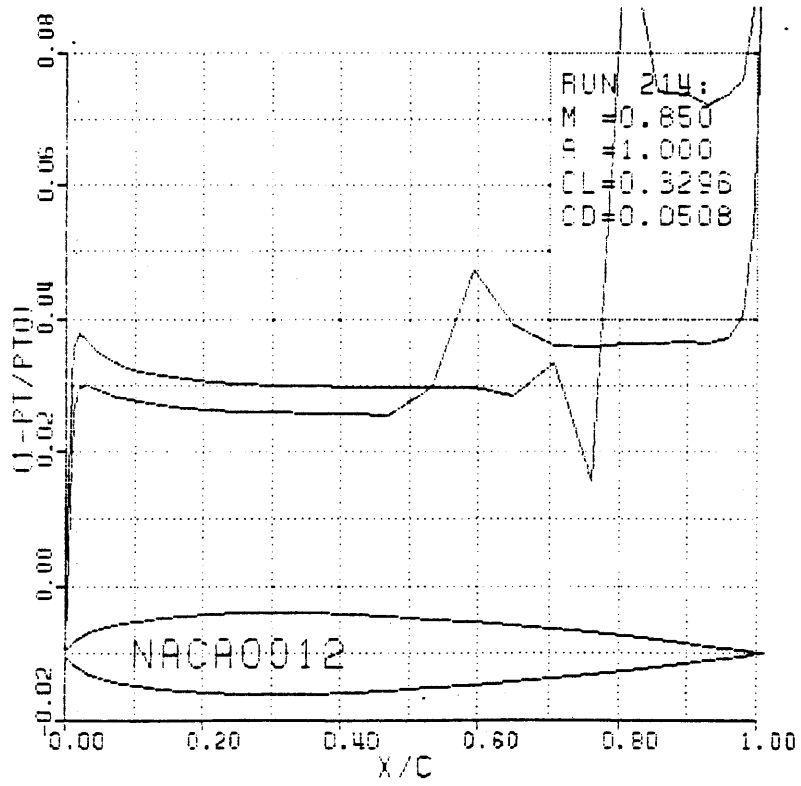

Figure 3-27b. Surface total pressure loss.

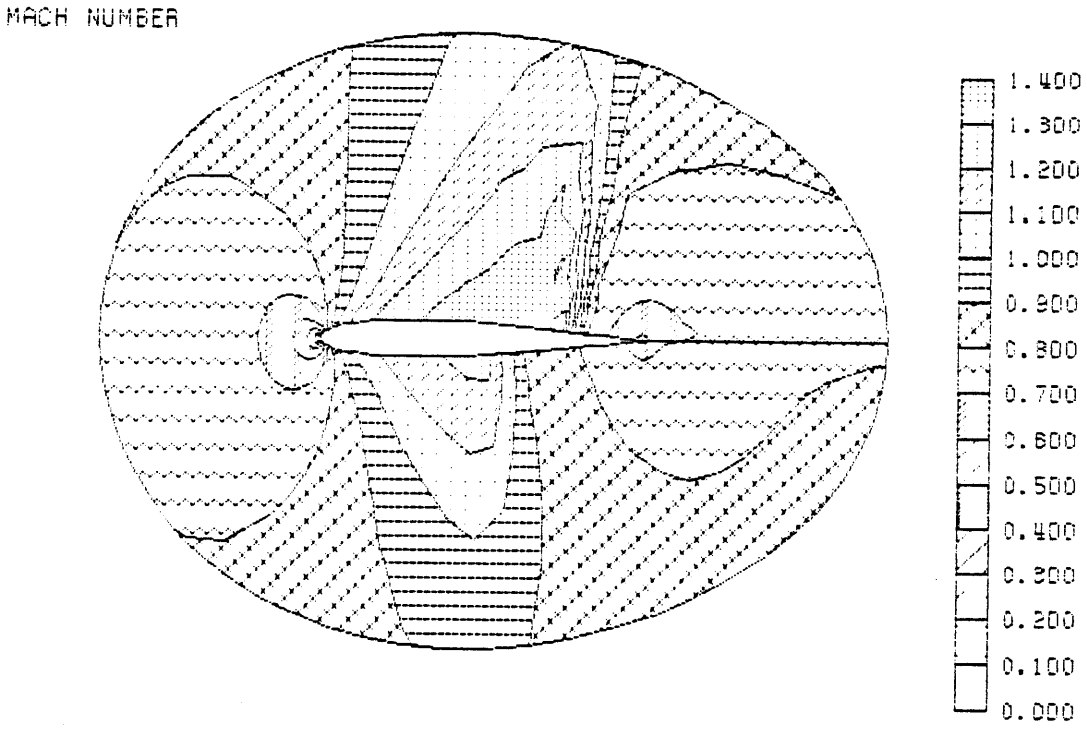

Figure 3-27c. Mach number contours.

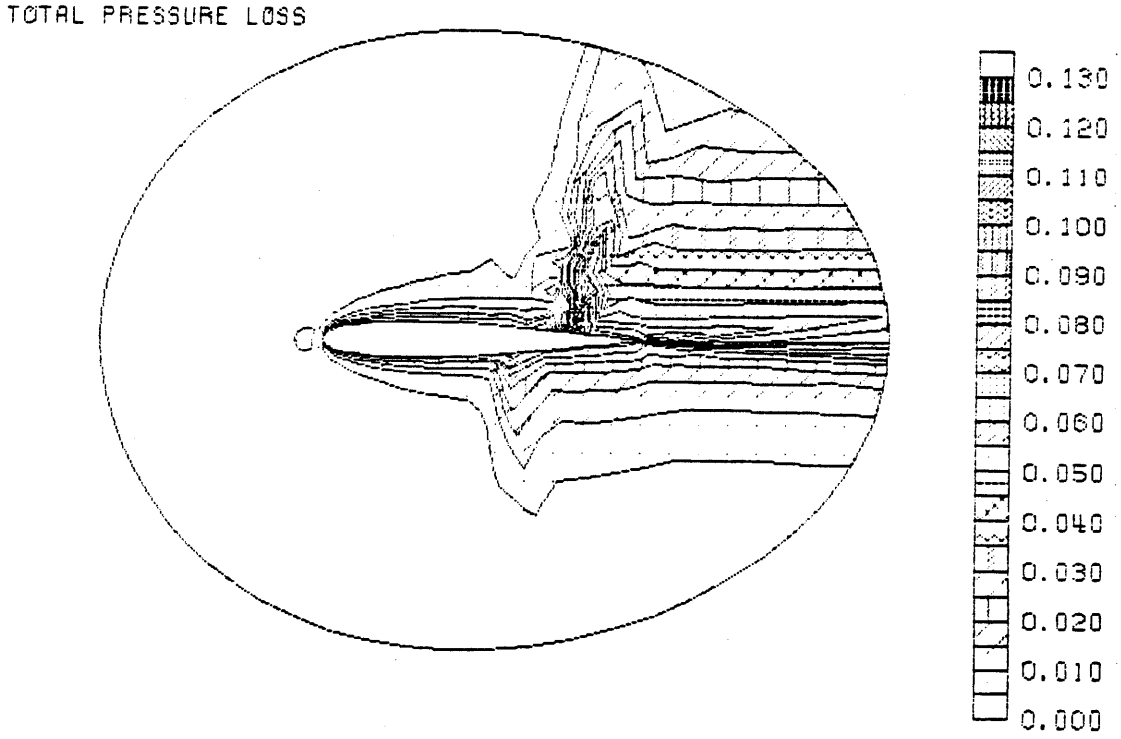

Figure 3-27d. Total pressure loss contours.

Figure 3-27. NACA0012 airfoil for M = 0.85 and 1.0 degree angle of attack. Multiple-grid solution on 65*17 O-type mesh with 3 global levels. [Run 214]

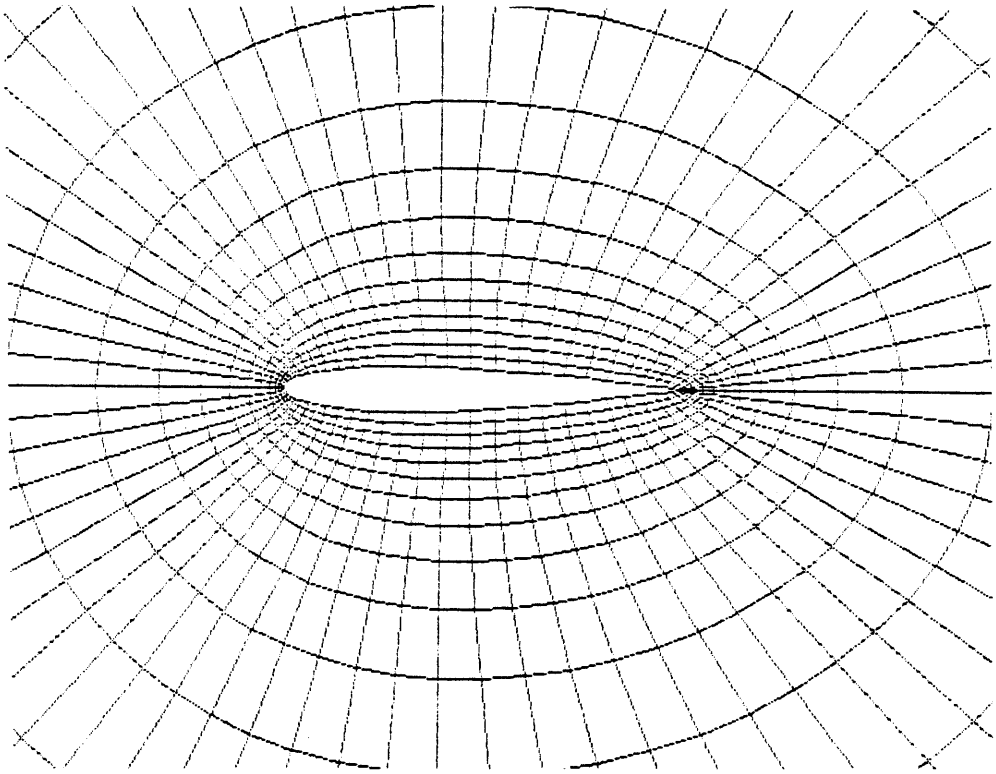

Figure 3-28. Near field of NACA0012 airfoil for 65*17 global O-type mesh. [Run 214]

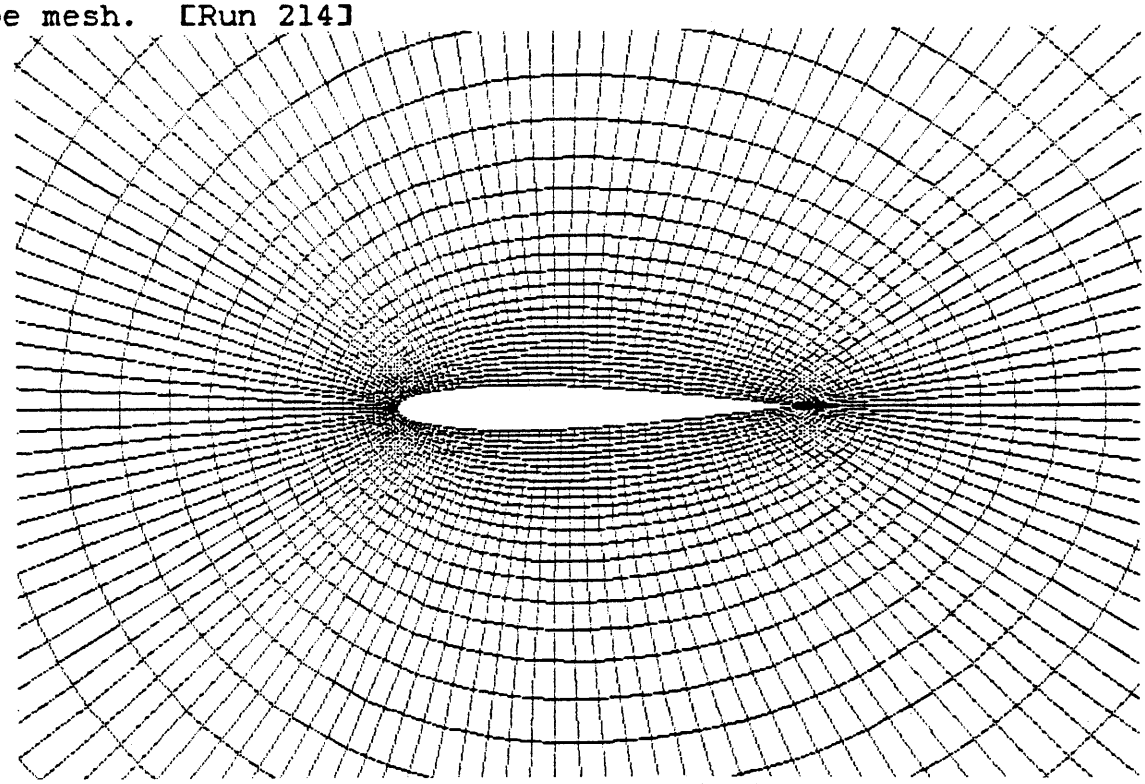

Figure 3-29. Near field of NACA0012 airfoil for 129*33 global O-type mesh. [Run 216]

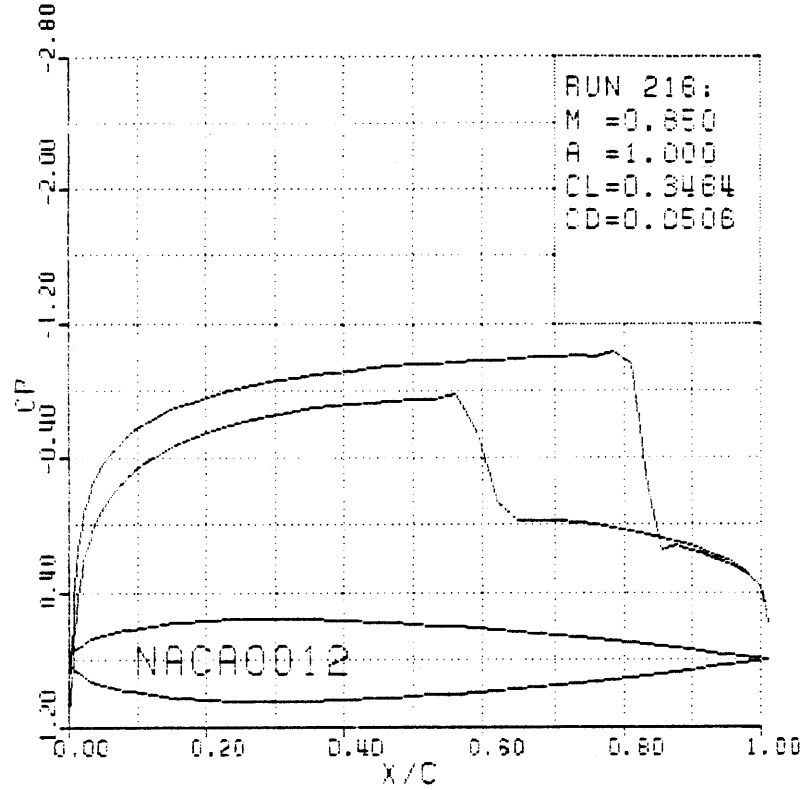

Figure 3-30a. Surface pressure coefficient.

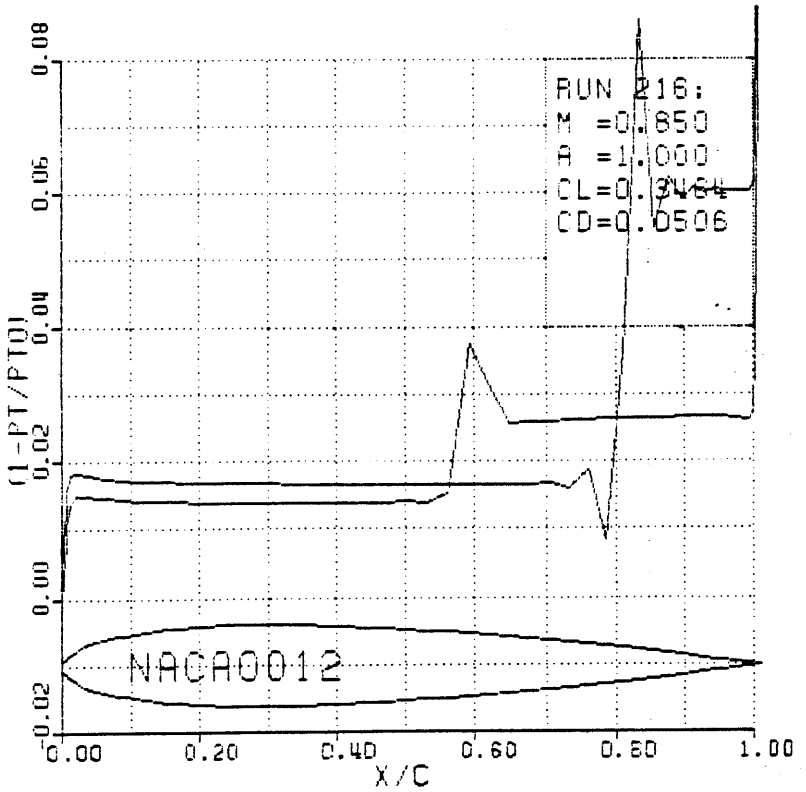

Figure 3-30b. Surface total pressure loss.

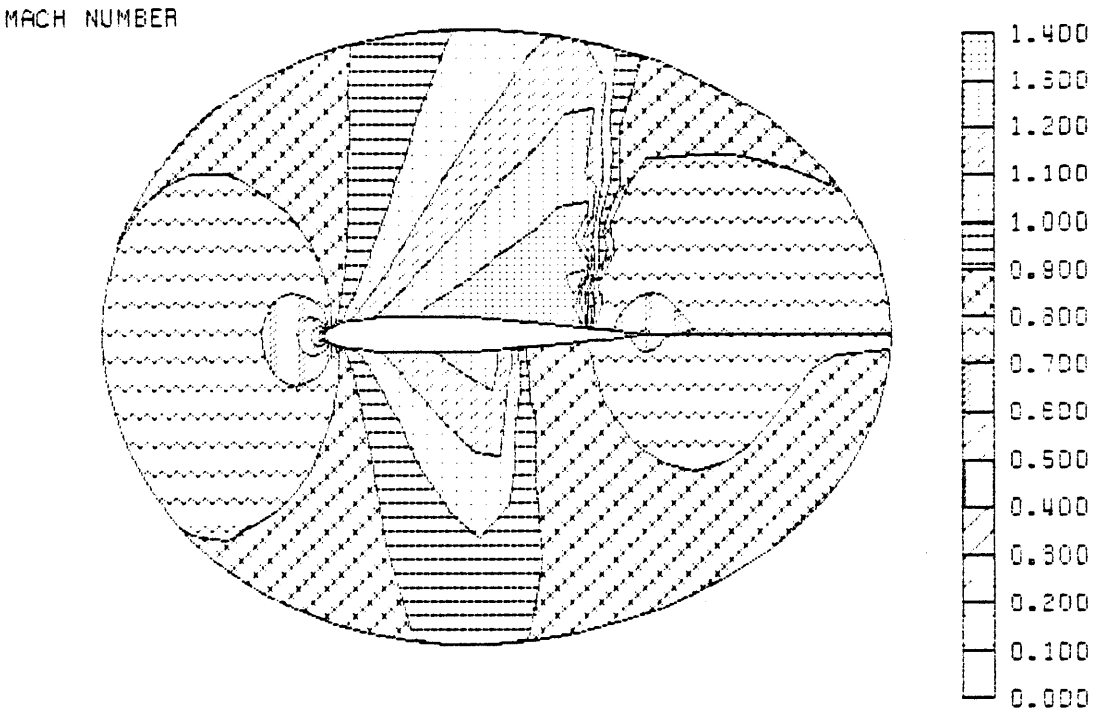

Figure 3-30c. Mach number contours.

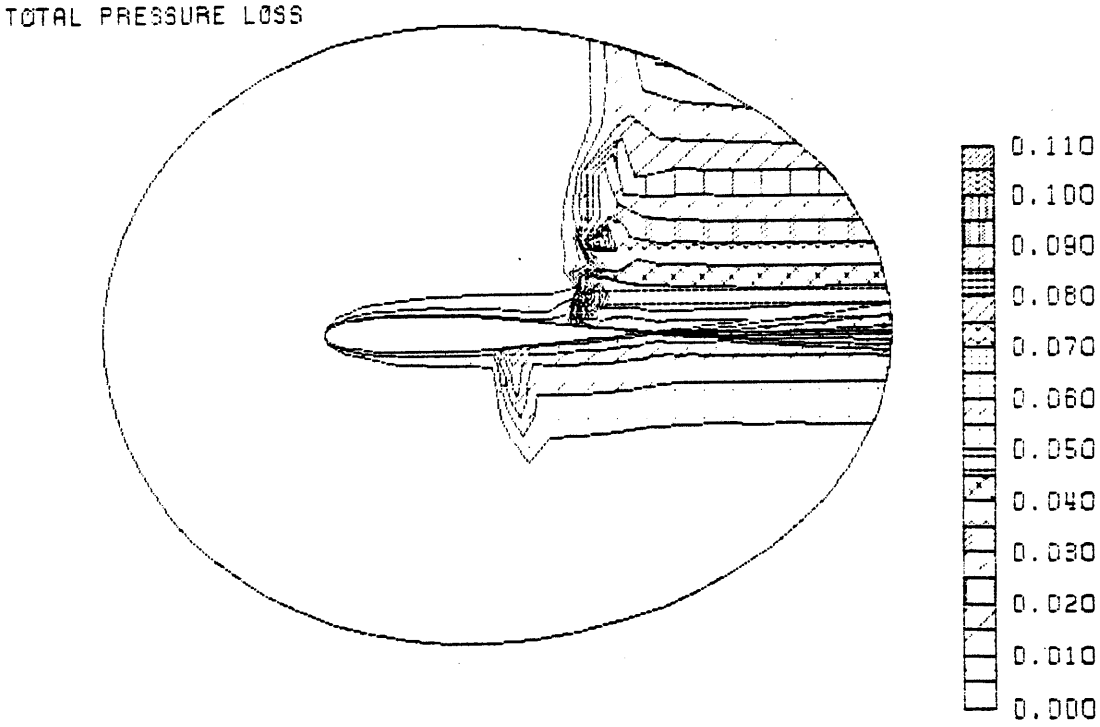

Figure 3-30d. Total pressure loss contours.

Figure 3-30. NACA0012 airfoil for M = 0.85 and 1.0 degree angle of attack. Multiple-grid solution on 129*33 O-type mesh with 4 global levels. [Run 216]

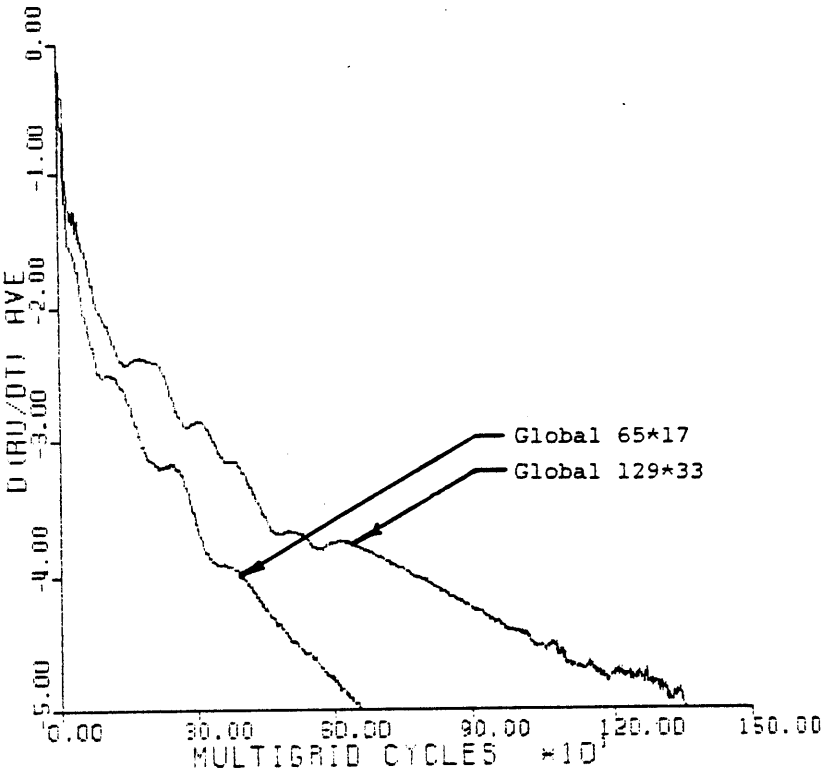

Figure 3-31. Comparison of convergence histories for solutions of figures 3-24 and 3-30. [Runs 214 and 216]

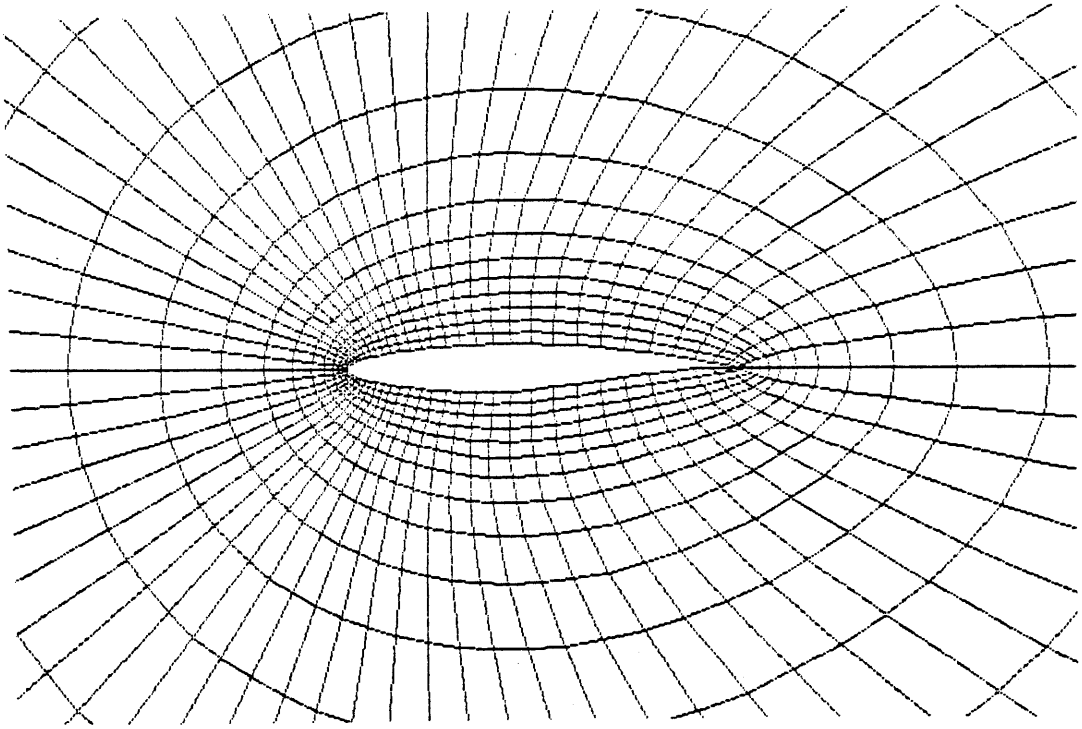

Figure 3-32. Near field of RAE2822 airfoil for 65*17 global O-type mesh. [Run 211]

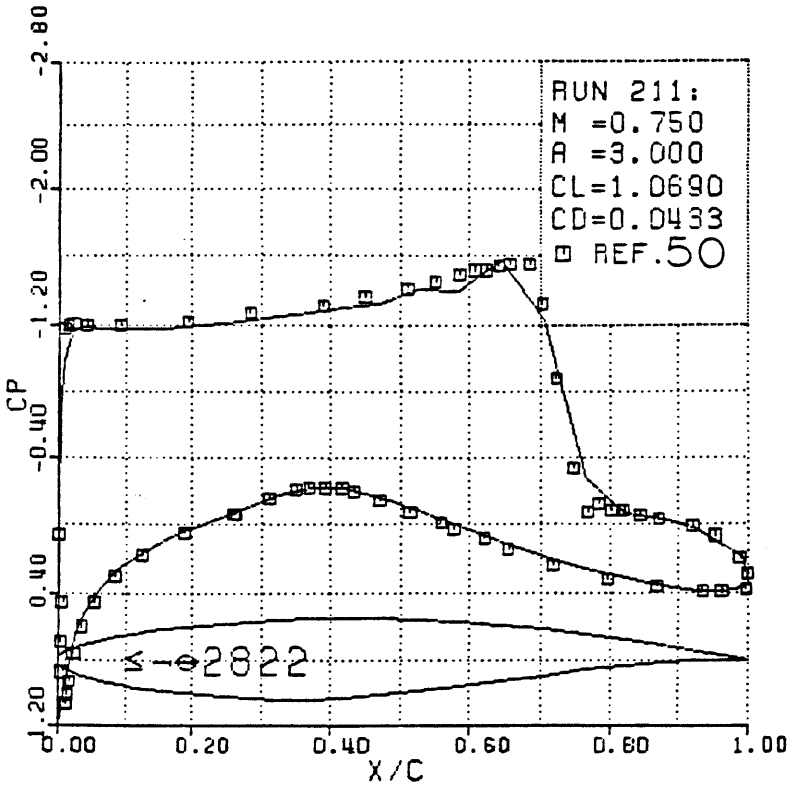

Figure 3-33a. Surface pressure coefficient.

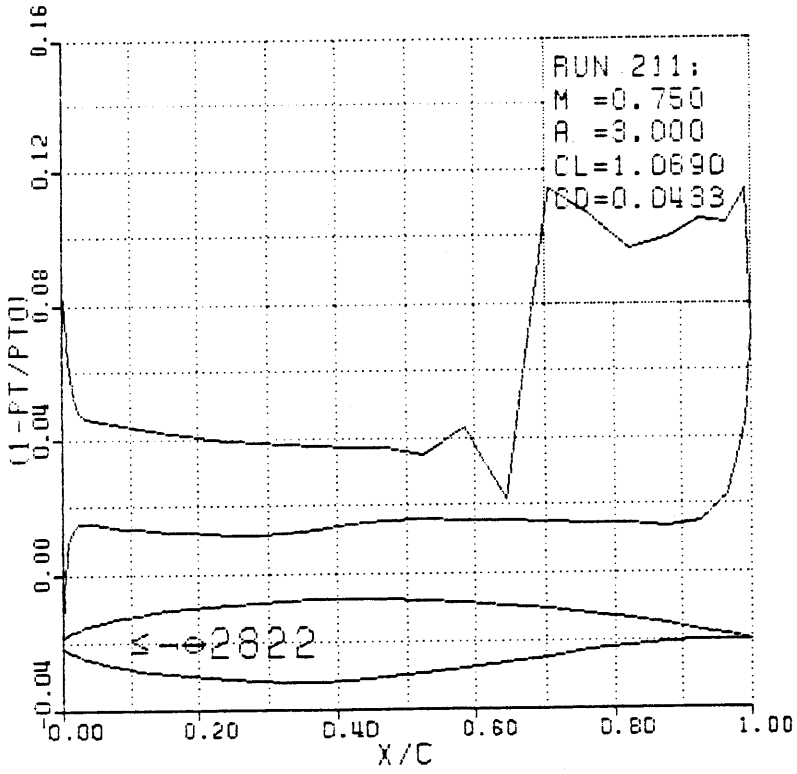

Figure 3-33b. Surface total pressure loss.

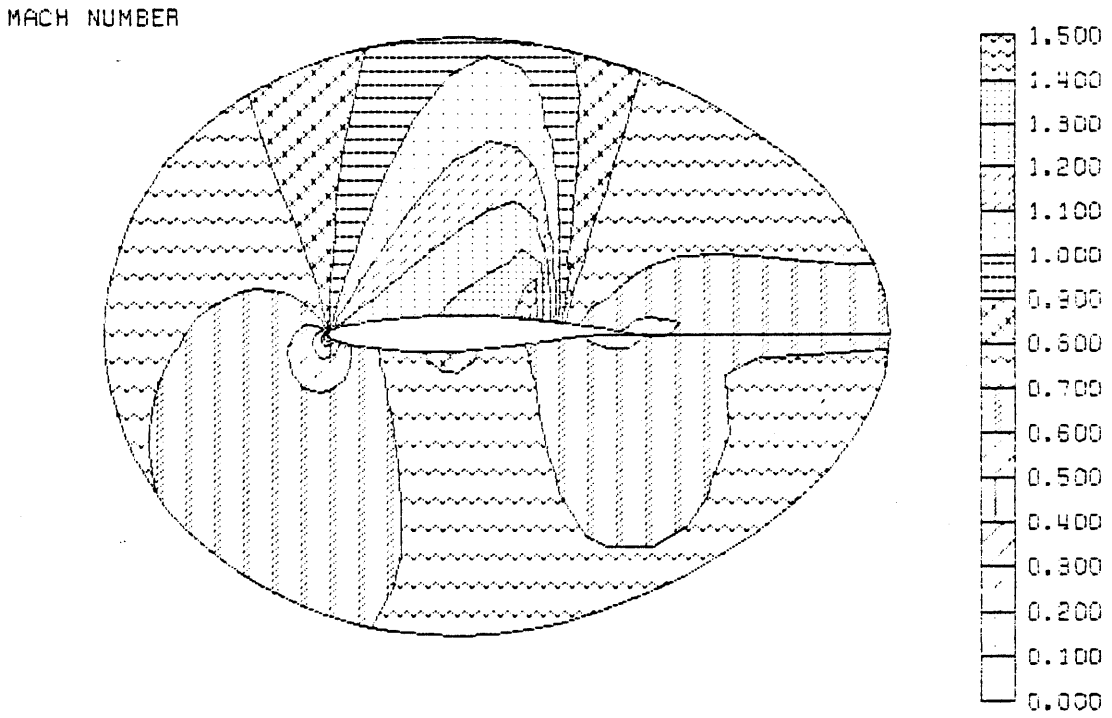

Figure 3-33c. Mach number contours.

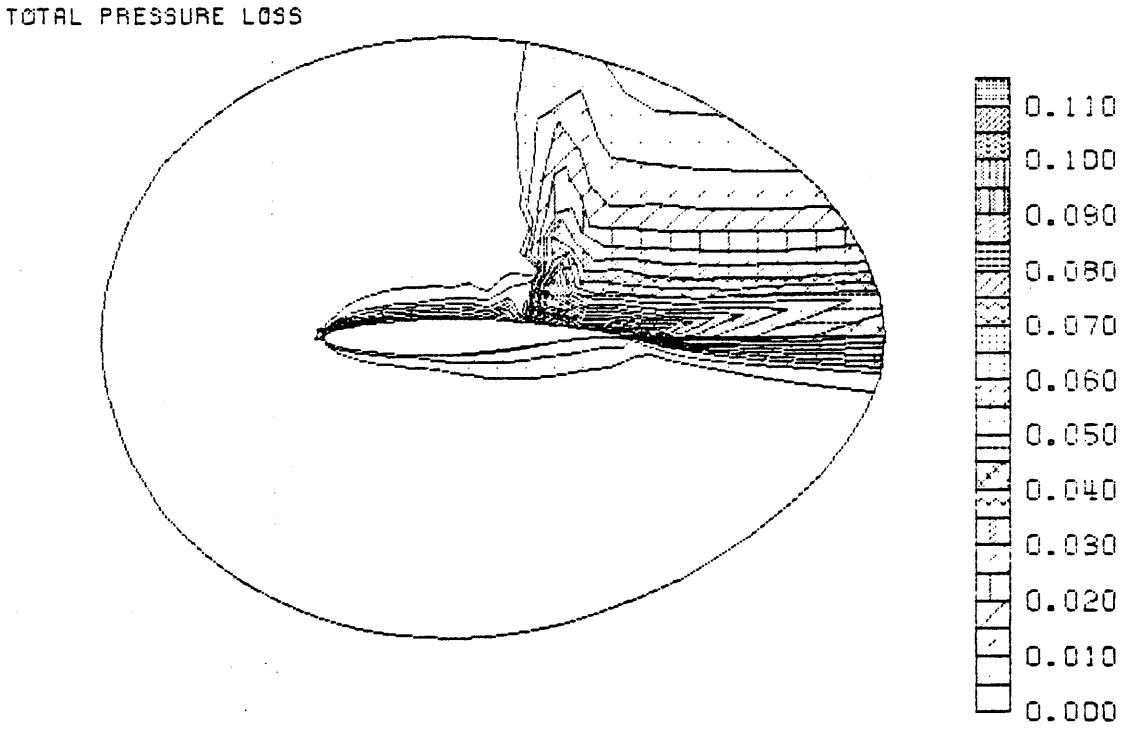

Figure 3-33d. Total pressure loss contours.

Figure 3-33. RAE2822 airfoil for M = 0.75 and 3.0 degrees angle of attack. Multiple-grid solution on 65*17 O-type mesh with 3 global levels. [Run 211]

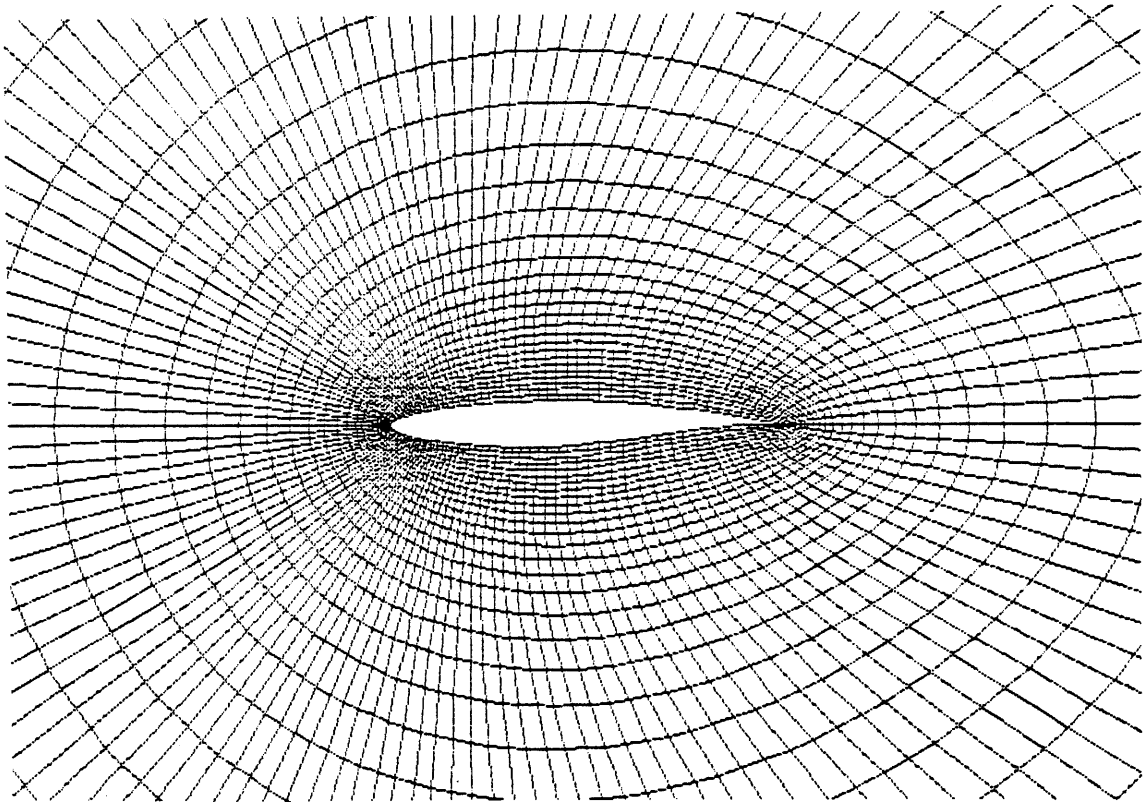

Figure 3-34. Near field of RAE2822 airfoil for 129*33 global O-type mesh. [Run 213]

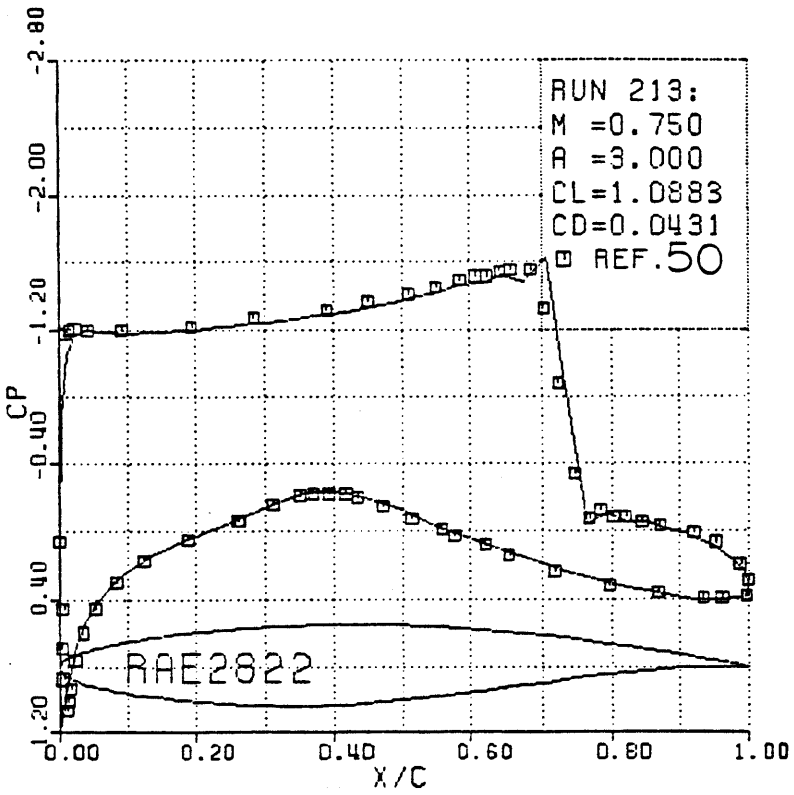

Figure 3-35a. Surface pressure coefficient.

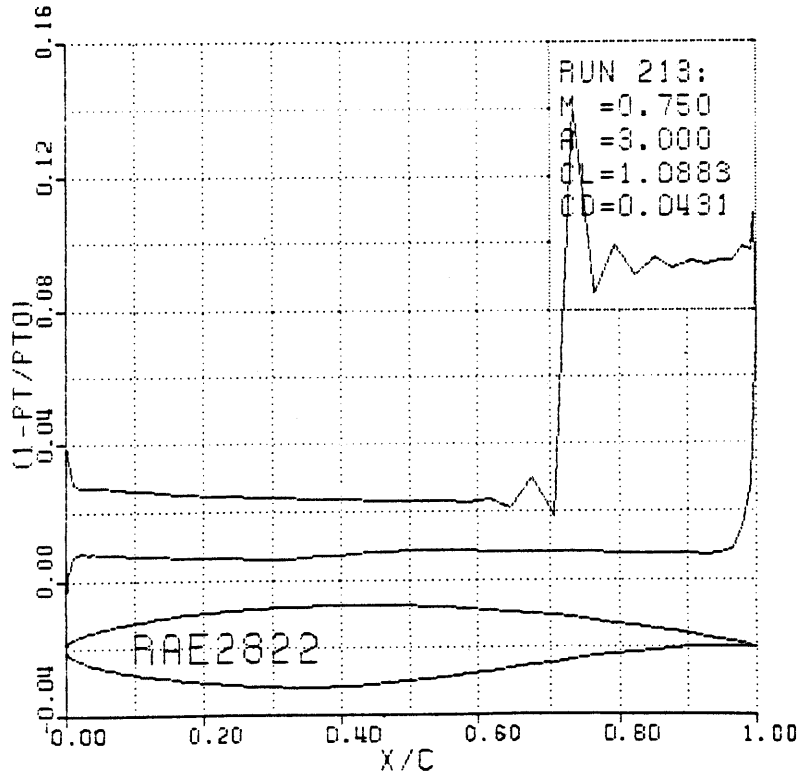

Figure 3-35b. Surface total pressure loss.

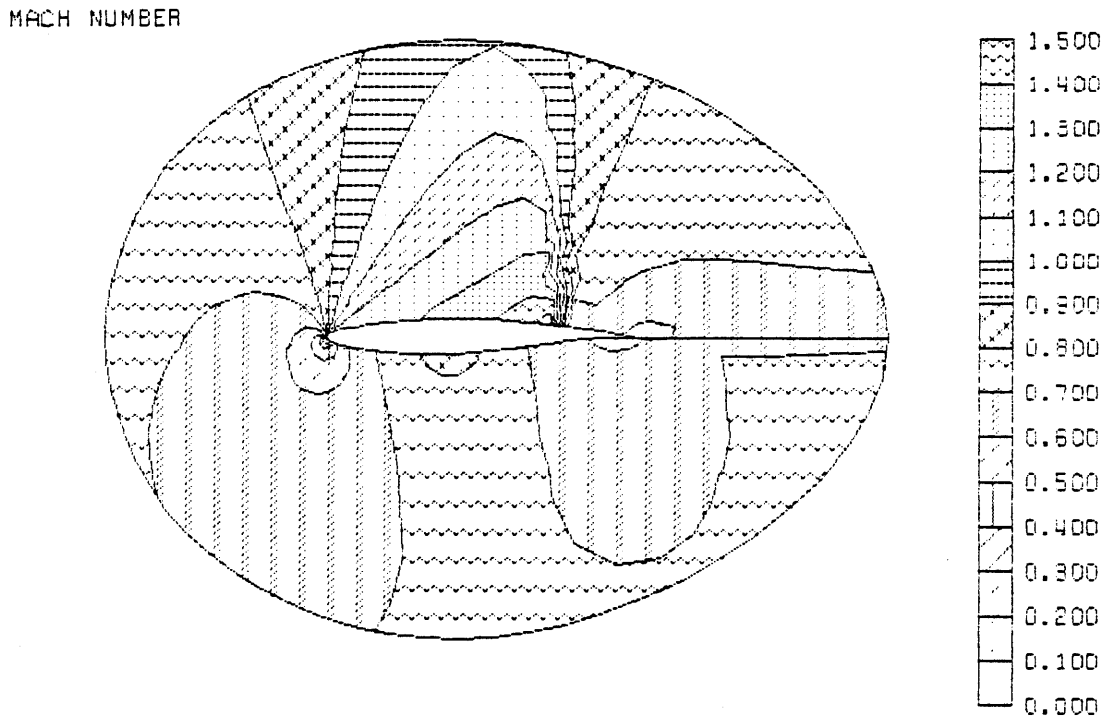

Figure 3-35c. Mach number contours.

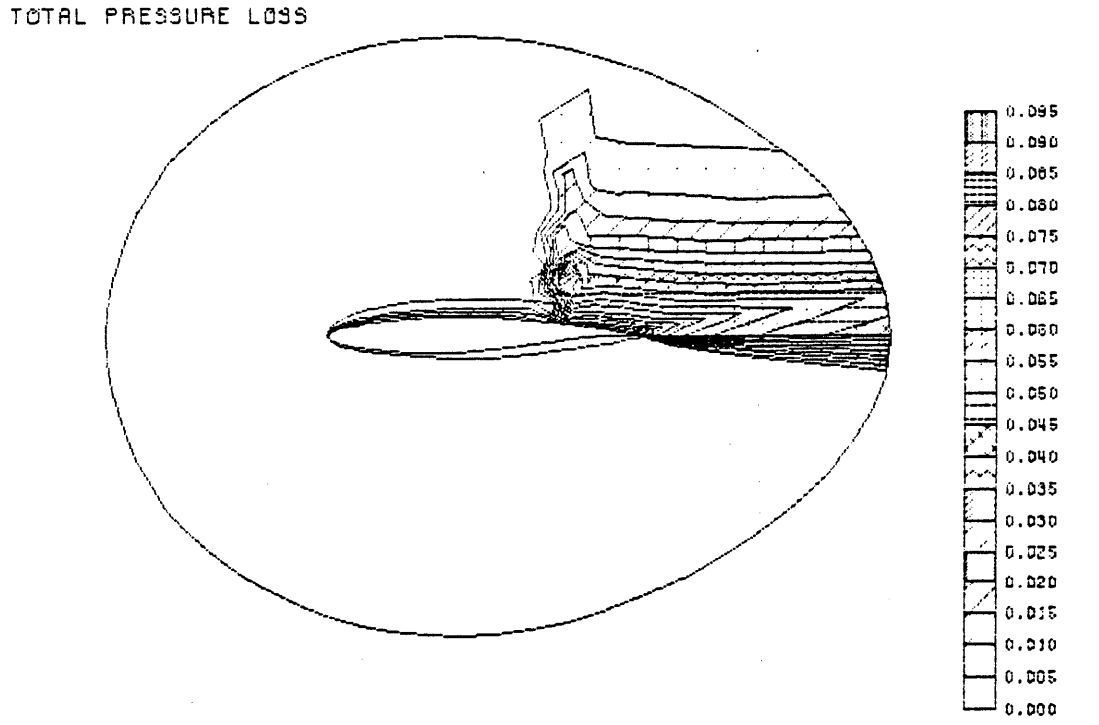

Figure 3-35d. Total pressure loss contours.

Figure 3-35. RAE2822 airfoil for M = 0.75 and 3.0 degrees angle of attack. Multiple-grid solution on 129*33 O-type mesh with 4 global levels. [Run 213]

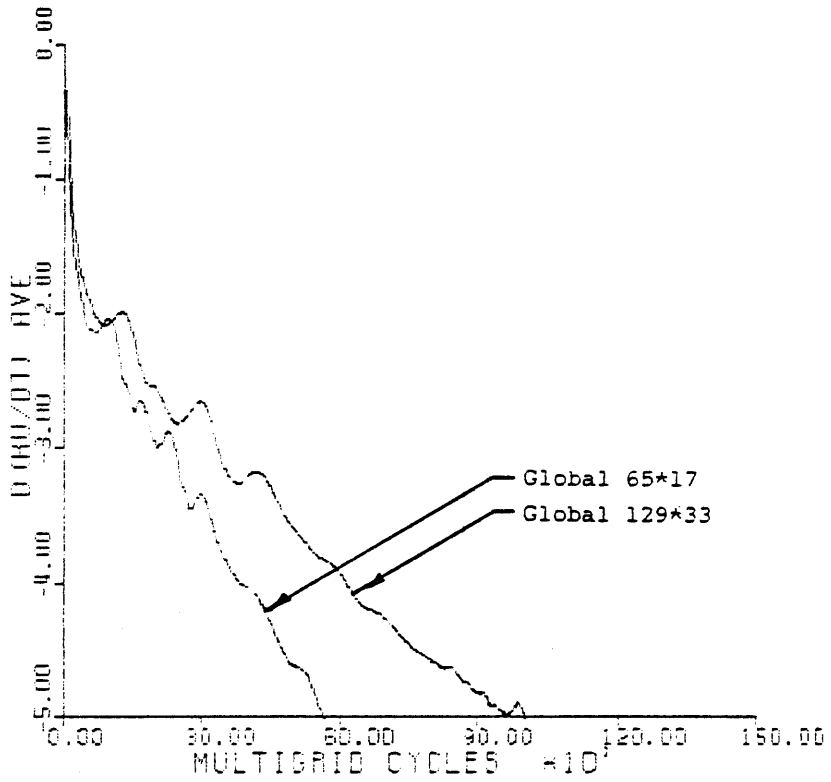

Figure 3-36. Comparison of convergence histories for solutions of figures 3-33 and 3-35. [Runs 211 and 213]

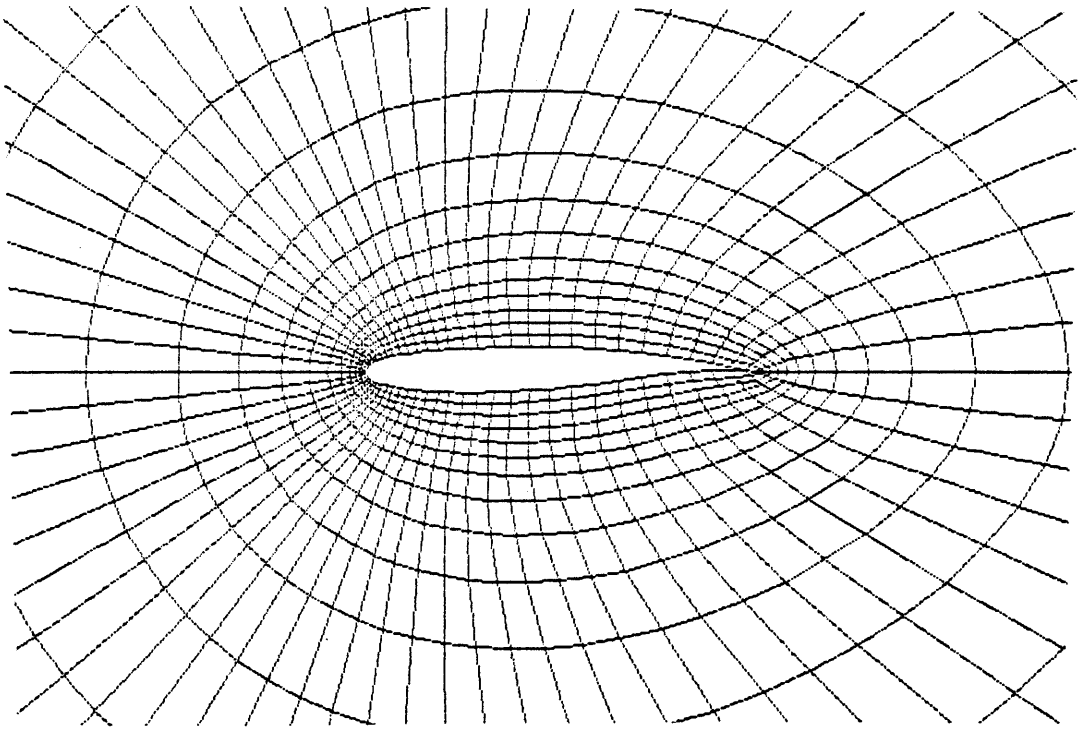

Figure 3-37. Near field of KORN airfoil for 65*17 global O-type mesh. [Run 208]

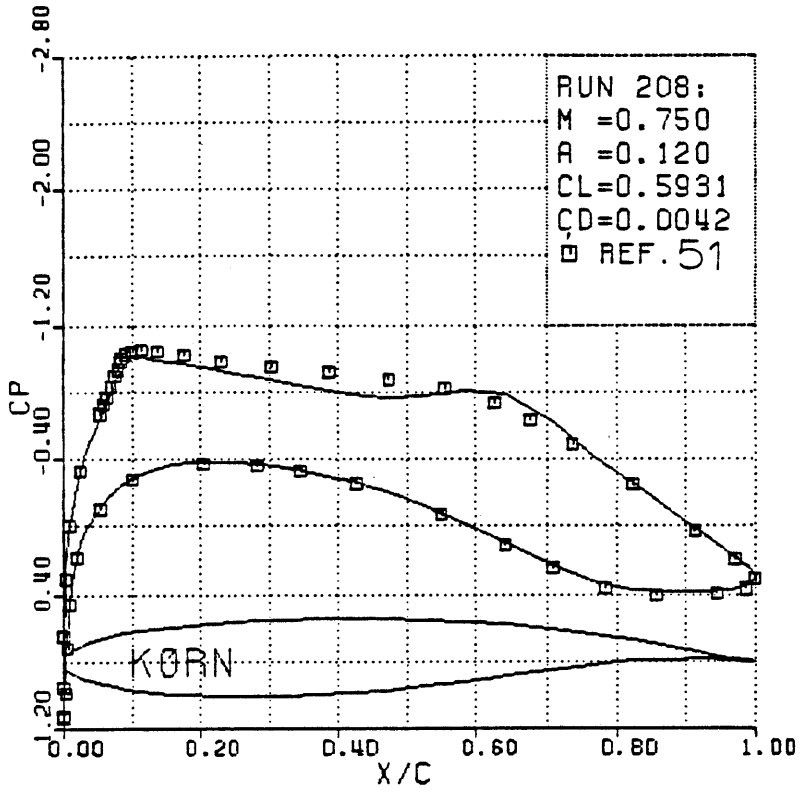

Figure 3-38a. Surface pressure coefficient.

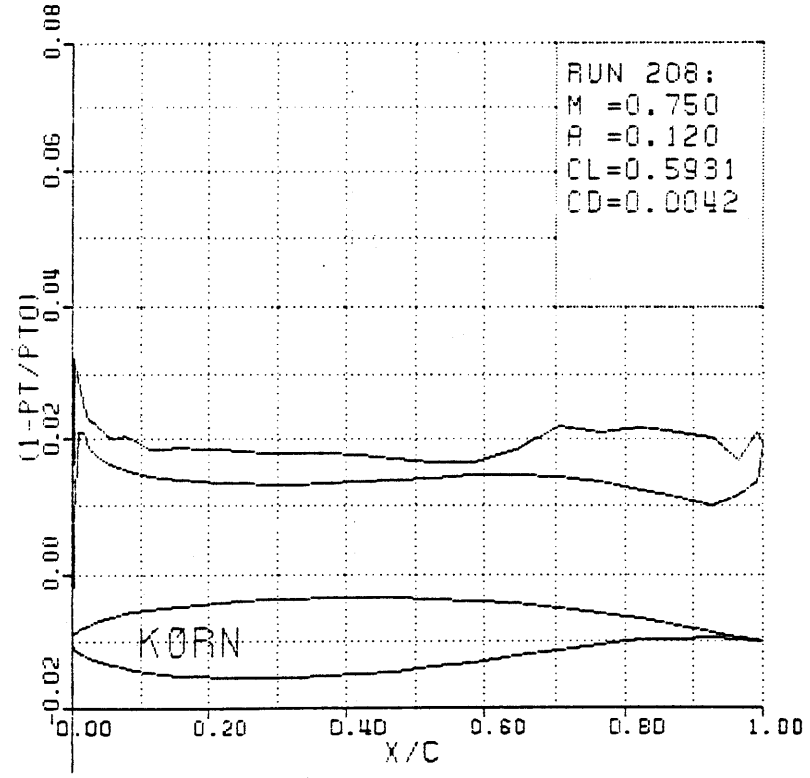

Figure 3-38b. Surface total pressure loss.

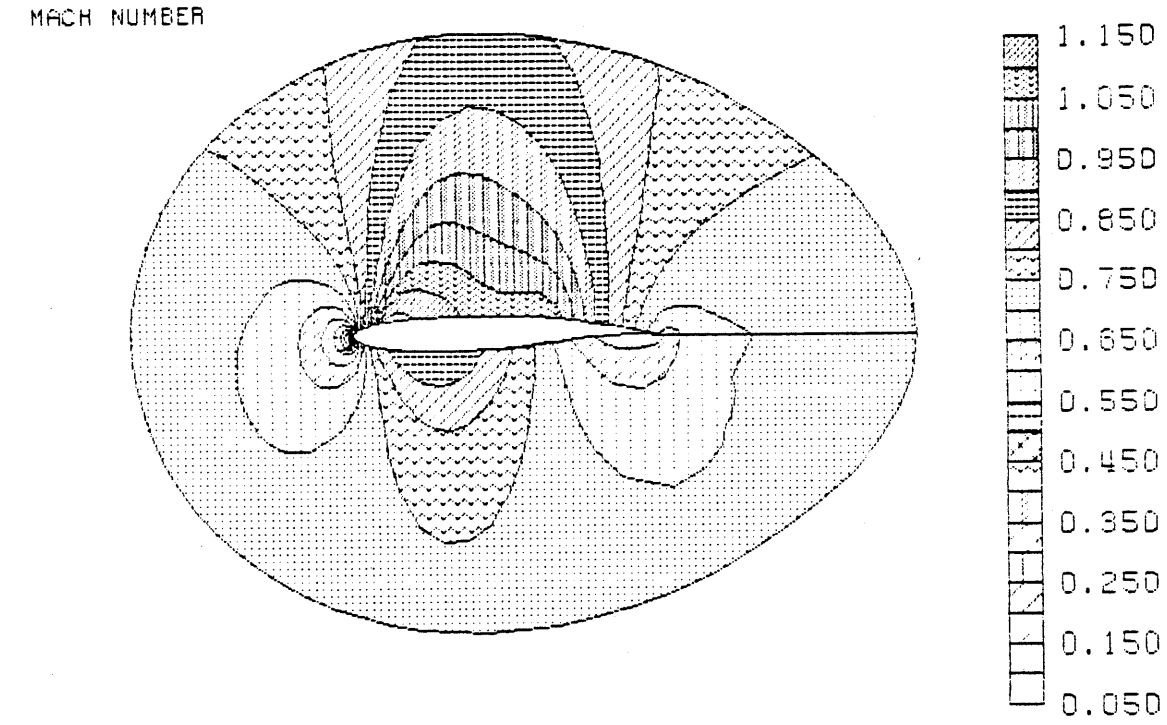

Figure 3-38c. Mach number contours.

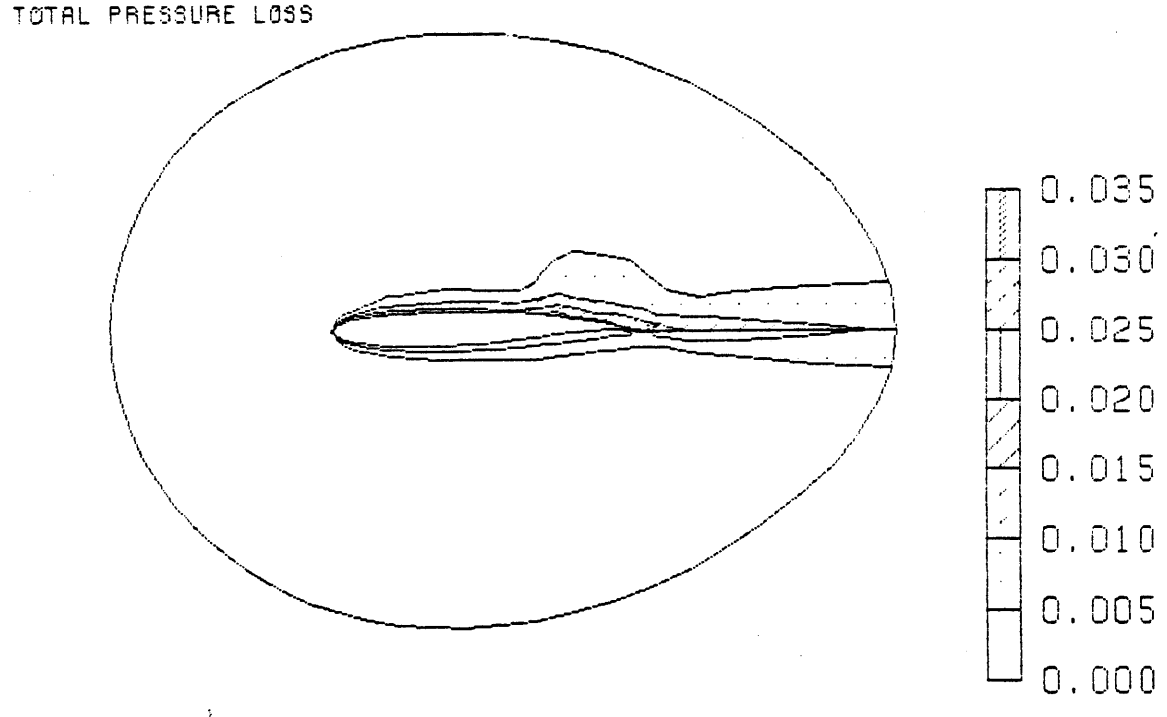

Figure 3-38d. Total pressure loss contours.

Figure 3-38. KORN airfoil for M = 0.75 and 0.12 degrees angle of attack. Multiple-grid solution on 65*17 O-type mesh with 3 global levels. [Run 208]

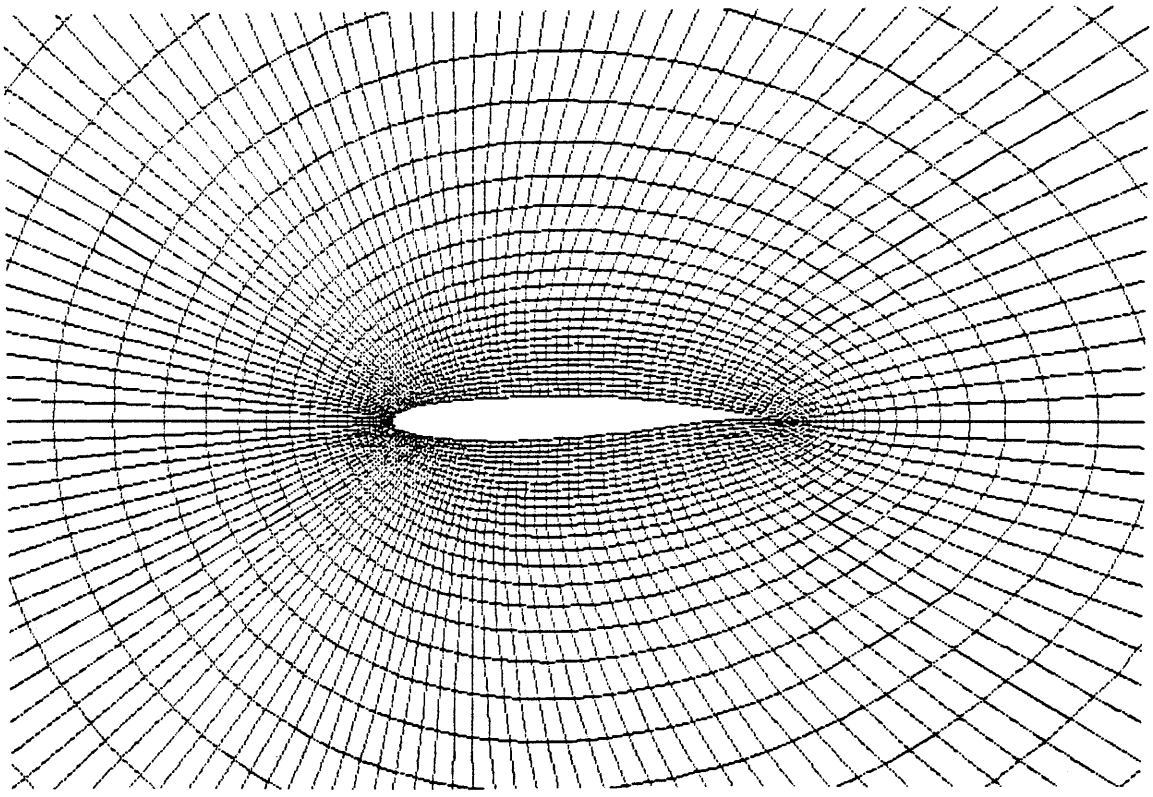

Figure 3-39. Near field of KORN airfoil for 129*33 global O-type mesh. [Run 210]

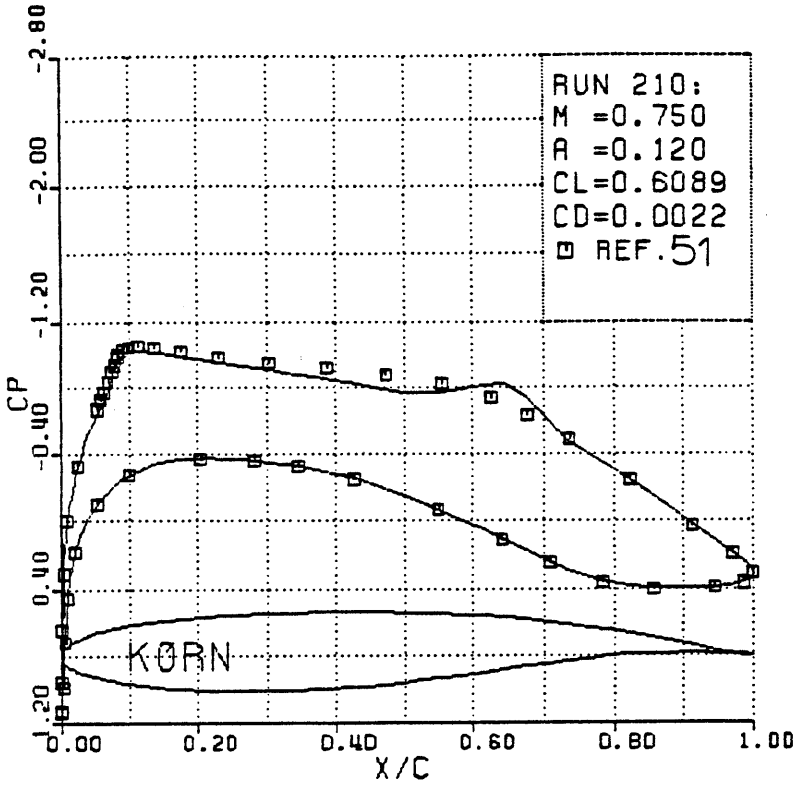

Figure 3-40a. Surface pressure coefficient.

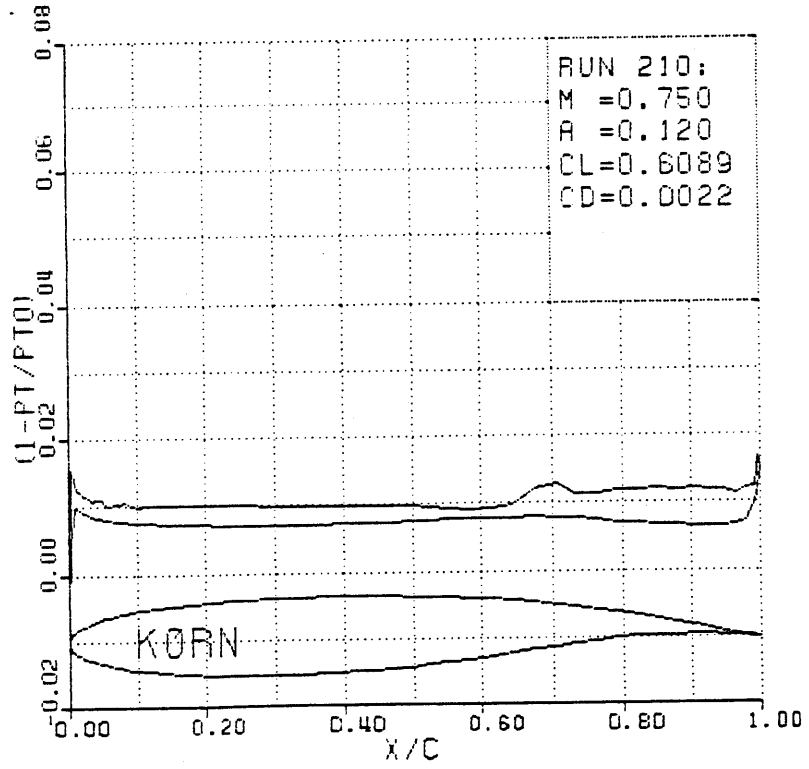

Figure 3-40b. Surface total pressure loss.

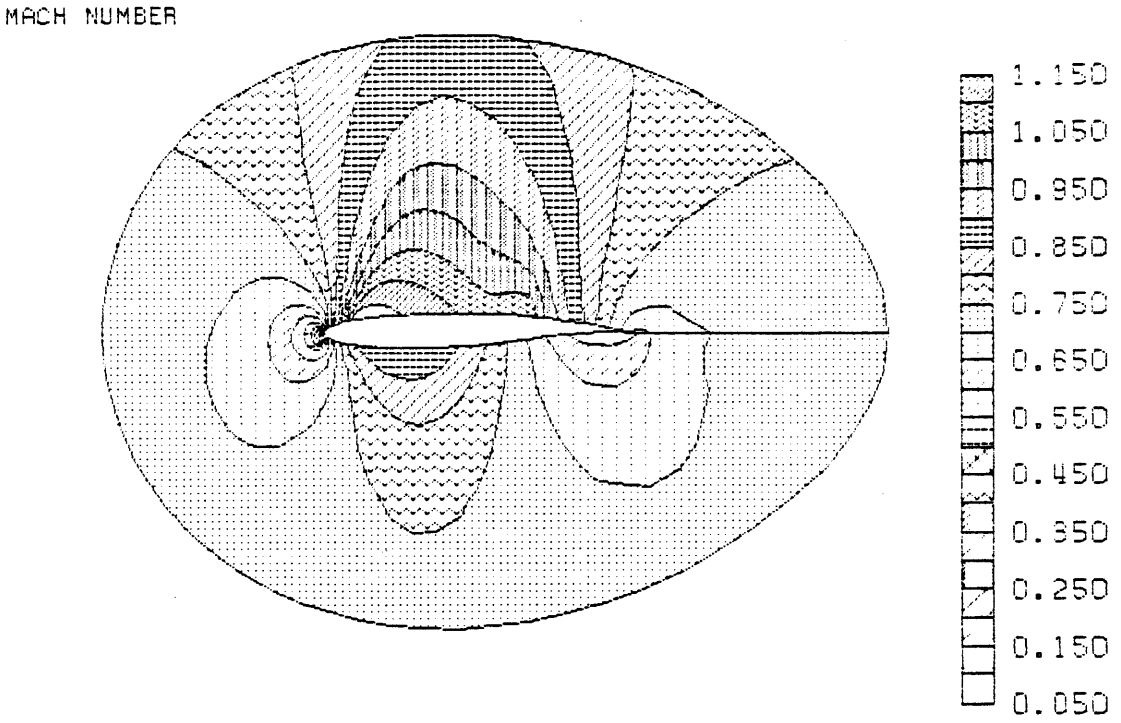

Figure 3-40c. Mach number contours.

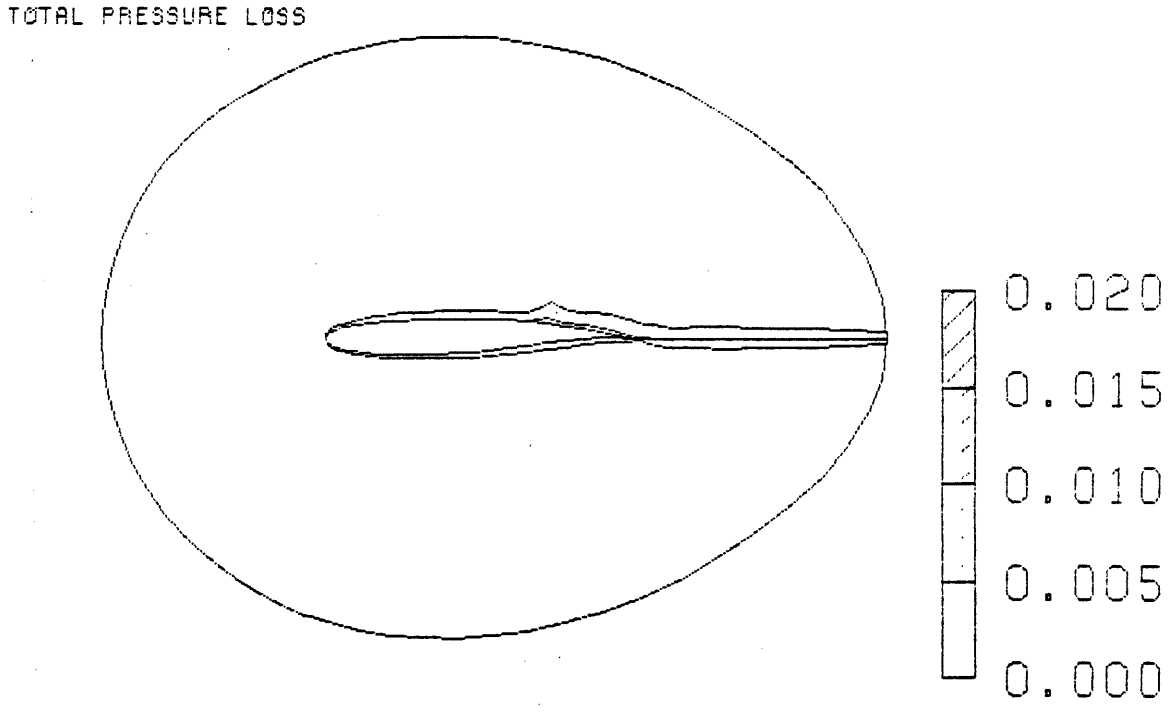

Figure 3-40d. Total pressure loss contours.

Figure 3-40. KORN airfoil for M = 0.75 and 0.12 degrees angle of attack. Multiple-grid solution on 129*33 O-type mesh with 4 global levels. [Run 210]

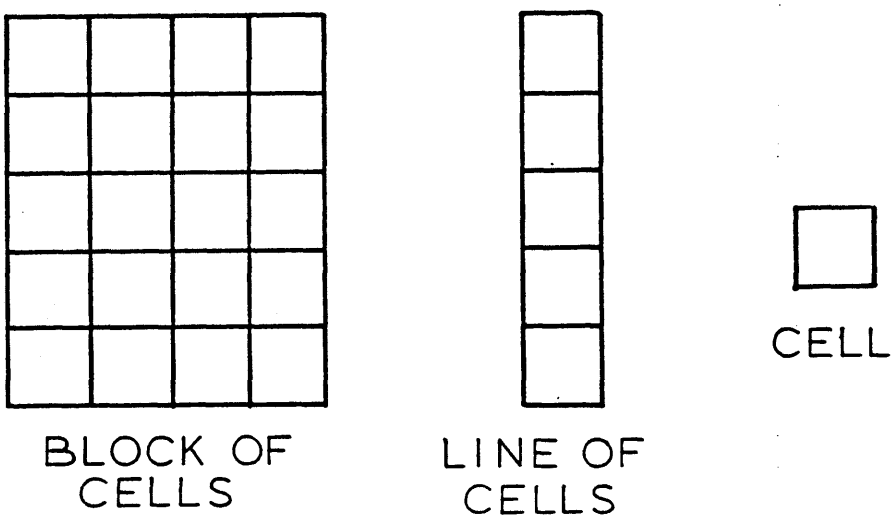

Figure 4-1. Three possible base structures.

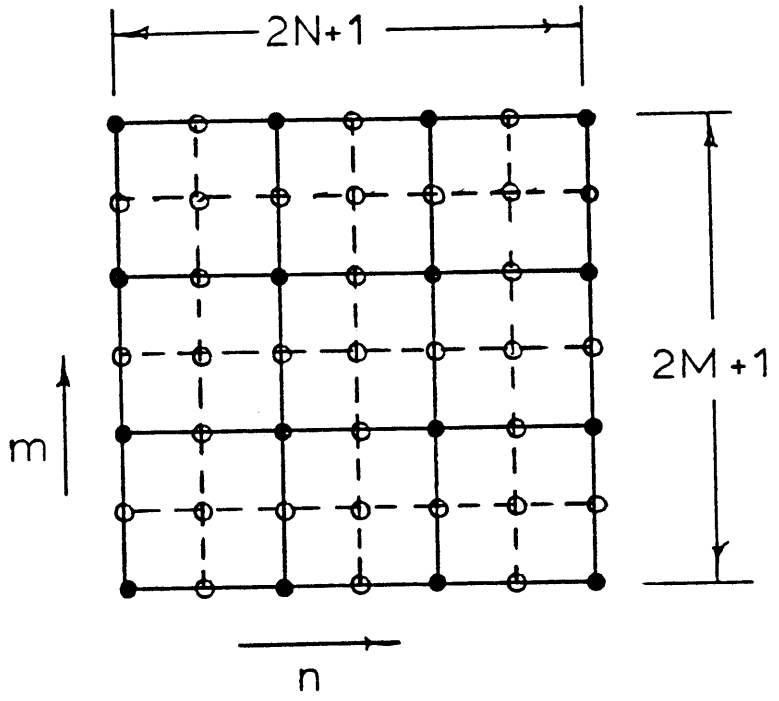

Figure 4-2. Block base structure.

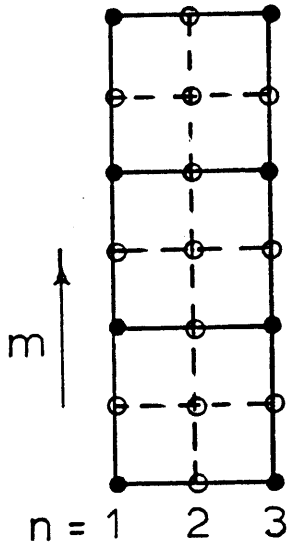

Figure 4-3. Line base structure.

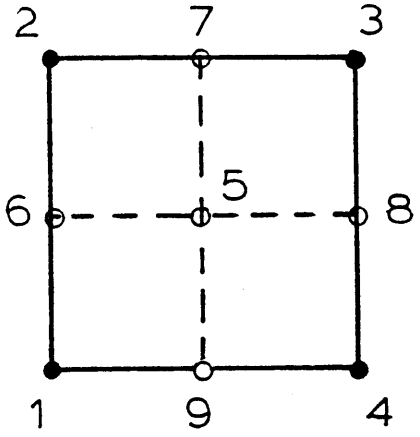

Figure 4-4. Cell base structure.

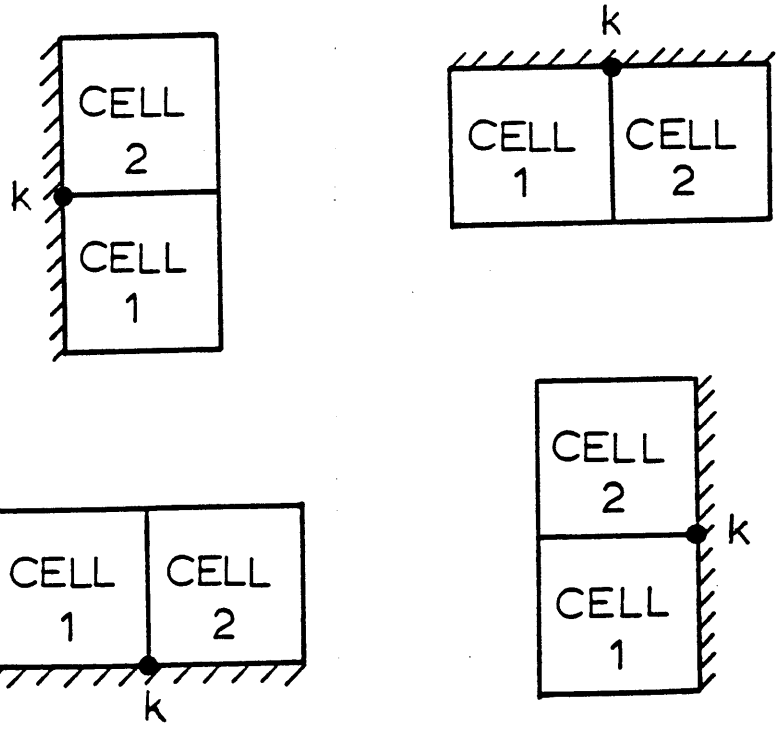

Figure 4-5. Four possible boundary cell orientations.

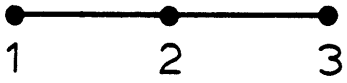

Figure 4-6. Embedded mesh interface pointer notation.

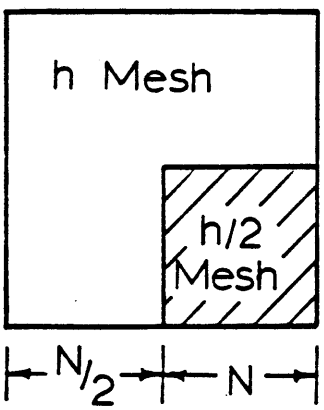

Figure 4-7a.

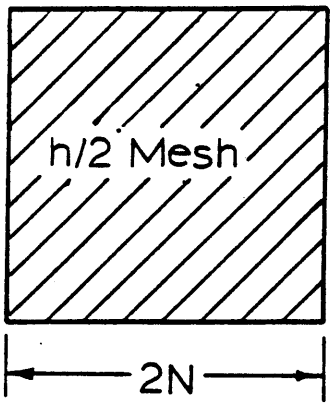

Figure 4-7b.

Figure 4-7 Embedded and global mesh refinement for 2-D example

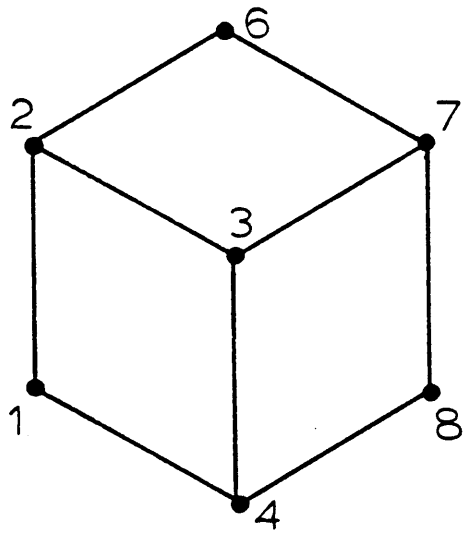

Figure 4-8a.

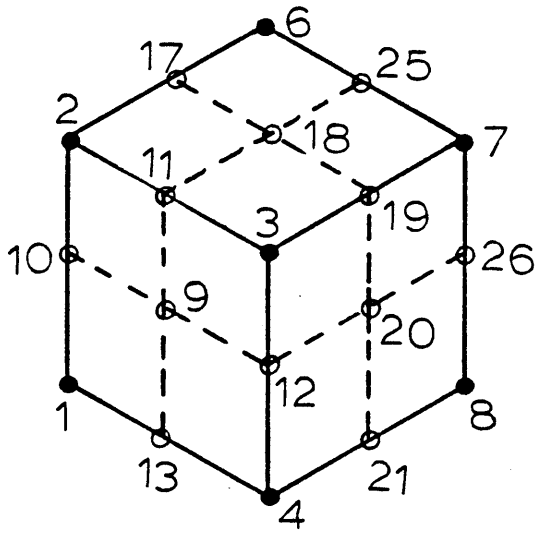

Figure 4-8b.

Figure 4-8. 3-D cell structure for the base and coarse mesh accelerator.

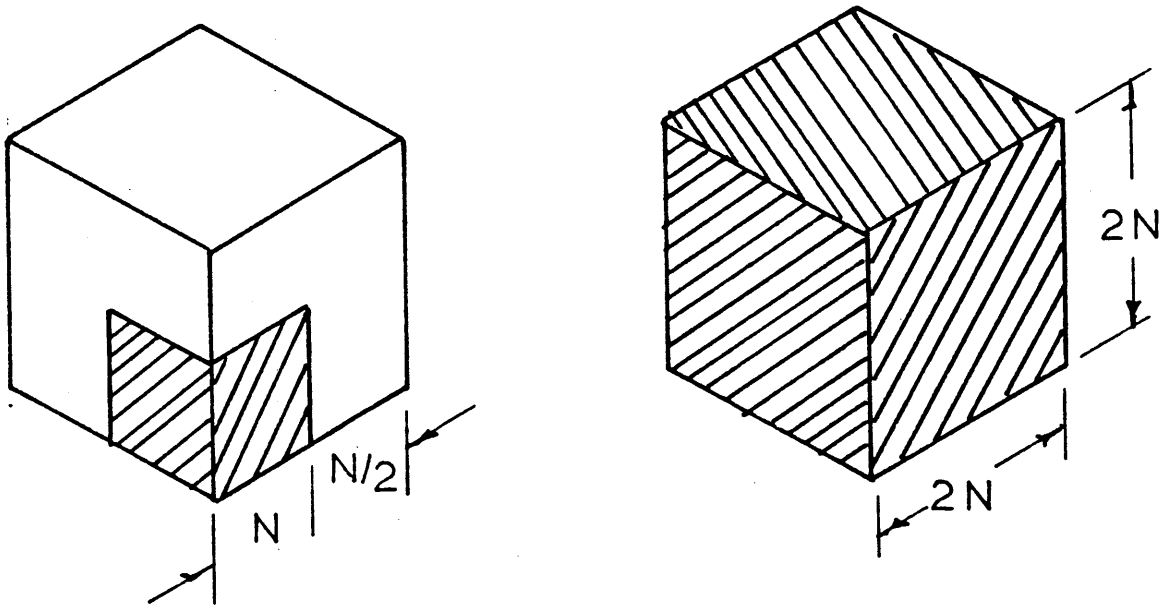

Figure 4-9. Example of embedded and global mesh refinement.

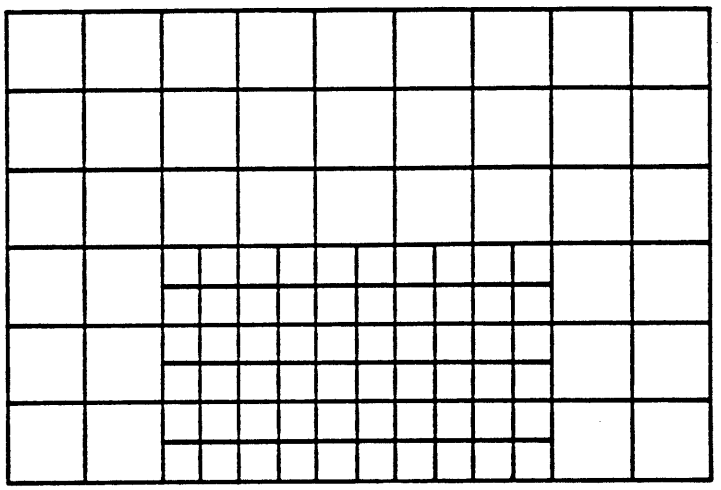

Figure 5-1. Embedded mesh topology.

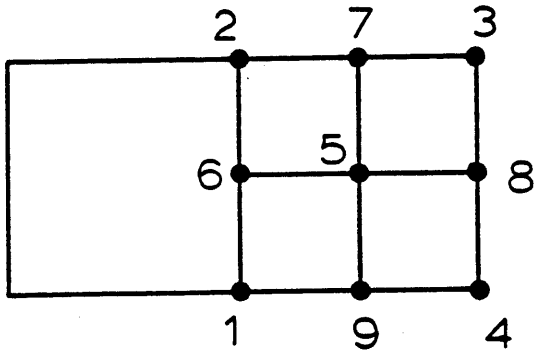

Figure 5-2. Embedded mesh interface notation.

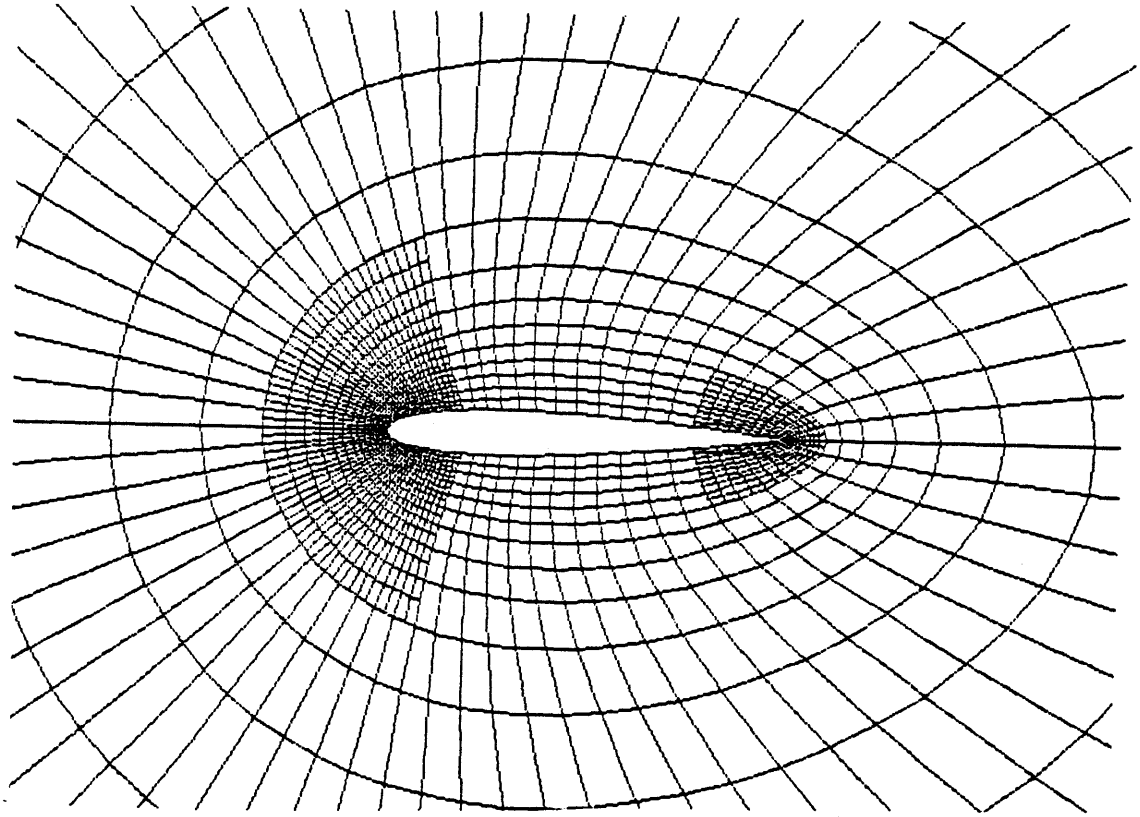

Figure 5-3. Near field of NACA0012 airfoil for embedded O-type mesh. [run 197]

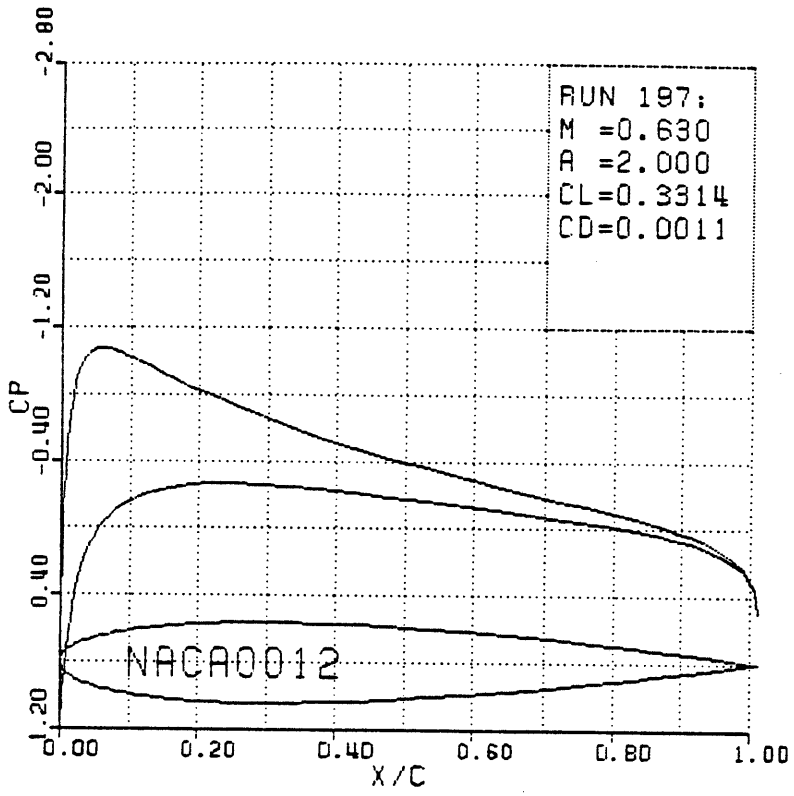

Figure 5-4a. Surface pressure coefficient.

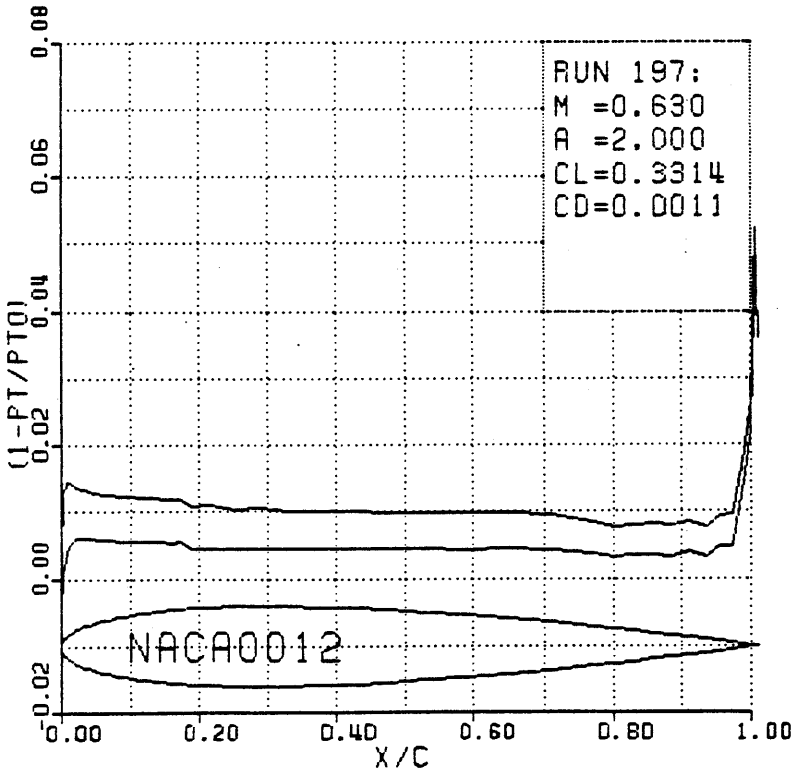

Figure 5-4b. Surface total pressure loss.

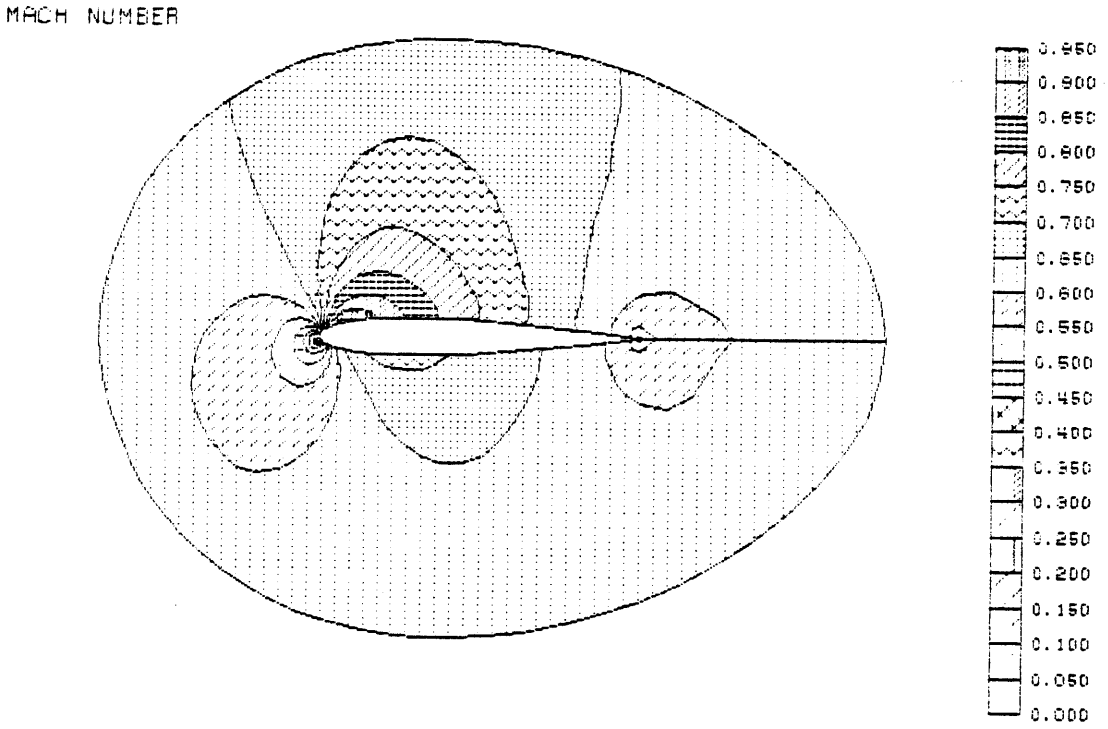

Figure 5-4c. Mach number contours.

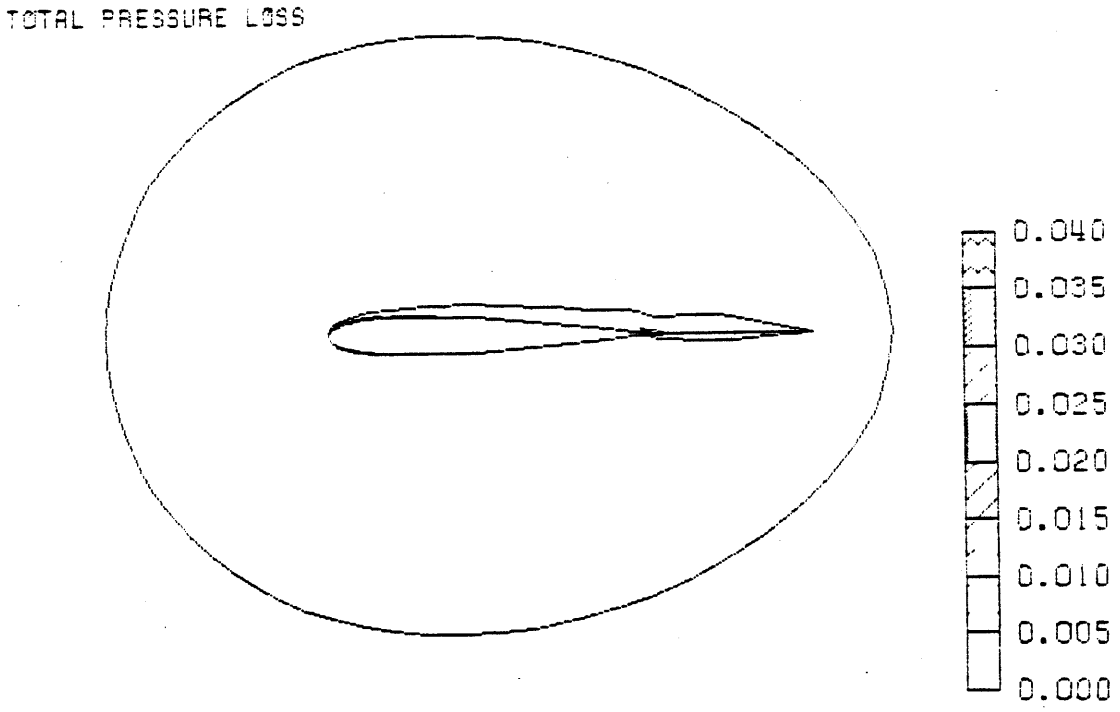

Figure 5-4d. Total pressure loss contours.

Figure 5-4. NACA0012 airfoil for M = 0.63 and angle of attack of 2.0 degrees. Embedded mesh solution on O-type mesh. [run 197]

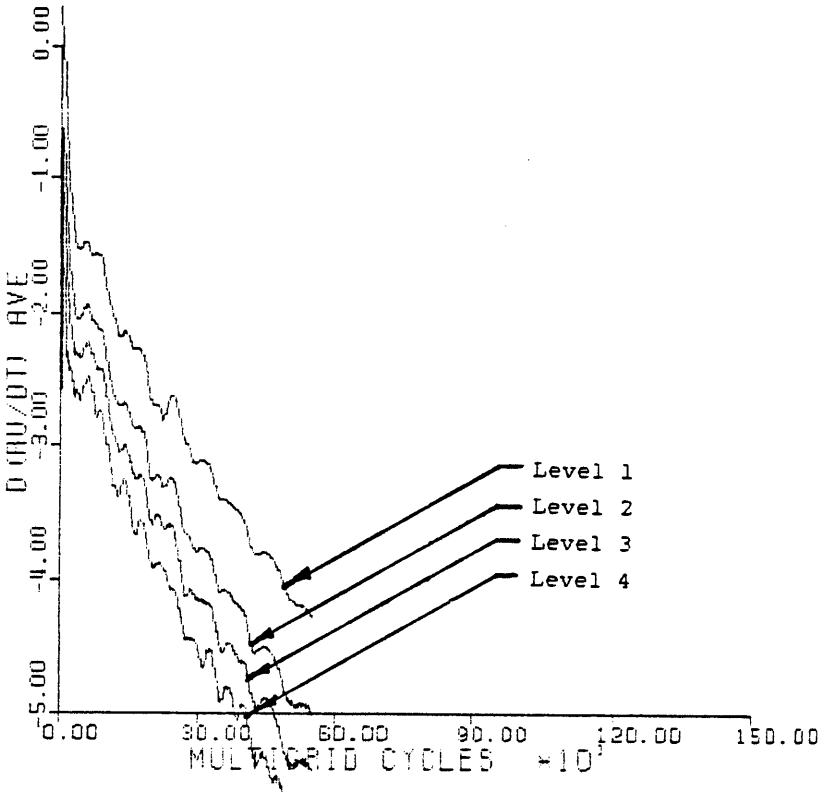

Figure 5-5. Comparison of residuals on each of the four multiple-grid levels for solution of figure 5-4. [run 197]

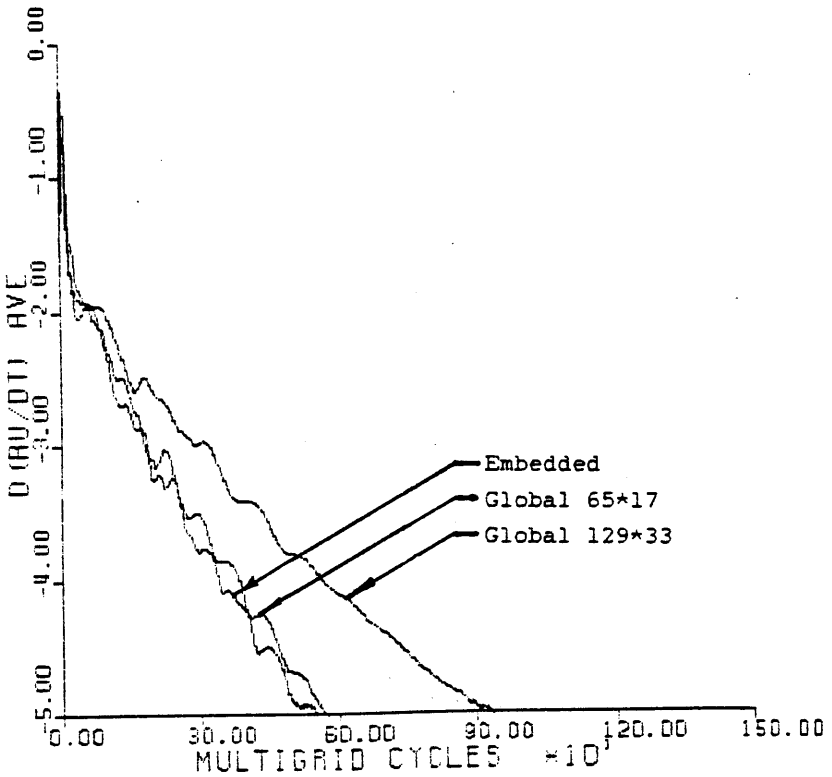

Figure 5-6. Comparison of convergence histories for embedded solution, 65*17 global solution, and 129*33 global solution (figures 5-4,3-10,3-13). [runs 197,182,181]

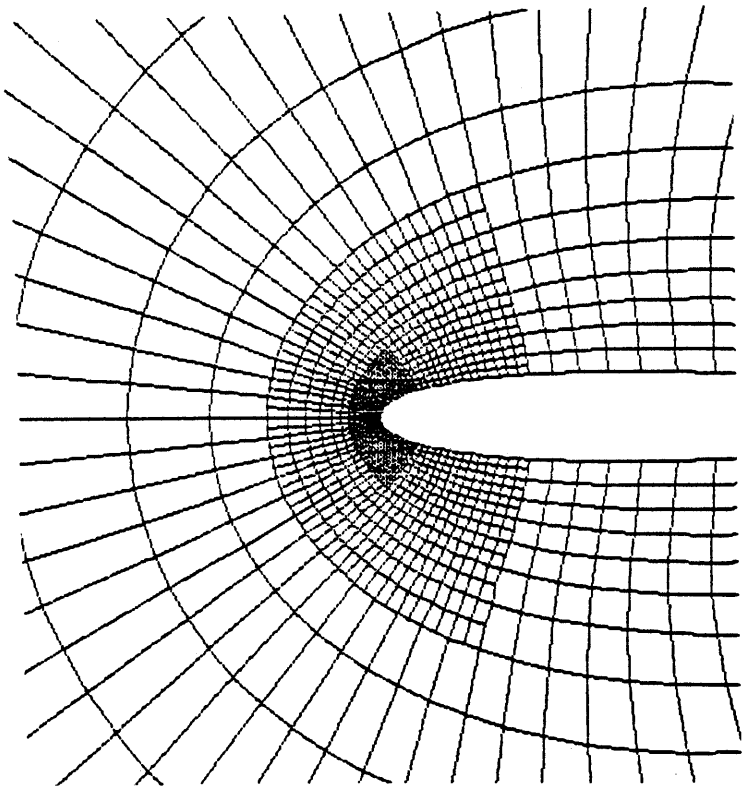

Figure 5-7. Double embedded mesh in leading edge region. [run 199]

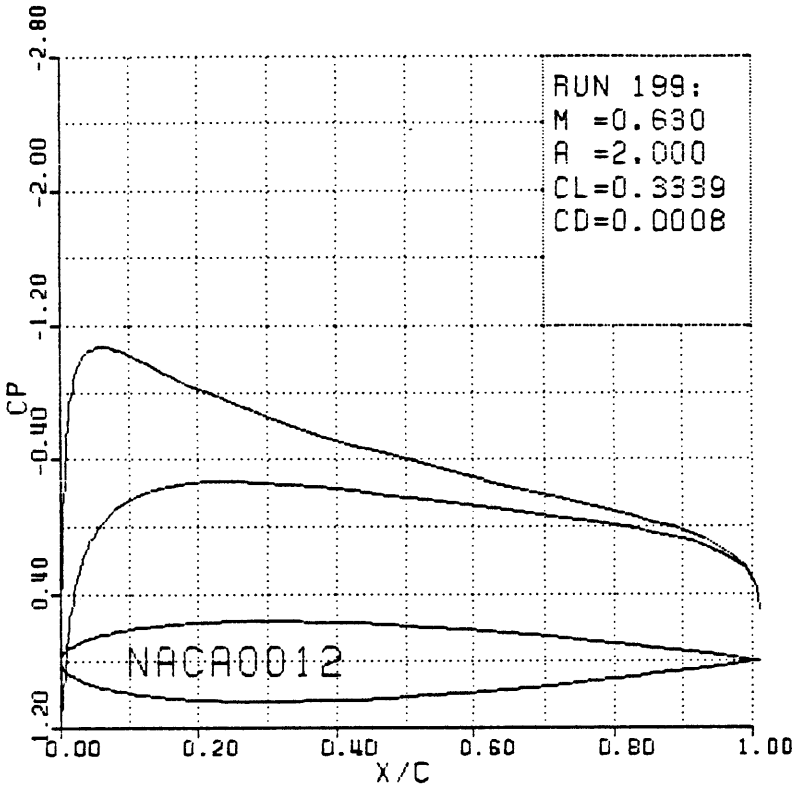

Figure 5-8a. Surface pressure coefficient.

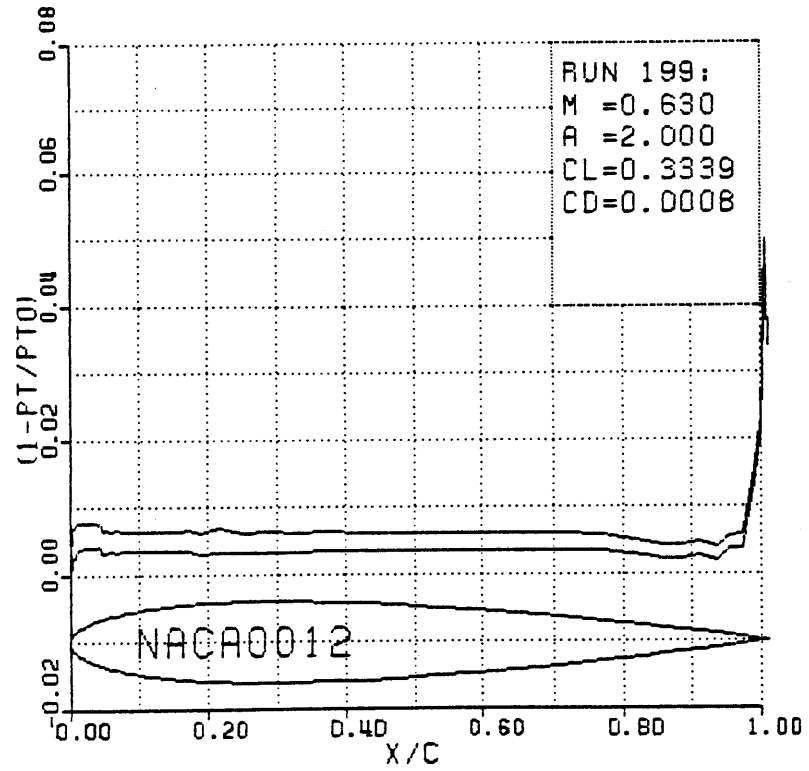

Figure 5-8b. Surface total pressure loss.

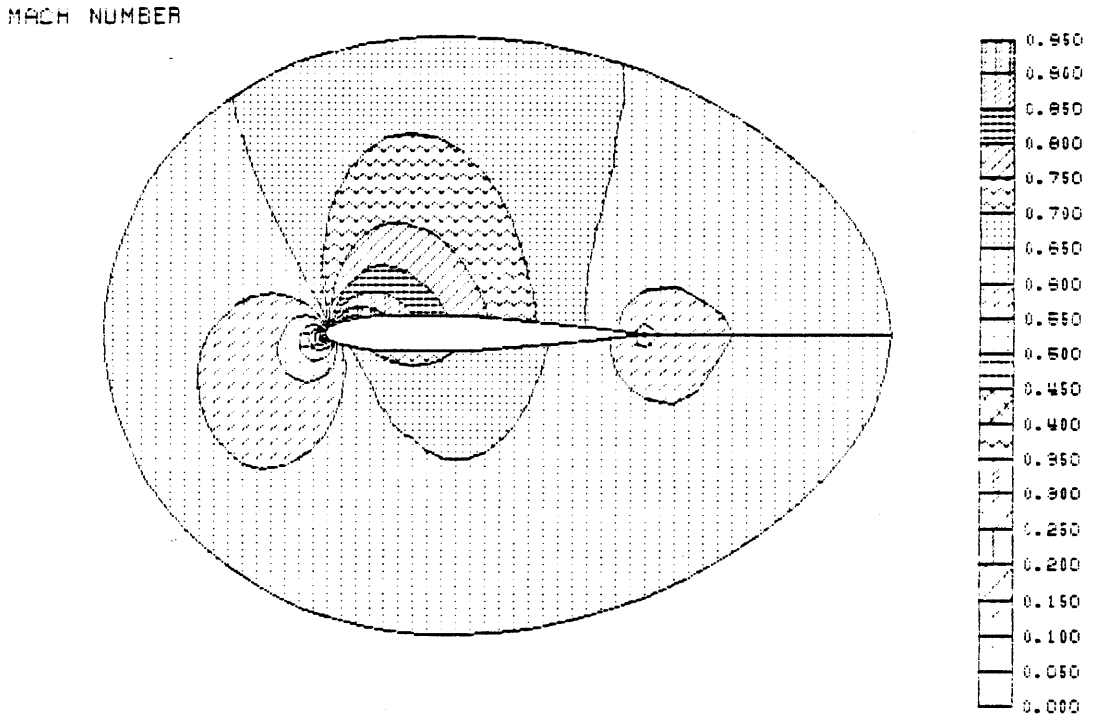

Figure 5-8c. Mach number contours.

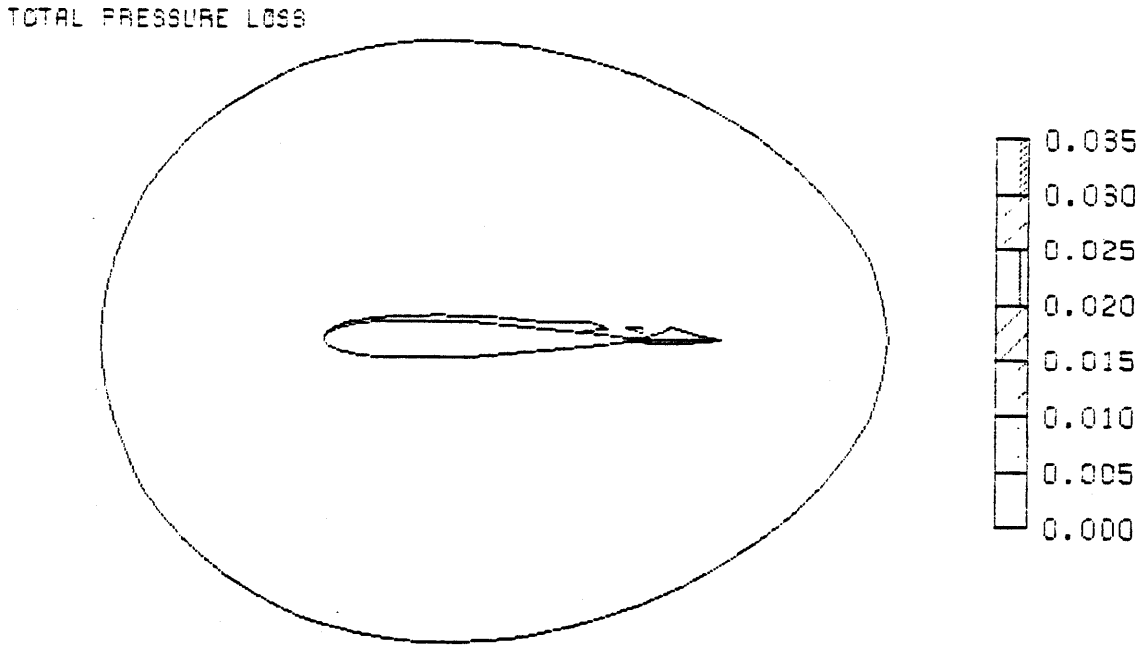

Figure 5-8d. Total pressure loss contours.

Figure 5-8. NACA0012 airfoil for M = 0.63 and angle of attack of 2.0 degrees. Double embedded mesh solution on O-type mesh. [run 199]

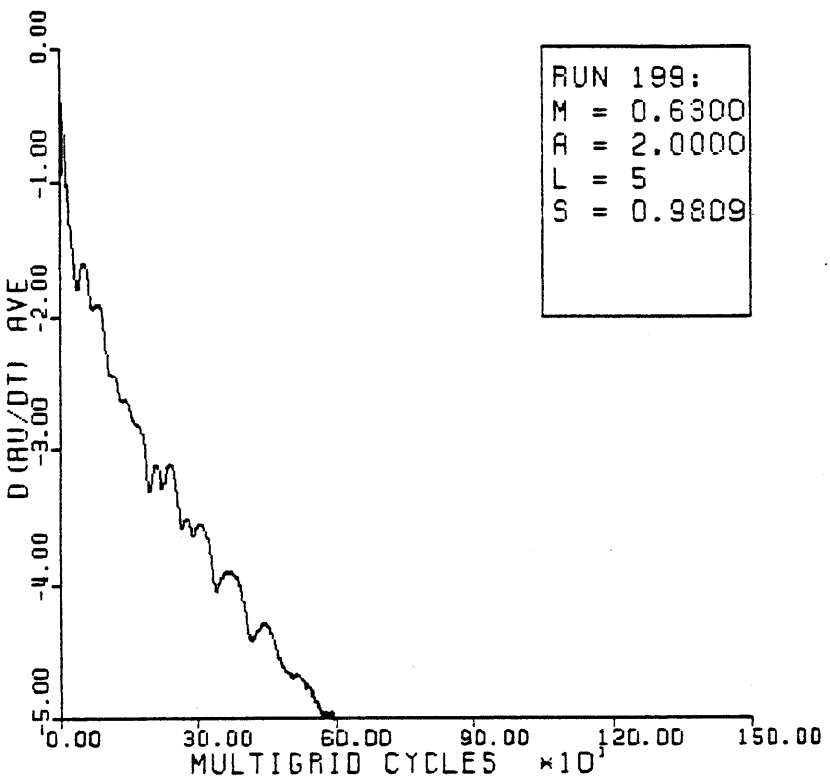

Figure 5-9. Convergence history for double embedded mesh solution. [run 199]

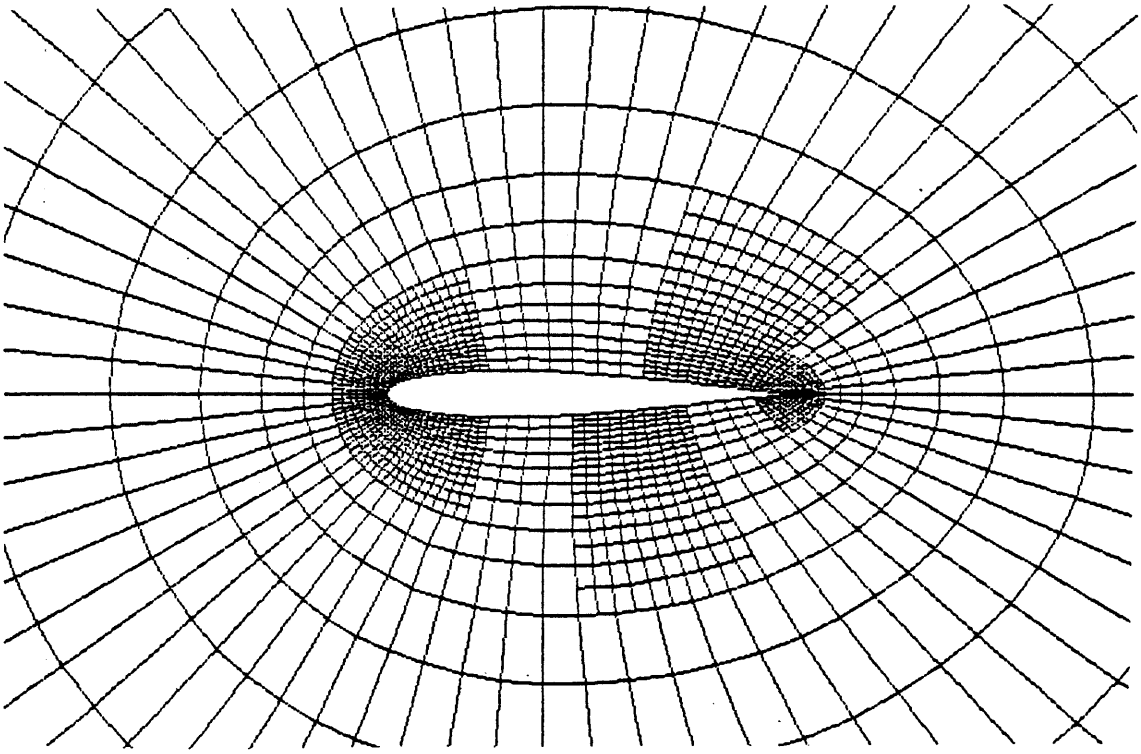

Figure 5-10. Near field of NACA0012 airfoil for embedded O-type mesh. [run 198]

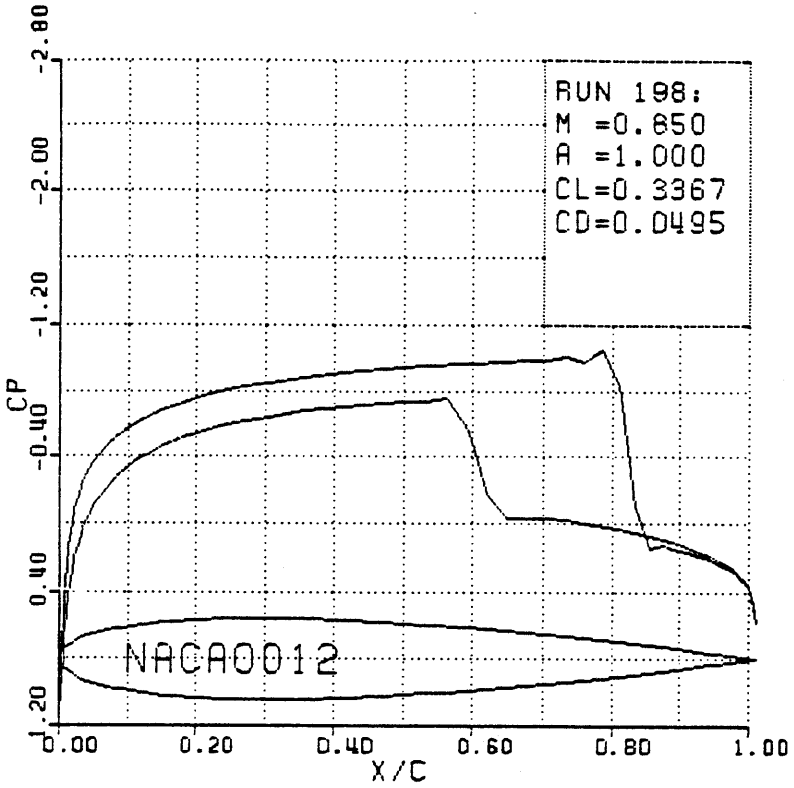

Figure 5-11a. Surface pressure coefficient.

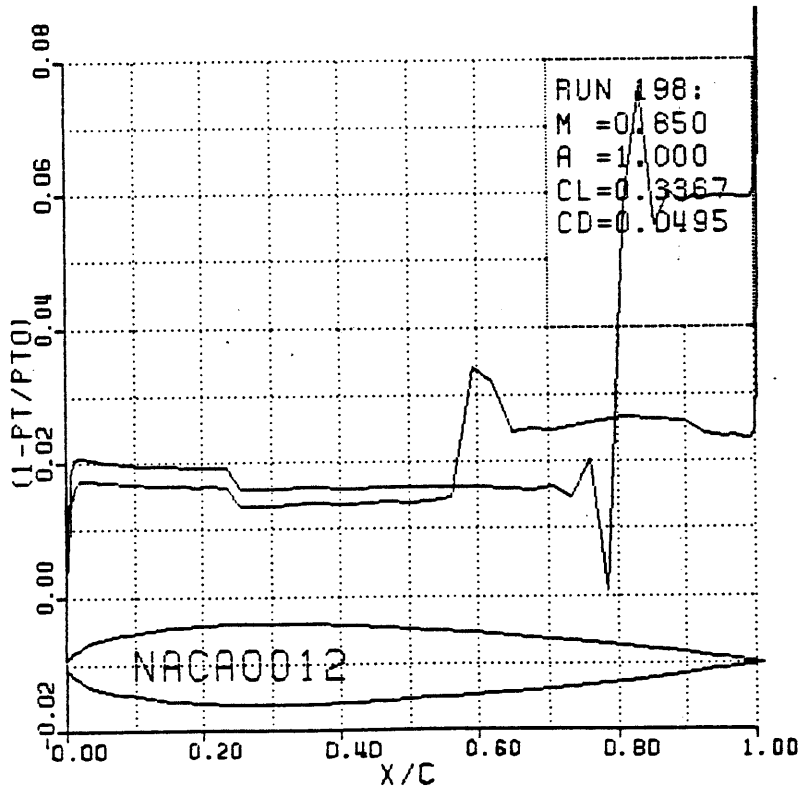

Figure 5-11b. Surface total pressure loss.

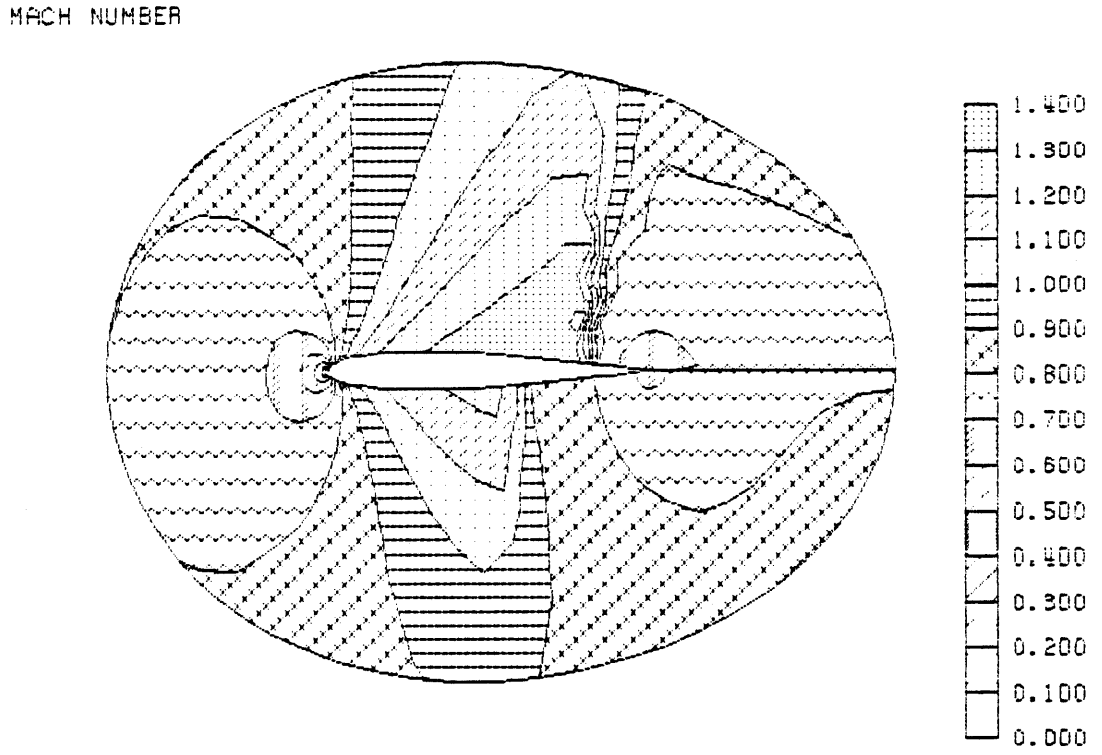

Figure 5-11c. Mach number contours.

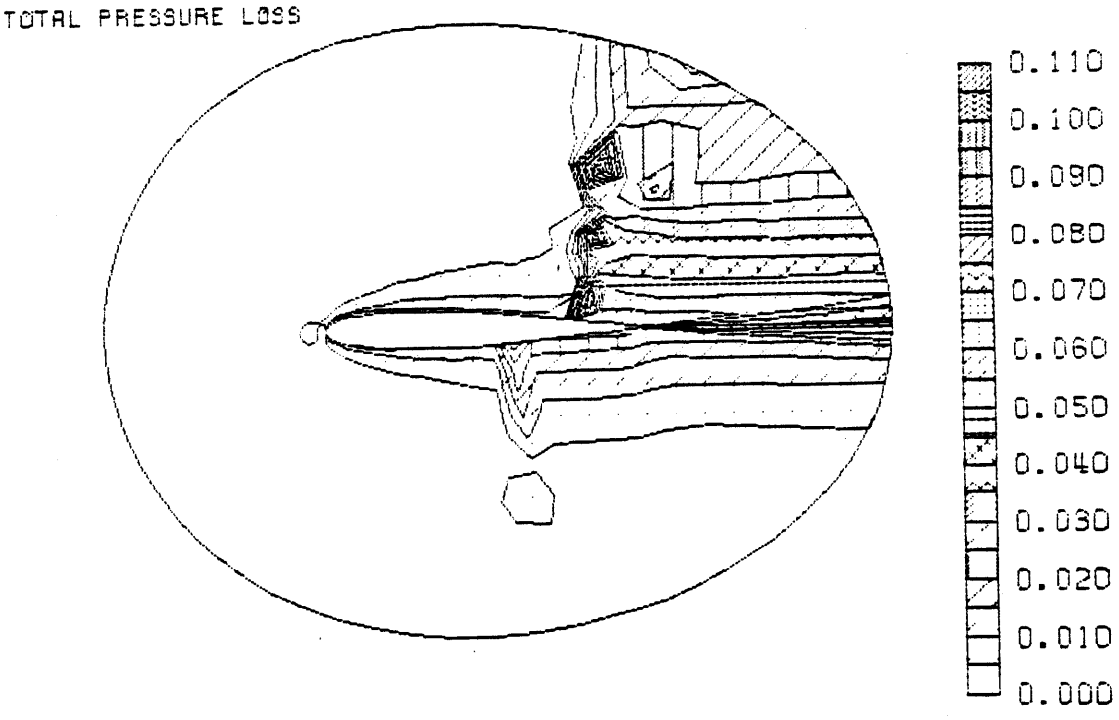

Figure 5-11d. Total pressure loss contours.

Figure 5-11. NACA0012 airfoil for M = 0.85 and angle of attack of 1.0 degrees. Embedded mesh solution on O-type mesh. [run 198]

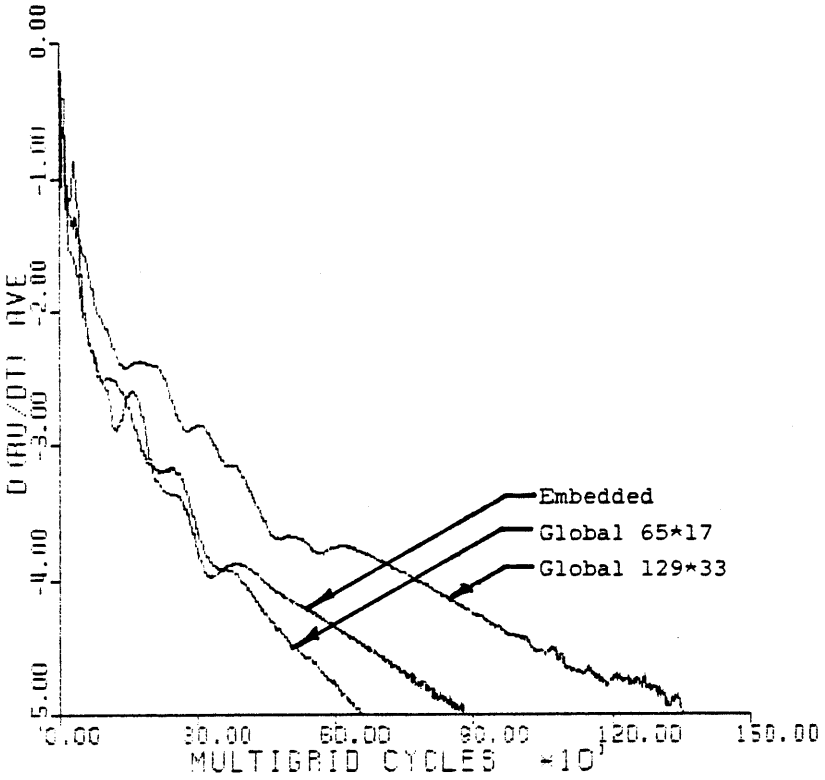

Figure 5-12. Comparison of convergence histories for embedded solution, 65*17 global solution, and 129*33 global solution (figures 5-11,3-27,3-30). [runs 214,215,216]

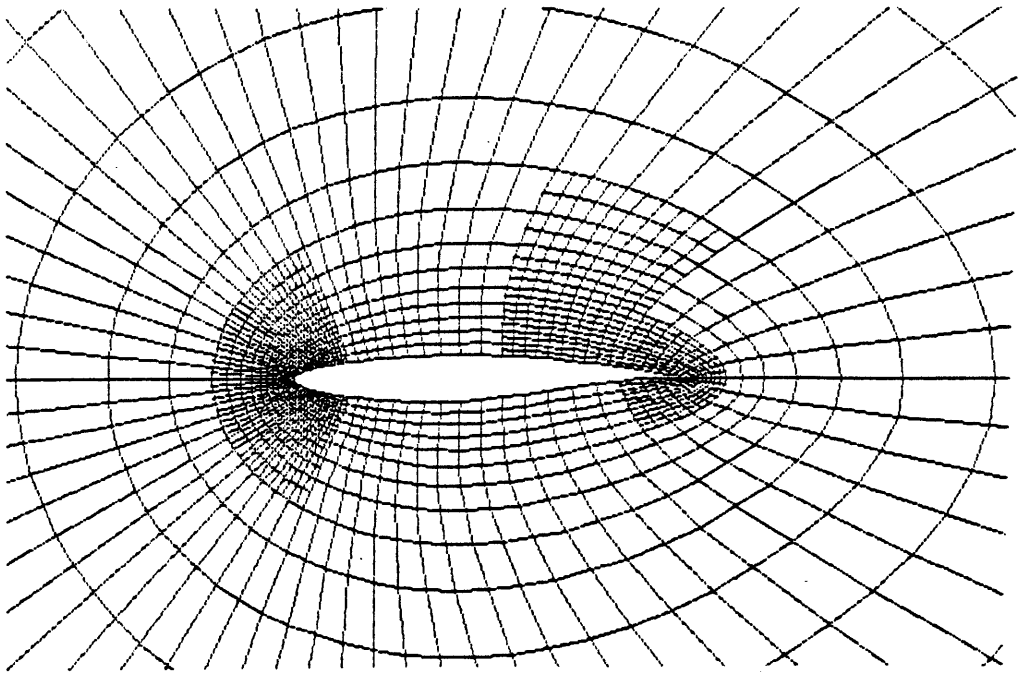

Figure 5-13. Near field of RAE2822 airfoil for embedded O-type mesh. [Run 212]

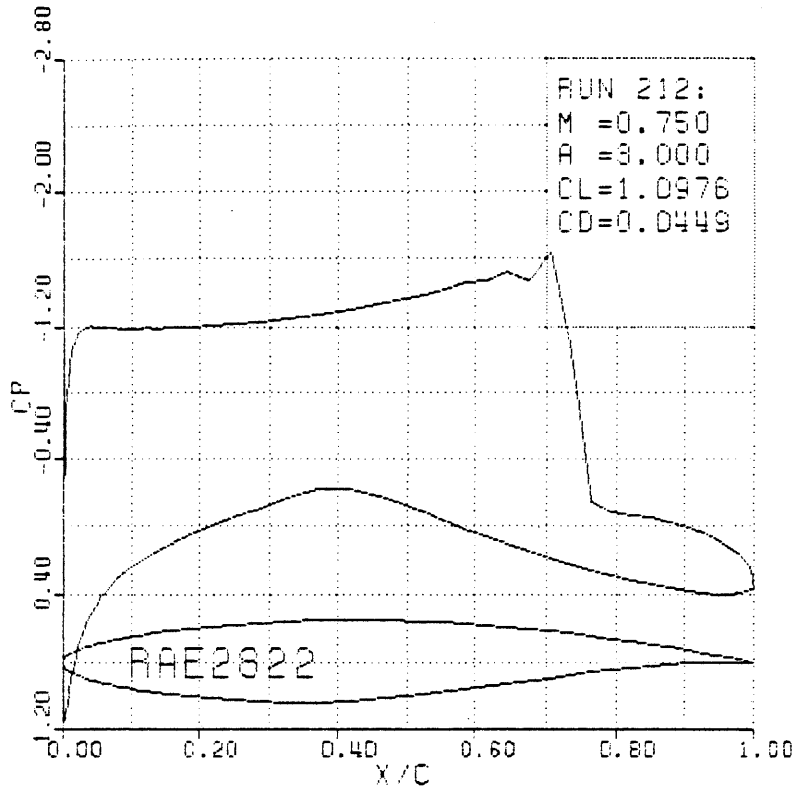

Figure 5-14a. Surface pressure coefficient.

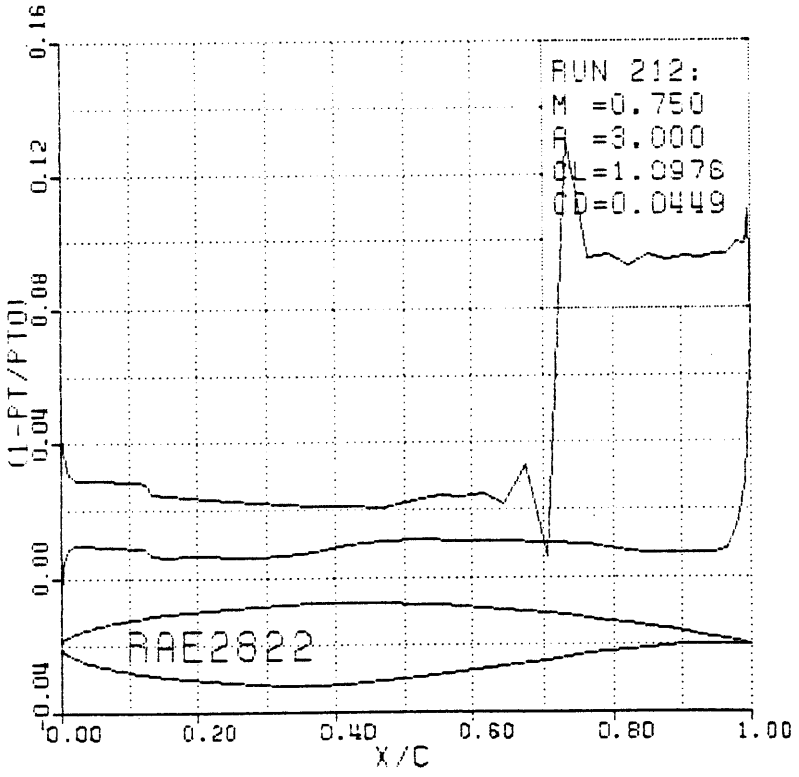

Figure 5-14b. Surface total pressure loss.

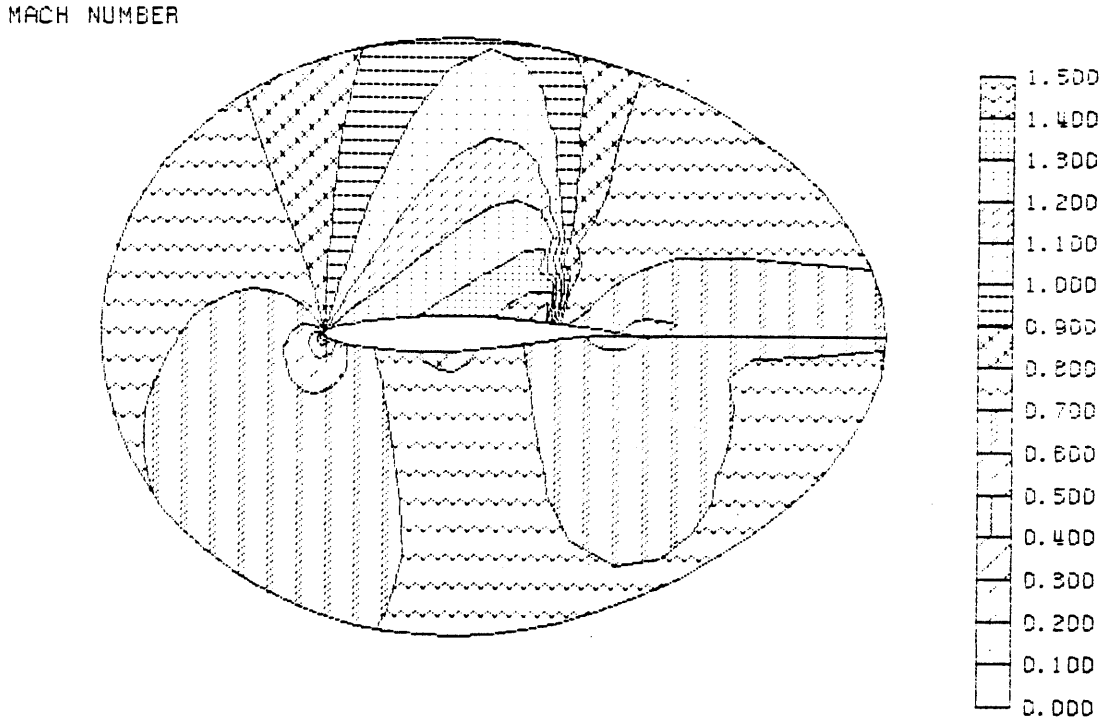

Figure 5-14c. Mach number contours.

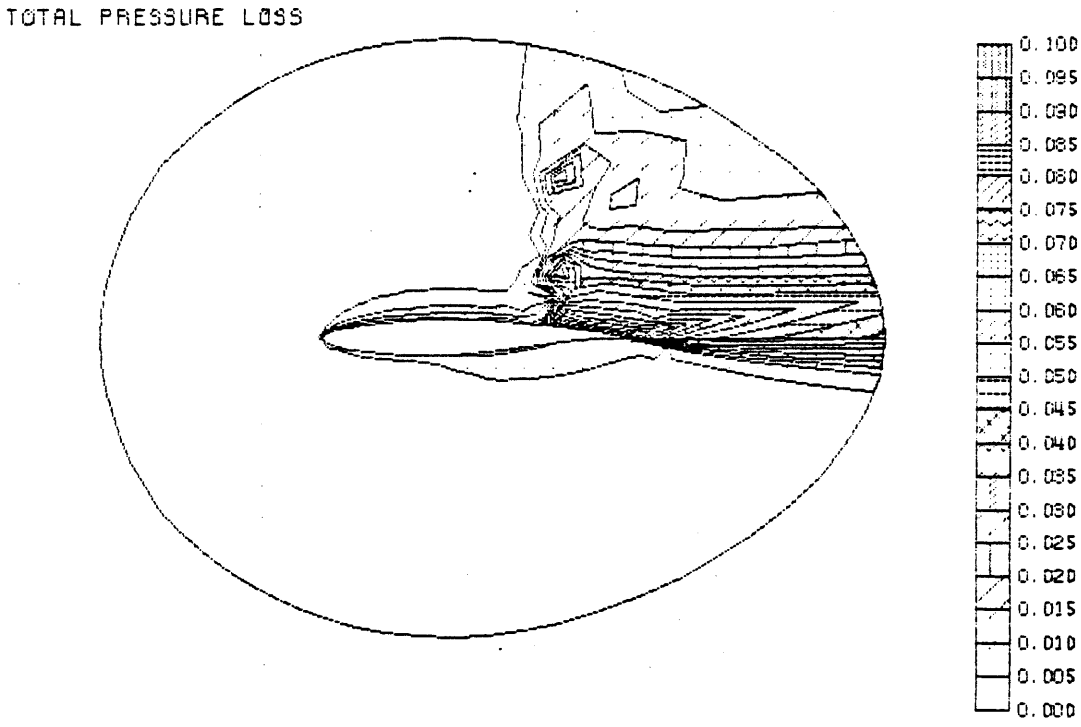

Figure 5-14d. Total pressure loss contours.

Figure 5-14. RAE2822 airfoil for M = 0.75 and angle of attack of 3.0 degrees. Embedded mesh solution on O-type mesh. [Run 212]

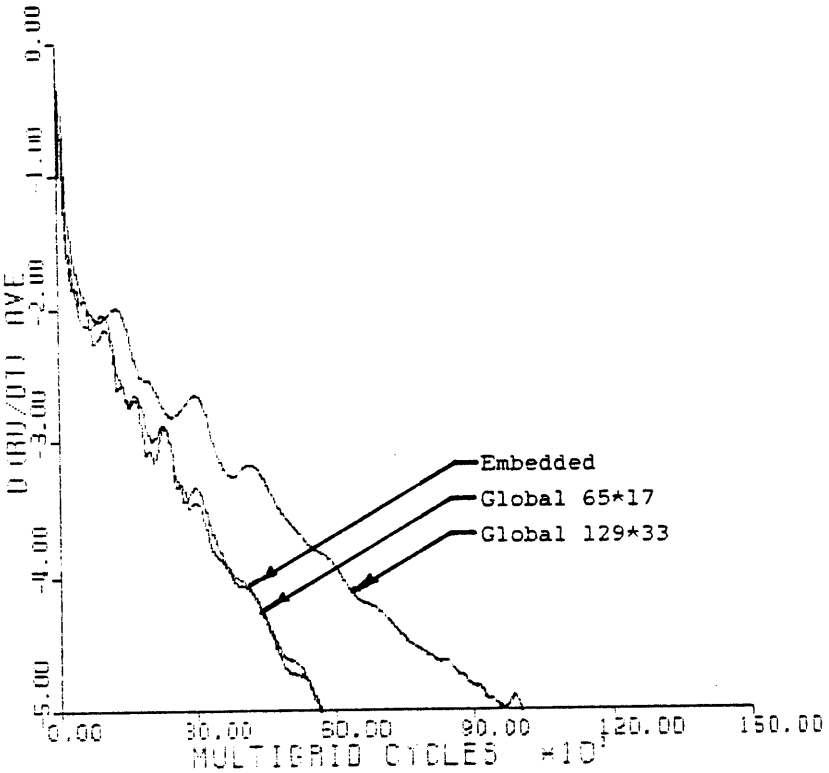

Figure 5-15. Comparison of convergence histories for embedded solution, 65*17 global solution, and 129*33 global solution (figures 5-14,3-33,3-35). [Runs 211, 212, 213]

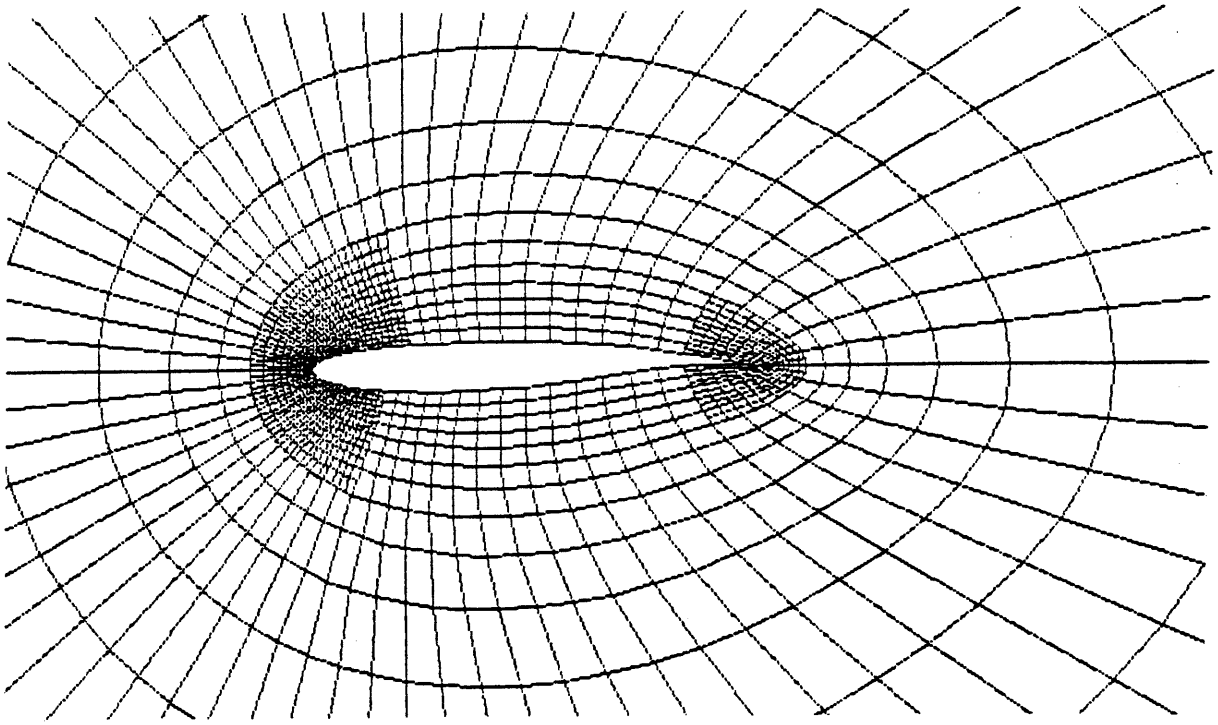

Figure 5-16. Near field of KORN airfoil for embedded O-type mesh. [Run 209]

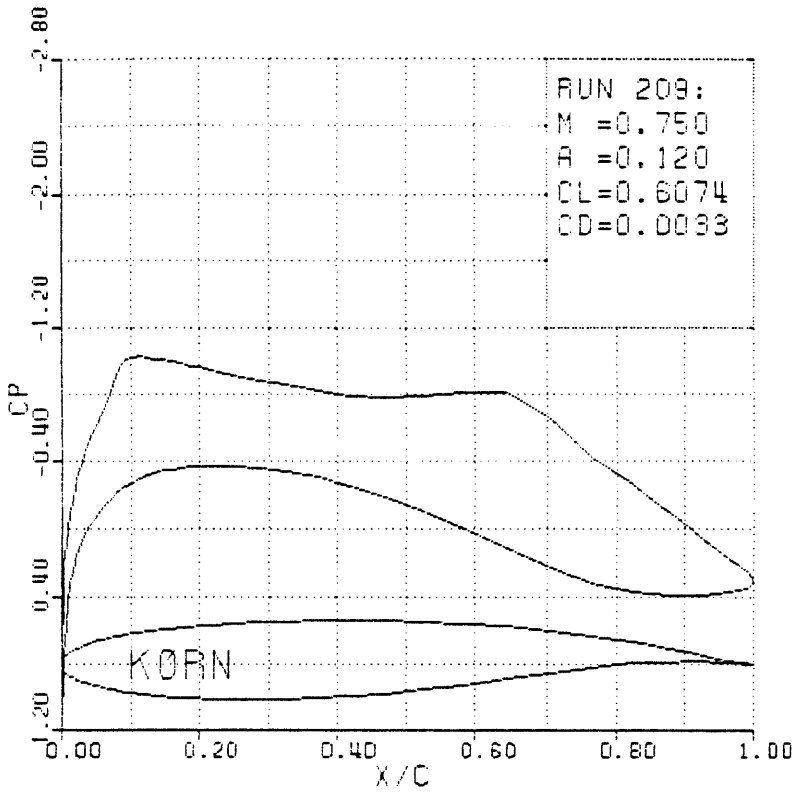

Figure 5-17a. Surface pressure coefficient.

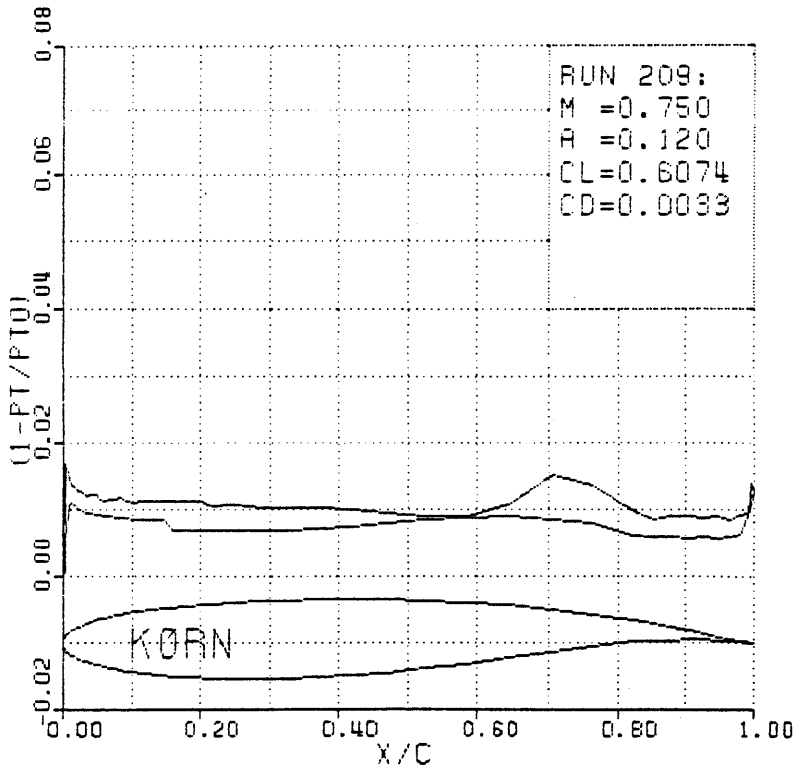

Figure 5-17b. Surface total pressure loss.

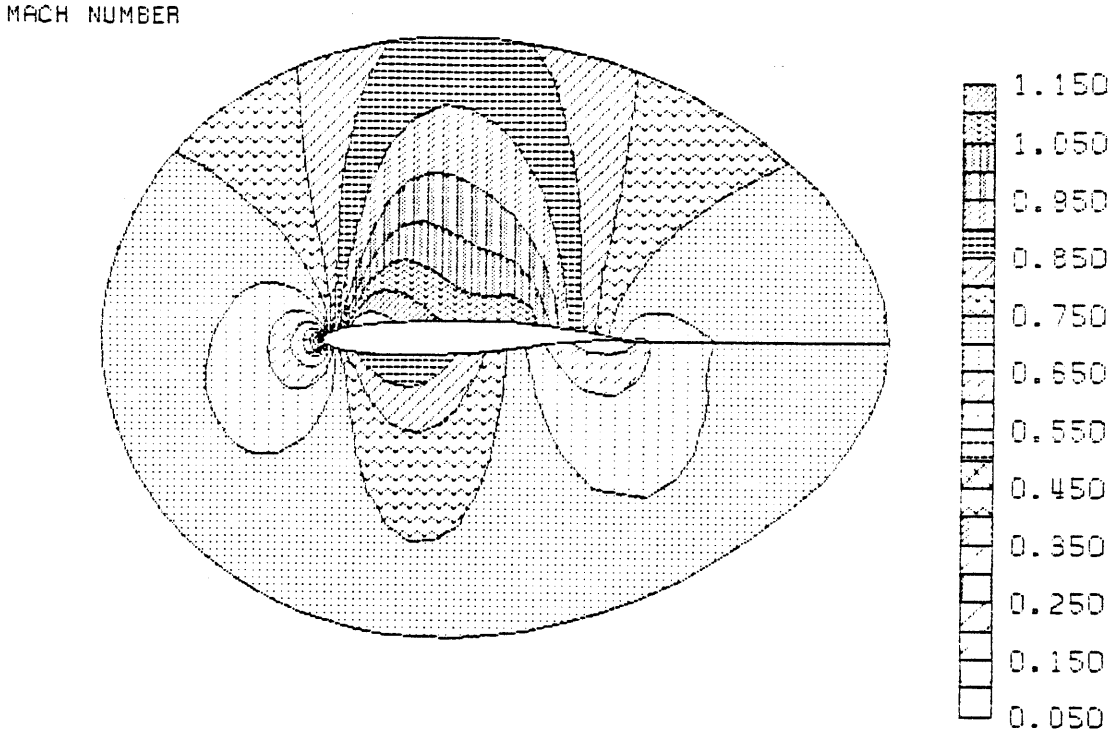

Figure 5-17c. Mach number contours.

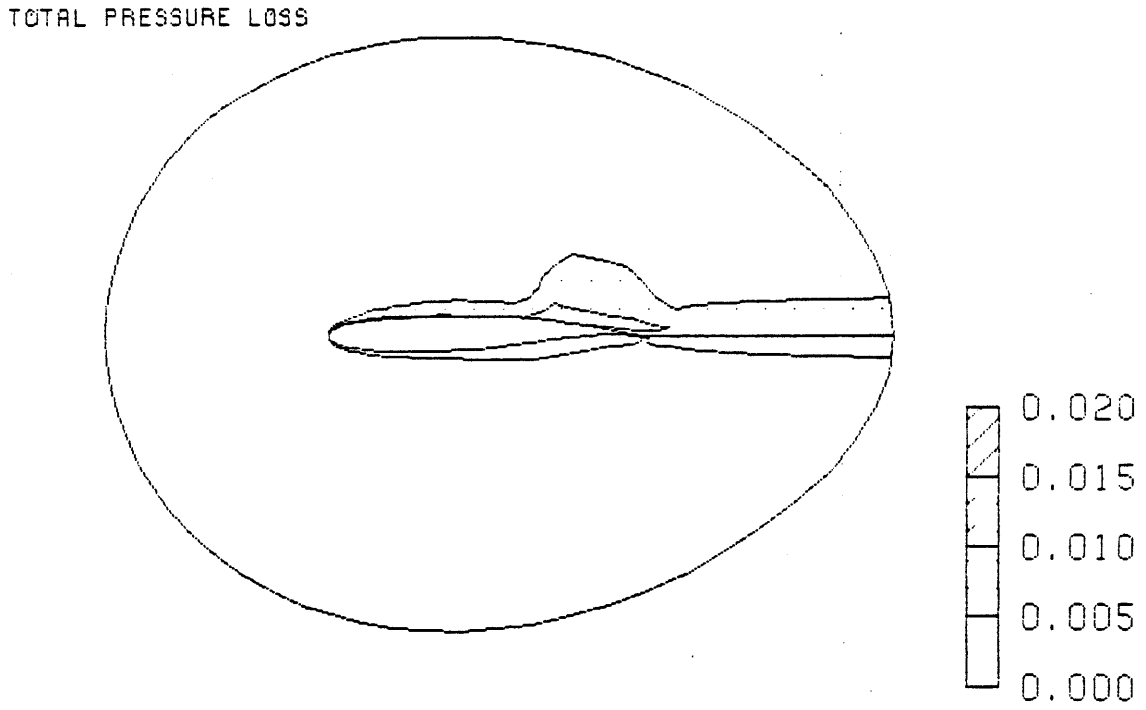

Figure 5-17d. Total pressure loss contours.

Figure 5-17. KORN airfoil for M = 0.75 and angle of attack of 0.12 degrees. Embedded mesh solution on O-type mesh. [Run 209]

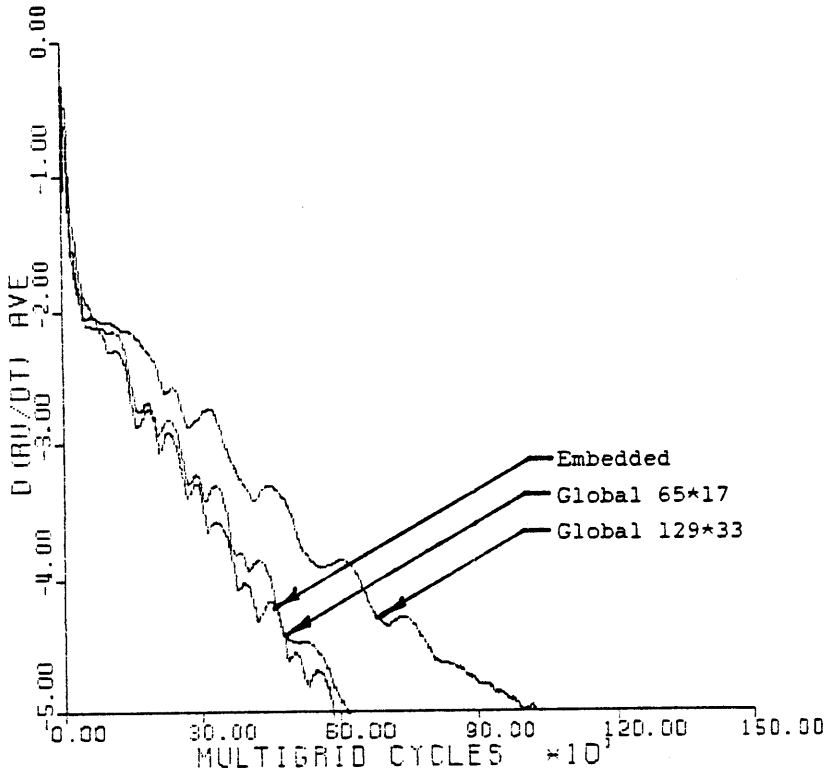

Figure 5-18. Comparison of convergence histories for embedded solution, 65*17 global solution, and 129*33 global solution (figures 5-17,3-38,3-40). [Runs 208, 209, 210]

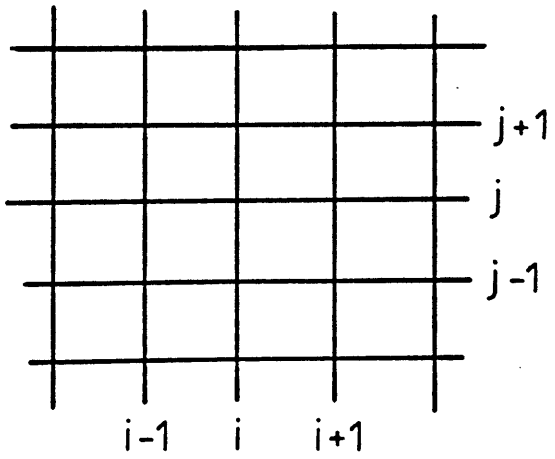

Figure B-1. Stability analysis mesh notation.

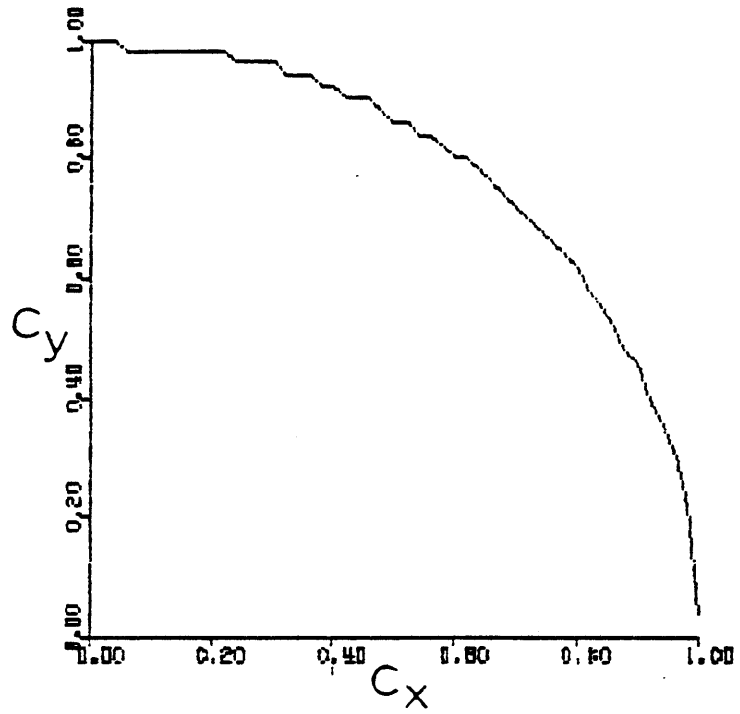

Figure B-2. Stability boundary |G|=1 for 2-D wave equation using the Ni scheme.

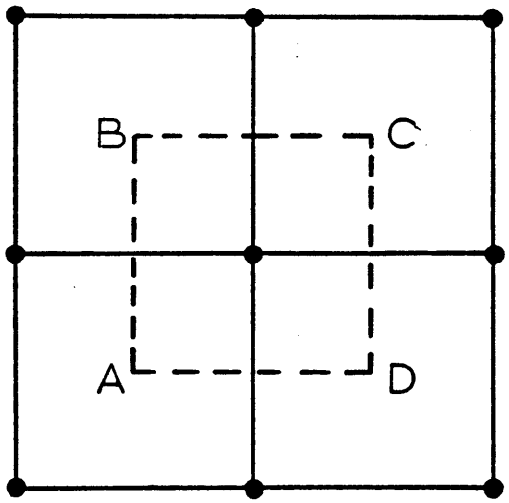

Figure D-1. Control volume for integration of dissipation terms using Ni multiple-grid method.

APPENDIX E

2-D AIRFOIL EULER CODE FOR O-TYPE MESHES

This appendix contains listings of the two computer codes used to generate the 2-D transonic airfoil solutions on O-type meshes presented in this thesis. The first program called GEOCREAT reads in a global O-type mesh as input and then interactively generates a file which contains the pointer system and mesh coordinates for the complete embedded mesh structure. The second program, EULERCELL, is the embedded multiple-grid Euler solver. EULERCELL requires two files as input, the pointer file defining the embedded mesh structure and a second file containing the flow conditions and control parameters (a sample of this file is given at the end of this appendix).

```
12345678901234567890123456789012345678901234567890123456789012345
C-----------------------------------------------------------------
C*****************************************************************
C-----------------------------------------------------------------
C
C-----------------------------------------------------------------
C          PROGRAM: GEOCREAT
C-----------------------------------------------------------------
C
C         THIS PROGRAM GENERATES THE GRID STRUCTURE AND POINTERS
C         FOR A O-MESH GRID TOPOLOGY.
C
      CALL GEOIN3
C
      CALL GEOPONT4
C
      CALL GEOOUT3
C
      STOP
      END
C-----------------------------------------------------------------
C          INCLUDE FILE: GEOCOM.COM
C-----------------------------------------------------------------
      COMMON/GEOCOM/IE,JE,IEM1,JEM1,LMAX,ISUBD,
     1              DELTA,AK,Y0,IC1,IC2,JC2,IF2,JF2,
     2              X(257,65),Y(257,65),XS(49,9),YS(49,9),
     3              IPC(257,65),IPS(49,9),IQMAX,Q(2,8424),
     4              ILEVP(2,5),LEVP(2,5),IP(9,10816),
     5              IPBIMX(2,5),IPBUMX,IPBDMX,IPBTMX,IPBBMX,
     6              IPBI(3,257),IPBU(3,257),IPBD(3,9),
     7              IPBT(3,33),IPBB(3,257),
     8              LEVSET(257,65),IPSET(257,65),
     9              ICONST(50),RCONST(50),LU1,LU2,LU3,LU5,LU6
      COMMON/GEOLAB/GLABEL1,GLABEL2,RLABEL1,RLABEL2,
     1             IN_NAME,OUT_NAME
      CHARACTER GLABEL1*30,GLABEL2*100,RLABEL1*10,RLABEL2*100,
     1          IN_NAME*15,OUT_NAME*15
C-----------------------------------------------------------------
C          SUBROUTINE: GEOBCEL
C-----------------------------------------------------------------
      SUBROUTINE GEOBCEL(I1,I2,I3,ICEL1,ICEL2)
C
C         THIS SUBROUTINE FINDS THE BOUNDARY CELLS
C         CORRESPONDING TO THE GIVEN BOUNDARY NODES
C
      INCLUDE 'GEOCOM.COM'
C
      ICEL1 = 0
      ICEL2 = 0
      ICON3 = 0
      I = 0
C
   10 I = I+1
      ICON1 = 0
```

```
      ICON2 = 0
      J1 = ABS(IP(1,I))
      J2 = ABS(IP(2,I))
      J3 = ABS(IP(3,I))
      J4 = ABS(IP(4,I))
C
      IF(I1.EQ.J1) THEN
        ICON1 = ICON1+1
      ELSE IF(I1.EQ.J2) THEN
        ICON1 = ICON1+1
      ELSE IF(I1.EQ.J3) THEN
        ICON1 = ICON1+1
      ELSE IF(I1.EQ.J4) THEN
        ICON1 = ICON1+1
      END IF
C
      IF(I2.EQ.J1) THEN
        ICON1 = ICON1+1
        ICON2 = ICON2+1
      ELSE IF(I2.EQ.J2) THEN
        ICON1 = ICON1+1
        ICON2 = ICON2+1
      ELSE IF(I2.EQ.J3) THEN
        ICON1 = ICON1+1
        ICON2 = ICON2+1
      ELSE IF(I2.EQ.J4) THEN
        ICON1 = ICON1+1
        ICON2 = ICON2+1
      END IF
C
      IF(I3.EQ.J1) THEN
        ICON2 = ICON2+1
      ELSE IF(I3.EQ.J2) THEN
        ICON2 = ICON2+1
      ELSE IF(I3.EQ.J3) THEN
        ICON2 = ICON2+1
      ELSE IF(I3.EQ.J4) THEN
        ICON2 = ICON2+1
      END IF
C
      IF(ICON1.EQ.2) THEN
        ICEL1 = I
        ICON3 = ICON3+1
      ENDIF
C
      IF(ICON2.EQ.2) THEN
        ICEL2 = I
        ICON3 = ICON3+1
      END IF
C
      IF(ICON3.EQ.2) THEN
        RETURN
      ELSE IF(ICON3.EQ.1) THEN
        IF(I1.EQ.0) RETURN
```

```
        IF(I3.EQ.0) RETURN
      END IF
C
      IF(I.LT.ILEVP(2,LMAX)) GO TO 10
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: GEOIN3
C-----------------------------------------------------------------
      SUBROUTINE GEOIN3
C
C         THIS SUBROUTINE READS REQUIRED INPUT
C         PARAMETERS FROM THE TERMINAL
C
      INCLUDE 'GEOCOM.COM'
C
C         LOGICAL UNIT ASSIGNMENTS FOR INPUT AND OUTPUT
C     INPUT:
C           LU1 = GLOBAL O-MESH GRID FILE
C           LU5 = INTERACTIVE INPUT
C     OUTPUT:
C           LU2 = POINTER SYSTEM FILE
C           LU3 = POINTER SYSTEM SUMMARY
C           LU6 = INTERACTIVE PROMPTS
C
      LU1 = 1
      LU2 = 2
      LU3 = 3
      LU5 = 5
      LU6 = 6
C
      WRITE(LU6,*)' ENTER AIRFOIL SECTION NAME (GLABEL1<30 CHARACTERS'
      READ(LU5,1000)GLABEL1
 1000 FORMAT(A)
      WRITE(LU6,*)' ENTER GRID COMMENTS (GLABEL2<100CHARACTERS)'
      READ(LU5,1001)GLABEL2
 1001 FORMAT(A)
      WRITE(LU6,*)' ENTER NUMBER OF GLOBAL GRID TO BE READ IN:'
      READ(LU5,*)IGRID
      WRITE(LU6,*)' ENTER TOTAL NUMBER OF GRID LEVELS'
      WRITE(LU6,*)' FOR POINTER SYSTEM:'
      READ(LU5,*)LMAX
C
C         READ GRID
      OPEN(UNIT=LU1,READONLY,
     1     TYPE='OLD',FORM='UNFORMATTED')
      DO 10 N=1,IGRID
      READ(LU1)IE,JE
      WRITE(LU6,*)'  IE,JE=' ,IE,JE
      READ(LU1)CL,((X(I,J),Y(I,J),I=1,IE),J=1,JE)
   10 CONTINUE
C
      IEM1 = IE-1
```

```
      JEM1 = JE-1
C
C         SET UP GRID STRUCTURE
      WRITE(LU6,*)' ENTER GLOBAL GRID LEVEL FOR POINTER SYSTEM:'
      READ(LU5,*)LGLOB
      DO 15 I=1,IE
      DO 15 J=1,JE
   15 LEVSET(I,J) = LGLOB
C
   18 WRITE(LU6,*)' DO YOU WISH A SUBDOMAIN? 1=YES 0=NO'
      READ(LU5,*)ISUB
      IF(ISUB.EQ.0) RETURN
      WRITE(LU6,*)' ENTER IC1,IC2,JC1,JC2 BASED ON THE GLOBAL LEVEL'
      READ(LU5,*) IC1,IC2,JC1,JC2
      WRITE(LU6,*)' ENTER LEVEL FOR THIS SUBGRID, LSUB'
      READ(LU5,*) LSUB
      LGLOBS = 2**(LGLOB-1)
      DO 20 I=LGLOBS*(IC1-1)+1,LGLOBS*(IC2-1)+1
      DO 20 J=LGLOBS*(JC1-1)+1,LGLOBS*(JC2-1)+1
   20 LEVSET(I,J) = LSUB
C
      GO TO 18
C
      END
C-----------------------------------------------------------------
C          SUBROUTINE: GEOOUT3
C-----------------------------------------------------------------
      SUBROUTINE GEOOUT3
C
C         THIS SUBROUTINE CREATES THE OUTPUT FILE
C         AND POINTER SYSTEM SUMMARY
C
      INCLUDE 'GEOCOM.COM'
C
      WRITE(LU6,*)' DO YOU WANT THE CELL POINTERS '
      WRITE(LU6,*)'           AND (X,Y) WRITTEN?'
      WRITE(LU6,*)'     ENTER 1 = YES AND 0 = NO'
      READ(LU5,*)IPRINT1
C
      WRITE(LU3,1027) GLABEL1,GLABEL2
 1027 FORMAT(5X,A30,/,5X,A100,/,5X,A10,/,5X,A100)
      WRITE(LU3,1028) IN_NAME,OUT_NAME
 1028 FORMAT(//,5X,'INPUT GRID FILE NAME:    ',A15,/,
     1          5X,'OUTPUT POINTER FILE NAME:',A15,//)
      WRITE(LU3,*) '  LEVSET(I,J)='
      DO 100 I=1,IE
      WRITE(LU3,1003) (LEVSET(I,J), J=1,JE)
  100 CONTINUE
      WRITE(LU3,*) '  IPSET(I,J)='
      DO 101 I=1,IE
      WRITE(LU3,1000) (IPSET(I,J), J=1,JE)
  101 CONTINUE
C
C         SET CONSTANTS FOR OUTPUT
```

```
      ICONST(1)  = IE
      ICONST(2)  = JE
      ICONST(3)  = IC1
      ICONST(4)  = IC2
      ICONST(5)  = JC1
      ICONST(6)  = JC2
      ICONST(7)  = IF1
      ICONST(8)  = IF2
      ICONST(9)  = JF1
      ICONST(10) = JF2
      NICONST = 50
C
      RCONST(16) = DELTA
      RCONST(17) = AK
      RCONST(18) = Y0
      NRCONST = 50
C
      WRITE(LU2) GLABEL1,GLABEL2,RLABEL1,RLABEL2
      WRITE(LU2) NICONST,NRCONST
      WRITE(LU2) (ICONST(N), N=1,NICONST)
      WRITE(LU2) (ICONST(N), N=1,NRCONST)
      WRITE(LU2) LMAX,IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
      WRITE(LU2) ((IPBIMX(M,N), M=1,2), N=1,LMAX)
      WRITE(LU3,*) '  IE,JE,LMAX,IC1,IC2,JC2,IF2,JF2'
      WRITE(LU3,*) '  IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX'
      WRITE(LU3,*) '  DELTA,AK,Y0'
      WRITE(LU3,1000) IE,JE,LMAX,IC1,IC2,JC2,IF2,JF2
      WRITE(LU3,1000) IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
      WRITE(LU3,1001) DELTA,AK,Y0
      WRITE(LU3,*) '  IPBIMX(2,LEV)='
      WRITE(LU3,1000) ((IPBIMX(M,N), M=1,2), N=1,LMAX)
 1000 FORMAT(1X,20I5)
 1001 FORMAT(1X,10E13.4)
 1003 FORMAT(1X,33I3)
C
C         WRITE OUT GRID POINTERS
      WRITE(LU2) ((ILEVP(M,N), M=1,2), N=1,LMAX)
      WRITE(LU3,*) '  ILEVP(2,LEV)='
      WRITE(LU3,1000) ((ILEVP(M,N), M=1,2), N=1,LMAX)
C
      WRITE(LU3,*) '  IP(M,N)='
      DO 10 LEV = 1,LMAX
      IF(IPRINT1.EQ.1) WRITE(LU3,*) ' LEV =',LEV
      WRITE(LU2) ((IP(M,N), M=1,9), N=ILEVP(1,LEV),ILEVP(2,LEV))
      IF(IPRINT1.EQ.1)
     1 WRITE(LU3,1002) ((IP(M,N), M=1,9), N=ILEVP(1,LEV),ILEVP(2,LEV))
   10 CONTINUE
 1002 FORMAT(1X,18I5)
C
      WRITE(LU3,*) '  IPBI(2,N)='
      DO 15 LEV=1,LMAX
      WRITE(LU3,*) ' LEV=',LEV
      IF(IPBIMX(2,LEV).NE.0)
     1  WRITE(LU2) ((IPBI(M,N), M=1,3), N=IPBIMX(1,LEV),IPBIMX(2,LEV))
```

```
      IF(IPBIMX(2,LEV).NE.0)
     1  WRITE(LU3,1002) ((IPBI(M,N), M=1,3),
     2                  N=IPBIMX(1,LEV),IPBIMX(2,LEV))
   15 CONTINUE
C
      WRITE(LU2) ((IPBU(M,N), M=1,3), N=1,IPBUMX)
      WRITE(LU2) ((IPBD(M,N), M=1,3), N=1,IPBDMX)
      WRITE(LU2) ((IPBT(M,N), M=1,3), N=1,IPBTMX)
      WRITE(LU2) ((IPBB(M,N), M=1,3), N=1,IPBBMX)
      WRITE(LU3,*) '  IPBU(M,N)='
      WRITE(LU3,1002) ((IPBU(M,N), M=1,3), N=1,IPBUMX)
      WRITE(LU3,*) '  IPBD(M,N)='
      WRITE(LU3,1002) ((IPBD(M,N), M=1,3), N=1,IPBDMX)
      WRITE(LU3,*) '  IPBT(3,N)='
      WRITE(LU3,1002) ((IPBT(M,N), M=1,3), N=1,IPBTMX)
      WRITE(LU3,*) '  IPBB(3,N)='
      WRITE(LU3,1002) ((IPBB(M,N), M=1,3), N=1,IPBBMX)
      WRITE(LU2) (Q(1,I), I=1,IQMAX)
      WRITE(LU2) (Q(2,I), I=1,IQMAX)
      TZERO = 0.0
      DO 8 KT =3,6
    8 WRITE(LU2) (TZERO, I=1,IQMAX)
C
      IF(IPRINT1.EQ.1) WRITE(LU3,*) '  Q(K,I)='
      IF(IPRINT1.EQ.1) WRITE(LU3,*) '  I=1'
      IF(IPRINT1.EQ.1) WRITE(LU3,1005) (Q(1,I), I=1,IQMAX)
      IF(IPRINT1.EQ.1) WRITE(LU3,*) '  I=2'
      IF(IPRINT1.EQ.1) WRITE(LU3,1005) (Q(2,I), I=1,IQMAX)
 1005 FORMAT(2X,(10E12.5))
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: GEOPONT4
C-----------------------------------------------------------------
      SUBROUTINE GEOPONT4
C
C         THIS SUBROUTINE GENERATES THE POINTERS
C         FOR THE GIVEN GRID AND BOUNDARY CONDITIONS.
C
      INCLUDE 'GEOCOM.COM'
C
C         SETUP AND FILLING OF Q-VECTOR
      IND = 0
      DO 10 L=1,LMAX
      LSKIP = 2**(L-1)
      DO 10 I=1,IEM1,LSKIP
      DO 10 J=1,JE,LSKIP
      IF(LEVSET(I,J).EQ.L) THEN
        IND = IND+1
        Q(1,IND) = X(I,J)
        Q(2,IND) = Y(I,J)
        IPSET(I,J) = IND
      END IF
```

```
   10 CONTINUE
C
      DO 12 J=1,JE
   12 IPSET(IE,J) = IPSET(1,J)
C
      IQMAX = IND
      WRITE(LU6,*) ' IQMAX=' , IQMAX
C
C         GENERATE POINTERS
      IND = 0
C
      DO 30 LEV=1,LMAX
      ILEVP(1,LEV) = IND+1
      LSKIP = 2**(LEV-1)
      LSKIP2 = LSKIP/2
      DO 25 I=1,IEM1,LSKIP
      DO 25 J=1,JEM1,LSKIP
      ICOUNT = 0
      IF(LEVSET(I,J).LE.LEV)              ICOUNT = ICOUNT+1
      IF(LEVSET(I,J+LSKIP).LE.LEV)        ICOUNT = ICOUNT+1
      IF(LEVSET(I+LSKIP,J+LSKIP).LE.LEV) ICOUNT = ICOUNT+1
      IF(LEVSET(I+LSKIP,J).LE.LEV)        ICOUNT = ICOUNT+1
      IF(ICOUNT.EQ.4) THEN
        IND = IND+1
        IP(1,IND) = -IPSET(I,J)
        IP(2,IND) = -IPSET(I,J+LSKIP)
        IP(3,IND) = -IPSET(I+LSKIP,J+LSKIP)
        IP(4,IND) = -IPSET(I+LSKIP,J)
        IF(LEV.EQ.1) THEN
          IF(J.GT.1) THEN
            IF(I.GT.1) THEN
              IF(LEVSET(I-1,J).EQ.1.AND.LEVSET(I,J-1).EQ.1)
     1          IP(1,IND) = -IP(1,IND)
            ELSE
              IF(LEVSET(IEM1,J).EQ.1.AND.LEVSET(I,J-1).EQ.1)
     1          IP(1,IND) = -IP(1,IND)
            END IF
            IF(I.LT.IEM1) THEN
              IF(LEVSET(I+2,J).EQ.1.AND.LEVSET(I+1,J-1).EQ.1)
     1          IP(4,IND) = -IP(4,IND)
            ELSE
              IF(LEVSET(2,J).EQ.1.AND.LEVSET(I+1,J-1).EQ.1)
     1          IP(4,IND) = -IP(4,IND)
            END IF
          END IF
          IF(J.LT.JEM1) THEN
            IF(I.GT.1) THEN
              IF(LEVSET(I-1,J+1).EQ.1.AND.LEVSET(I,J+2).EQ.1)
     1          IP(2,IND) = -IP(2,IND)
            ELSE
              IF(LEVSET(IEM1,J+1).EQ.1.AND.LEVSET(I,J+2).EQ.1)
     1          IP(2,IND) = -IP(2,IND)
            END IF
            IF(I.LT.IEM1) THEN
```

```
            IF(LEVSET(I+2,J+1).EQ.1.AND.LEVSET(I+1,J+2).EQ.1)
     1       IP(3,IND) = -IP(3,IND)
          ELSE
            IF(LEVSET(2,J+1).EQ.1.AND.LEVSET(I+1,J+2).EQ.1)
     1       IP(3,IND) = -IP(3,IND)
          END IF
        END IF
        DO 17 K=5,9
  17    IP(K,IND) = 0
      ELSE
        IF(LEVSET(I+LSKIP2,J+LSKIP2).EQ.LEV) THEN
          IF(J.GT.1) THEN
            IF(I.GT.1) THEN
              IF(LEVSET(I-LSKIP,J).LE.LEV
     1          .AND.LEVSET(I,J-LSKIP).LE.LEV)
     1         IP(1,IND) = -IP(1,IND)
            ELSE
              IF(LEVSET(IE-LSKIP,J).LE.LEV
     1          .AND.LEVSET(I,J-LSKIP).LE.LEV)
     1         IP(1,IND) = -IP(1,IND)
            END IF
            IF(I.LT.IE-LSKIP) THEN
              IF(LEVSET(I+2*LSKIP,J).LE.LEV
     1          .AND.LEVSET(I+LSKIP,J-LSKIP).LE.LEV)
     1         IP(4,IND) = -IP(4,IND)
            ELSE
              IF(LEVSET(LSKIP,J).LE.LEV
     1          .AND.LEVSET(I+LSKIP,J-LSKIP).LE.LEV)
     1         IP(4,IND) = -IP(4,IND)
            END IF
          END IF
          IF(J.LT.JE-LSKIP) THEN
            IF(I.GT.1) THEN
              IF(LEVSET(I-LSKIP,J+LSKIP).LE.LEV
     1          .AND.LEVSET(I,J+2*LSKIP).LE.LEV)
     1         IP(2,IND) = -IP(2,IND)
            ELSE
              IF(LEVSET(IE-LSKIP,J+LSKIP).LE.LEV
     1          .AND.LEVSET(I,J+2*LSKIP).LE.LEV)
     1         IP(2,IND) = -IP(2,IND)
            END IF
            IF(I.LT.IE-LSKIP) THEN
              IF(LEVSET(I+2*LSKIP,J+LSKIP).LE.LEV
     1          .AND.LEVSET(I+LSKIP,J+2*LSKIP).LE.LEV)
     1         IP(3,IND) = -IP(3,IND)
            ELSE
              IF(LEVSET(LSKIP,J+LSKIP).LE.LEV
     1          .AND.LEVSET(I+LSKIP,J+2*LSKIP).LE.LEV)
     1         IP(3,IND) = -IP(3,IND)
            END IF
          END IF
        ELSE IF(LEVSET(I+LSKIP2,J+LSKIP2).EQ.LEV-1) THEN
          IF(J.GT.1) THEN
            IF(I.GT.1) THEN
```

```
            IF(LEVSET(I-LSKIP,J).EQ.LEV
     1        .OR.LEVSET(I,J-LSKIP).EQ.LEV)
     1       IP(1,IND) = -IP(1,IND)
          ELSE
            IF(LEVSET(IE-LSKIP,J).EQ.LEV
     1        .OR.LEVSET(I,J-LSKIP).EQ.LEV)
     1       IP(1,IND) = -IP(1,IND)
          END IF
          IF(I.LT.IE-LSKIP) THEN
            IF(LEVSET(I+2*LSKIP,J).EQ.LEV
     1        .OR.LEVSET(I+LSKIP,J-LSKIP).EQ.LEV)
     1       IP(4,IND) = -IP(4,IND)
          ELSE
            IF(LEVSET(LSKIP,J).EQ.LEV
     1        .OR.LEVSET(I+LSKIP,J-LSKIP).EQ.LEV)
     1       IP(4,IND) = -IP(4,IND)
          END IF
        END IF
        IF(J.LT.JE-LSKIP) THEN
          IF(I.GT.1) THEN
            IF(LEVSET(I-LSKIP,J+LSKIP).EQ.LEV
     1        .OR.LEVSET(I,J+2*LSKIP).EQ.LEV)
     1       IP(2,IND) = -IP(2,IND)
          ELSE
            IF(LEVSET(IE-LSKIP,J+LSKIP).EQ.LEV
     1        .OR.LEVSET(I,J+2*LSKIP).EQ.LEV)
     1       IP(2,IND) = -IP(2,IND)
          END IF
          IF(I.LT.IE-LSKIP) THEN
            IF(LEVSET(I+2*LSKIP,J+LSKIP).EQ.LEV
     1        .OR.LEVSET(I+LSKIP,J+2*LSKIP).EQ.LEV)
     1       IP(3,IND) = -IP(3,IND)
          ELSE
            IF(LEVSET(LSKIP,J+LSKIP).EQ.LEV
     1        .OR.LEVSET(I+LSKIP,J+2*LSKIP).EQ.LEV)
     1       IP(3,IND) = -IP(3,IND)
          END IF
        END IF
      END IF
      IF(LEVSET(I+LSKIP2,J+LSKIP2).LT.LEV)
     1     IP(5,IND) = IPSET(I+LSKIP2,J+LSKIP2)
      IF(LEVSET(I,J+LSKIP2).LT.LEV)
     1     IP(6,IND) = IPSET(I,J+LSKIP2)
      IF(LEVSET(I+LSKIP2,J+LSKIP).LT.LEV)
     1     IP(7,IND) = IPSET(I+LSKIP2,J+LSKIP)
      IF(LEVSET(I+LSKIP,J+LSKIP2).LT.LEV)
     1     IP(8,IND) = IPSET(I+LSKIP,J+LSKIP2)
      IF(LEVSET(I+LSKIP2,J).LT.LEV)
     1     IP(9,IND) = IPSET(I+LSKIP2,J)
    END IF
   END IF
25 CONTINUE
   ILEVP(2,LEV) = IND
   WRITE(LU6,*)' ILEVP(2,LEV)=',ILEVP(2,LEV)
```

```
   30 CONTINUE
C
C          INTERPOLATION BC POINTER
       IND = 0
       DO 47 LEV = 2,LMAX
       LEVM1 = LEV-1
       LSKIP = 2**(LEV-1)
       LSKIP2 = LSKIP/2
       IPBIMX(1,LEV-1) = IND+1
       DO 45 I=1,IEM1,LSKIP
       DO 45 J=1,JEM1,LSKIP
       IF(LEVSET(I+LSKIP2,J+LSKIP2).EQ.LEV) THEN
         IF(LEVSET(I,J+LSKIP2).EQ.LEV-1) THEN
           IND = IND+1
           IPBI(1,IND) = IPSET(I,J+LSKIP)
           IPBI(2,IND) = IPSET(I,J+LSKIP2)
           IPBI(3,IND) = IPSET(I,J)
         END IF
         IF(LEVSET(I+LSKIP2,J+LSKIP).EQ.LEVM1) THEN
           IND = IND+1
           IPBI(1,IND) = IPSET(I+LSKIP,J+LSKIP)
           IPBI(2,IND) = IPSET(I+LSKIP2,J+LSKIP)
           IPBI(3,IND) = IPSET(I,J+LSKIP)
         END IF
         IF(LEVSET(I+LSKIP,J+LSKIP2).EQ.LEVM1) THEN
           IND = IND+1
           IPBI(1,IND) = IPSET(I+LSKIP,J)
           IPBI(2,IND) = IPSET(I+LSKIP,J+LSKIP2)
           IPBI(3,IND) = IPSET(I+LSKIP,J+LSKIP)
         END IF
         IF(LEVSET(I+LSKIP2,J).EQ.LEVM1) THEN
           IND = IND+1
           IPBI(1,IND) = IPSET(I,J)
           IPBI(2,IND) = IPSET(I+LSKIP2,J)
           IPBI(3,IND) = IPSET(I+LSKIP,J)
         END IF
       END IF
   45 CONTINUE
       IF(IPBIMX(1,LEV-1).GT.IND) THEN
         IPBIMX(1,LEV-1) = 0
         IPBIMX(2,LEV-1) = 0
       ELSE
         IPBIMX(2,LEV-1) = IND
       END IF
   47 CONTINUE
C
C          FARFIELD & SOLID WALL BC POINTERS
       IND = 0
       I = 0
   50 I = I+1
       IF(IPSET(I,JE).GT.0) THEN
         IND = IND+1
         IPBU(2,IND) = IPSET(I,JE)
         IF(IND.GT.1) THEN
```

```
          IPBU(3,IND-1) = IPBU(2,IND)
          IPBU(1,IND)   = IPBU(2,IND-1)
        END IF
      END IF
      IF(I.LT.IE) GO TO 50
C
      IPBU(1,1) = IPBU(1,IND)
      IPBUMX = IND-1
C
      IND = 0
      I = 0
   51 I = I+1
      IF(IPSET(I,1).GT.0) THEN
        IND = IND+1
        IPBB(2,IND) = IPSET(I,1)
        IF(IND.GT.1) THEN
          IPBB(3,IND-1) = IPBB(2,IND)
          IPBB(1,IND)   = IPBB(2,IND-1)
        END IF
      END IF
      IF(I.LT.IE) GO TO 51
C
      IPBB(1,1) = IPBB(1,IND)
      IPBBMX = IND-1
C
      DO 52 IND=1,IPBUMX
      I1 = IPBU(1,IND)
      I2 = IPBU(2,IND)
      I3 = IPBU(3,IND)
      CALL GEOBCEL(I1,I2,I3,ICEL1,ICEL2)
      IPBU(1,IND) = ICEL1
      IPBU(2,IND) = ICEL2
      IPBU(3,IND) = 2
   52 CONTINUE
C
      DO 53 IND=1,IPBBMX
      I1 = IPBB(1,IND)
      I2 = IPBB(2,IND)
      I3 = IPBB(3,IND)
      CALL GEOBCEL(I1,I2,I3,ICEL1,ICEL2)
      IPBB(1,IND) = ICEL1
      IPBB(2,IND) = ICEL2
      IF(IND.GT.1) THEN
        IPBB(3,IND) = 4
      ELSE
        IPBB(3,IND) = 5
      END IF
   53 CONTINUE
C
C         OTHER POINTERS
      IPBDMX = 0
      IPBTMX = 0
C
      RETURN
```

```
      END
C-------------------------------------------------------------
C          LINK COMMAND FILE: GEOLINK.COM
C-------------------------------------------------------------
$LINK GEOCREAT,GEOBCEL,GEOIN3,GEOPONT4,GEOOUT3
C-------------------------------------------------------------
C*************************************************************
C-------------------------------------------------------------
```

```
12345678901234567890123456789012345678901234567890123456789012345
C-----------------------------------------------------------------
C*****************************************************************
C-----------------------------------------------------------------
C
C-----------------------------------------------------------------
C          PROGRAM:O-MESH EULERCELL
C-----------------------------------------------------------------
C         PROGRAM EULERCELL SOLVES THE 2-D EULER EQN'S
C         USING A CELL POINTER BASED VERSION OF
C         NI'S METHOD.  IT INCLUDES THE CAPABILITY
C         OF ANY NUMBER OF SUBDOMAINS.
C
C
C         READ INPUT PROPERTIES
      CALL INPUT2
C
C         CALCULATE CONSTANTS CONTAINING GAMMA
      CALL GAMMAS
C
C         INITIALIZE FLOW FIELD TO UNIFORM FLOW
      CALL INITIA
C
C         SOLVE EULER EQN'S USING NI'S METHOD
      CALL NI
C
C         OUTPUT FINAL SOLUTION
      CALL OUTPUT3
C
      STOP
      END
C-----------------------------------------------------------------
C          INCLUDE FILE: GAM.INC
C-----------------------------------------------------------------
      COMMON/GAM/ GAMMA,HTOT,
     1            GM1,GM1D2,GM1DG,GM1D2G,
     2            GP1DG,GP1D2G,GM3
C-----------------------------------------------------------------
C          INCLUDE FILE: INPT.INC
C-----------------------------------------------------------------
      COMMON/INPT/ AMFS,CFL,AVISCF,EXITP,ITIM,
     1             ISTART,NSTART,NMAX,LMAX,
     2             LSTOP,DELSTP,IPRNT1,IPRNT2,
     3             WCFS(4),DELTA,AK,Y0,
     4             IE,JE,IC1,IC2,JC2,IF2,JF2,
     5             ALPHA,ROFS,APFS,UFS,VFS,
     6             NFINSH,DELMAX1(5),
     7             ICONST(50),RCONST(50),
     8             INSSWT,RE0,PR,CSTAR,TREF
      COMMON/INPTLAB/GLABEL1,GLABEL2,RLABEL1,RLABEL2
      CHARACTER GLABEL1*30,GLABEL2*100,RLABEL1*10,RLABEL2*100
C-----------------------------------------------------------------
C          INCLUDE FILE: LUNITS.INC
C-----------------------------------------------------------------
```

```
      COMMON/LUNITS/LU1,LU2,LU3,LU4,LU5,LU6,LU7
C-----------------------------------------------------------------
C           INCLUDE FILE: MAIN.INC
C-----------------------------------------------------------------
      COMMON/MAIN/T,DT,DTE(82),DELMAX(5),IMAX,
     1            IQMAX,LEVP(2,5),IPBIMX(2,5),IPBUMX,IPBDMX,
     2            IPBTMX,IPBBMX,Q(10,8424),IP(9,10816),IPBI(3,257),
     3            IPBU(3,257),IPBD(3,9),IPBT(3,33),IPBB(3,257),
     4            QIB(4,257),EDT(8424)
C-----------------------------------------------------------------
C           INCLUDE FILE: MET.INC
C-----------------------------------------------------------------
      COMMON/MET/ DV,DL,DM,DXDXI,DYDXI,DXDET,DYDET,
     1            DXIDX,DXIDY,DETDX,DETDY,AJAC
      COMMON/MET2/ IWP1(258),TX(258),TY(258),
     1             RC(258),TSCL(258),SSCL(258)
C-----------------------------------------------------------------
C           INCLUDE FILE: POINT.INC
C-----------------------------------------------------------------
      COMMON/POINT/ I1,I2,I3,I4,INC,IN1,IN2,IN3,IN4,
     1              VIS1,VIS2,VIS3,VIS4,IVIS
C-----------------------------------------------------------------
C           INCLUDE FILE: SOLV.INC
C-----------------------------------------------------------------
      COMMON/SOLV/ F(4,4),G(4,4),DELU(4),DELF(4),DELG(4)
C-----------------------------------------------------------------
C           SUBROUTINE: BDSMTH
C-----------------------------------------------------------------
      SUBROUTINE BDSMTH(LEV)
C
C         This subroutine smooths the far field and solid
C         wall boundary points.  Points are always smoothed
C         on the lowest level in which the two adjoining cell
C         to the boundary exist.  This is consistent with the
C         internal point smoothing.
C             For the far field boundary the smoothing used is
C         the corresponding one model applied along the boundary.
C             For the solid wall boundary two formulations are
C         possible:
C           Type 1:  The same as the farfield boundary with a
C                    ramp increase in smoothing around the t.e.
C           Type 2:  A standard internal smoothing using extrapolated
C                    information to define an imaginary line of points
C                    inside the wall.  In this case the smoothing is not
C                    increased in the t.e. region.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'GAM.INC'
C
      DIMENSION QN1(4),QN2(4),QAVE2(4)
C
```

```
C        Far Field boundary point smoothing using a 1-D smoothing
C        tangent to the boundary
      DO 10 I=1,IPBUMX
C
C        Is point to be smoothed on this level?
      ICONT1 = 0
      IF(IPBU(1,I).GE.LEVP(1,LEV).AND.IPBU(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBU(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBU(2,I).GE.LEVP(1,LEV).AND.IPBU(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBU(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C        If it is then calculate and add contributions
C        for each cell surrounding the point.
      IF(ICONT1.EQ.1) THEN
C
C        First cell:
        CALL CELPOINT(IPBU(1,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
        DO 4 K=1,4
        KP2 = K+2
        KP6 = K+6
    4   Q(KP6,I3) = Q(KP6,I3)+0.125*AVIS*(Q(KP2,I2)-Q(KP2,I3))
C
C        Second Cell:
        CALL CELPOINT(IPBU(2,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
        DO 6 K=1,4
        KP2 = K+2
        KP6 = K+6
    6   Q(KP6,I2) = Q(KP6,I2)+0.125*AVIS*(Q(KP2,I3)-Q(KP2,I2))
      END IF
   10 CONTINUE
C
C        Solid Wall Boundary point Smoothing
C        Possible forms:
C            IBCOND = 1  1-D tangent smoothing model
C                        with ramp increase at t.e.
C                   = 2  For reflected points and standard
C                        internal point smoothing model.
C
C        Constants
      IBCOND = 1
      JTESMTH = 5
      TECOEF  = 4.0
      IF(IBCOND.EQ.2) GO TO 40
C
```

```
C         Type 1: 1D smoothing formulation
      DO 30 I=1,IPBBMX
CCC      IF(IPBB(3,I).NE.4) GO TO 30
C
C         Is point to be smoothed on this level?
      ICONT1 = 0
      IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C         If yes, calculate and add contributions form
C         both cells surrounding the cell
      IF(ICONT1.EQ.1) THEN
C
C           First cell:
        CALL CELPOINT(IPBB(1,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Ramp smoothing near t.e.
        IF(I.LE.JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
        ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
     1             /FLOAT(JTESMTH))*AVIS
        END IF
C
        DO 24 K=1,4
        KP2 = K+2
        KP6 = K+6
          Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(Q(KP2,I1)-Q(KP2,I4))
   24   CONTINUE
C
C           Second cell:
        CALL CELPOINT(IPBB(2,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Ramp soothing near t.e.
        IF(I.LE.JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
        ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
     1             /FLOAT(JTESMTH))*AVIS
        END IF
C
        DO 26 K=1,4
        KP2 = K+2
        KP6 = K+6
```

```
          Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(Q(KP2,I4)-Q(KP2,I1))
   26   CONTINUE
      END IF
   30 CONTINUE
C
      RETURN
C
C         Type 2:  Reflection wall smoothing
   40 CONTINUE
      DO 70 I=1,IPBBMX
      IF(IPBB(3,I).NE.4) GO TO 70
C
C         Is this point to be smoothed on this level?
      ICONT1 = 0
      IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C         If Yes, calculate and add contributions form both cells
      IF(ICONT1.EQ.1) THEN
C
C           Calculate surface tangent vector (dx,dy)
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
C
C           First cell:
        CALL CELPOINT(IPBB(1,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
```

```
C           Extrapolate r,p,h0 and reflect u,v
        P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
        P3 = GM1*(Q(6,I3)-0.5*(Q(4,I3)*Q(4,I3)+Q(5,I3)*Q(5,I3))/Q(3,I3))
        PI4 = P3-2.*(P3-P4)
        H4 = (Q(6,I4)+P4)/Q(3,I4)
        H3 = (Q(6,I3)+P3)/Q(3,I3)
        HI4 = H3
        VELT3 = (Q(4,I3)*DX+Q(5,I3)*DY)/Q(3,I3)
        VELN3 = (-Q(4,I3)*DY+Q(5,I3)*DX)/Q(3,I3)
        UI4 = VELT3*DX+VELN3*DY
        VI4 = VELT3*DY-VELN3*DX
        RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
        EI4 = RI4*HI4-PI4
        P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
        P2 = GM1*(Q(6,I2)-0.5*(Q(4,I2)*Q(4,I2)+Q(5,I2)*Q(5,I2))/Q(3,I2))
        PI1 = P2-2.*(P2-P1)
        H1 = (Q(6,I1)+P1)/Q(3,I1)
        H2 = (Q(6,I2)+P2)/Q(3,I2)
        HI1 = H2
        VELT2 = (Q(4,I2)*DX+Q(5,I2)*DY)/Q(3,I2)
        VELN2 = (-Q(4,I2)*DY+Q(5,I2)*DX)/Q(3,I2)
        UI1 = VELT2*DX+VELN2*DY
        VI1 = VELT2*DY-VELN2*DX
        RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
        EI1 = RI1*HI1-PI1
C
C           Find reflected cell center values
        QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
        QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
        QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
        QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C           Add contribution
        DO 64 K=1,4
        KP2 = K+2
        KP6 = K+6
        QAVE1 = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
C
        Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I4))
   64   CONTINUE
C
C           Second Cell:
        CALL CELPOINT(IPBB(2,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Extrapolate r,p,h0 and reflect u,v
        P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
        P3 = GM1*(Q(6,I3)-0.5*(Q(4,I3)*Q(4,I3)+Q(5,I3)*Q(5,I3))/Q(3,I3))
        PI4 = P3-2.*(P3-P4)
        H4 = (Q(6,I4)+P4)/Q(3,I4)
        H3 = (Q(6,I3)+P3)/Q(3,I3)
        HI4 = H3
```

```
        VELT3 = (Q(4,I3)*DX+Q(5,I3)*DY)/Q(3,I3)
        VELN3 = (-Q(4,I3)*DY+Q(5,I3)*DX)/Q(3,I3)
        UI4 = VELT3*DX+VELN3*DY
        VI4 = VELT3*DY-VELN3*DX
        RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
        EI4 = RI4*HI4-PI4
        P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
        P2 = GM1*(Q(6,I2)-0.5*(Q(4,I2)*Q(4,I2)+Q(5,I2)*Q(5,I2))/Q(3,I2))
        PI1 = P2-2.*(P2-P1)
        H1 = (Q(6,I1)+P1)/Q(3,I1)
        H2 = (Q(6,I2)+P2)/Q(3,I2)
        HI1 = H2
        VELT2 = (Q(4,I2)*DX+Q(5,I2)*DY)/Q(3,I2)
        VELN2 = (-Q(4,I2)*DY+Q(5,I2)*DX)/Q(3,I2)
        UI1 = VELT2*DX+VELN2*DY
        VI1 = VELT2*DY-VELN2*DX
        RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
        EI1 = RI1*HI1-PI1
C
C           Find reflected cell center values
        QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
        QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
        QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
        QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C           Add contribution
        DO 66 K=1,4
        KP2 = K+2
        KP6 = K+6
        QAVE1 = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
C
        Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I1))
   66   CONTINUE
      END IF
   70 CONTINUE
C
      RETURN
      END
C-------------------------------------------------------------
C           SUBROUTINE: CELPOINT
C-------------------------------------------------------------
      SUBROUTINE CELPOINT(I)
C
C         Subroutine CELPOINT sets up local cell pointer names
C         for the current cell I from the cell pointer system.
C         In addition to the definition of the cell smoothing
C         switches are set for the 4 corner nodes.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
C
C         Define corner nodes
      I1 = ABS(IP(1,I))
      I2 = ABS(IP(2,I))
```

```
      I3 = ABS(IP(3,I))
      I4 = ABS(IP(4,I))
C
C         Define interpolation pointers
      INC = IP(5,I)
      IN1 = IP(6,I)
      IN2 = IP(7,I)
      IN3 = IP(8,I)
      IN4 = IP(9,I)
C
C         Set smoothing switches based on the sign
C         of the corner pointers
      IF(IP(1,I).GE.0) THEN
        VIS1 = 1.0
      ELSE
        VIS1 = 0.0
      END IF
      IF(IP(2,I).GE.0) THEN
        VIS2 = 1.0
      ELSE
        VIS2 = 0.0
      END IF
      IF(IP(3,I).GE.0) THEN
        VIS3 = 1.0
      ELSE
        VIS3 = 0.0
      END IF
      IF(IP(4,I).GE.0) THEN
        VIS4 = 1.0
      ELSE
        VIS4 = 0.0
      END IF
      IVIS = VIS1+VIS2+VIS3+VIS4
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: CTIME
C-----------------------------------------------------------------
      SUBROUTINE CTIME
C
C         Subroutine CTIME calculates the maximum stable time step
C         for the current cell based on the following equation:
C
C     (DT)    = CFL*MIN{ DV/(|U*Y  -V*X  |+A*DL), DV/(|U*Y  -V*X  |+A*DM) }
C         MAX                    ET    ET                XI    XI
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'LUNITS.INC'
      DIMENSION QAVE(4)
```

```
C
C         FIND MIN OF DX/(|U|+A) AND DY/(|V|+A)
      DO 2 K=1,4
      KP2 = K+2
      IF(INC.EQ.0) THEN
         QAVE(K) = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
      ELSE
         QAVE(K) = Q(KP2,INC)
      END IF
    2 CONTINUE
      UTEMP = QAVE(2)/QAVE(1)
      VTEMP = QAVE(3)/QAVE(1)
      A2    = GAMMA*GM1*(QAVE(4)/QAVE(1)
     1                  -0.5*(UTEMP*UTEMP+VTEMP*VTEMP))
      IF (A2.LT.0.0) THEN
         WRITE(LU1,*) '** ERROR IN CTIME A2<0, I=',I
         STOP
      END IF
      A     = SQRT(A2)
C
      DTA = DV/(ABS(UTEMP*DYDET-VTEMP*DXDET)+A*DL)
      DTB = DV/(ABS(UTEMP*DYDXI-VTEMP*DXDXI)+A*DM)
C
      DT = CFL*MIN(DTA,DTB)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: DELTFG
C-----------------------------------------------------------------
      SUBROUTINE DELTFG
C
C         THIS SUBROUTINE CALCULATES DELF AND DELG
C         USING THE VERY EFFICIENT DELTA FORM
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'SOLV.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'POINT.INC'
      DIMENSION QAVE(4),DF(4),DG(4)
C
C         CALCULATE THE CELL AVERAGE U
      DO 1  K = 1,4
      KP2 = K+2
      IF(INC.EQ.0) THEN
         QAVE(K) = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
      ELSE
         QAVE(K) = Q(KP2,INC)
      END IF
    1 CONTINUE
C
C         FIND DF AND DG
      W1 = QAVE(2)/QAVE(1)
```

```
      W2 = QAVE(3)/QAVE(1)
      W3 = GAMMA*QAVE(4)/QAVE(1)-GM1D2*(W1*W1+W2*W2)
      W4 = DELU(2)-W1*DELU(1)
      W5 = DELU(3)-W2*DELU(1)
      W6 = W1*(DELU(2)+W4)
      W7 = W2*(DELU(3)+W5)
      W8 = GM1*(DELU(4)-0.5*(W6+W7))
      W9 = DELU(4)+W8-W3*DELU(1)
      DTDV = DT/DV
C
      DF(1) = DELU(2)
      DF(2) = W6+W8
      DF(3) = W2*DELU(2)+W1*W5
      DF(4) = W3*DELU(2)+W1*W9
C
      DG(1) = DELU(3)
      DG(2) = DF(3)
      DG(3) = W7+W8
      DG(4) = W3*DELU(3)+W2*W9
C
C         CALCULATE DELF AND DELG
      DO 4 K=1,4
      DELF(K) = (DYDET*DF(K)-DXDET*DG(K))*DTDV
      DELG(K) = (DXDXI*DG(K)-DYDXI*DF(K))*DTDV
    4 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: DELTRS
C-----------------------------------------------------------------
      SUBROUTINE DELTRS
C         This subroutine calculates the Navier-Stokes
C         viscous terms in a manor similar to the artifical
C         viscousity and adds them to the DU's.
      INCLUDE 'MAIN.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
C
      DIMENSION QAVE(4),DR(4),DS(4),DQDXI(4),DQDET(4),
     1          DELR(4),DELS(4)
C
C         Calculate average properties for cell center
      DO 5 K=1,4
      KP2 = K+2
    5 QAVE(K) = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
      RAVE = QAVE(1)
      UAVE = QAVE(2)/QAVE(1)
      VAVE = QAVE(3)/QAVE(1)
      TAVE = GAMMA*GM1*(QAVE(4)
     1    -0.5*(QAVE(2)*QAVE(2)+QAVE(3)*QAVE(3))/QAVE(1))/QAVE(1)
      AMUE = (TAVE**(3./2.))*(1.+CSTAR)/(TAVE+CSTAR)
```

```
C
C         Calculate cell center gradients in coordinate directions
      DO 10 K=1,3
      KP2 = K+2
      DQDXI(K) = 0.5*(Q(KP2,I3)+Q(KP2,I4)-Q(KP2,I1)-Q(KP2,I2))
   10 DQDET(K) = 0.5*(Q(KP2,I2)+Q(KP2,I3)-Q(KP2,I1)-Q(KP2,I4))
C
      TI1 = GAMMA*GM1*(Q(6,I1)
     1    -0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))/Q(3,I1)
      TI2 = GAMMA*GM1*(Q(6,I2)
     1    -0.5*(Q(4,I2)*Q(4,I2)+Q(5,I2)*Q(5,I2))/Q(3,I2))/Q(3,I2)
      TI3 = GAMMA*GM1*(Q(6,I3)
     1    -0.5*(Q(4,I3)*Q(4,I3)+Q(5,I3)*Q(5,I3))/Q(3,I3))/Q(3,I3)
      TI4 = GAMMA*GM1*(Q(6,I4)
     1    -0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))/Q(3,I4)
      DTDXI = 0.5*(TI3+TI4-TI1-TI2)
      DTDET = 0.5*(TI2+TI3-TI1-TI4)
C
C         define constants
      C1 = 4./3.
      C2 = 2./3.
      C3 = -AMUE/(RAVE*DV*RE0)
      C4 = -AMUE/(GM1*PR*DV*RE0)
C
C         Calculate stress components
      DUDX = DYDET*DQDXI(2)-DYDXI*DQDET(2)
     1      -UAVE*(DYDET*DQDXI(1)-DYDXI*DQDET(1))
      DVDY =-DXDET*DQDXI(3)+DXDXI*DQDET(3)
     1      -VAVE*(-DXDET*DQDXI(1)+DXDXI*DQDET(1))
      TXX = C3*(C1*DUDX-C2*DVDY)
      TYY = C3*(C1*DVDY-C2*DUDX)
      TXY = C3*(-DXDET*DQDXI(2)+DXDXI*DQDET(2)
     1          +DYDET*DQDXI(3)-DYDXI*DQDET(3)
     2          +(UAVE*DXDET-VAVE*DYDET)*DQDXI(1)
     3          -(UAVE*DXDXI-VAVE*DYDXI)*DQDET(1))
C
C         Calculate DR(k) and DS(k)
      DR(2) = TXX
      DR(3) = TXY
      DR(4) = UAVE*TXX+VAVE*TXY
     1      +C4*(DYDET*DTDXI-DYDXI*DTDET)
      DS(2) = TXY
      DS(3) = TYY
      DS(4) = VAVE*TYY+UAVE*TXY
     1      +C4*(-DXDET*DTDXI+DXDXI*DTDET)
C
C         Calculate DELR(k) and DELS(k)
      DT2DV = 2.0*DT/DV
      DO 15 K=2,4
      DELR(K) = (DYDET*DR(K)-DXDET*DS(K))*DT2DV
      DELS(K) = (DXDXI*DS(K)-DYDXI*DR(K))*DT2DV
   15 CONTINUE
C
C         Distribute to cell corner nodes
```

```
      DO 20 K=2,4
      KP6 = K+6
      Q(KP6,I1) = Q(KP6,I1)+0.25*(-DELR(K)-DELS(K))
      Q(KP6,I2) = Q(KP6,I2)+0.25*(-DELR(K)+DELS(K))
      Q(KP6,I3) = Q(KP6,I3)+0.25*(DELR(K)+DELS(K))
      Q(KP6,I4) = Q(KP6,I4)+0.25*(DELR(K)-DELS(K))
   20 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: DELTU
C-----------------------------------------------------------------
      SUBROUTINE DELTU
C
C         Subroutine DELTU performs a flux balance over the current
C         cell.  If this is a fine mesh cell then the flux balance
C         is actually calculated, otherwise for coarser mesh cells
C         simple injection is used.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'SOLV.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'POINT.INC'
C
C         If this is a fine mesh cell the following flux balance
C         is performed
      IF(INC.EQ.0) THEN
C
C           Calculate cell side lengths
        XA = Q(1,I2)-Q(1,I1)
        XB = Q(1,I3)-Q(1,I2)
        XC = Q(1,I3)-Q(1,I4)
        XD = Q(1,I4)-Q(1,I1)
        YA = Q(2,I2)-Q(2,I1)
        YB = Q(2,I3)-Q(2,I2)
        YC = Q(2,I3)-Q(2,I4)
        YD = Q(2,I4)-Q(2,I1)
        DTDV2 = 0.5*DT/DV
C
C           Find F and G at the cell corner nodes
        CALL FINDFG(I1,1)
        CALL FINDFG(I2,2)
        CALL FINDFG(I3,3)
        CALL FINDFG(I4,4)
C
C           Perform flux balance using simple
C           averaging alone cell sides
        DO 8 K=1,4
    8   DELU(K)=DTDV2*(((F(K,1)+F(K,2))*YA-(G(K,1)+G(K,2))*XA)
     2                -((F(K,3)+F(K,4))*YC-(G(K,3)+G(K,4))*XC)
     3                +((G(K,1)+G(K,4))*XD-(F(K,1)+F(K,4))*YD)
     4                -((G(K,2)+G(K,3))*XB-(F(K,2)+F(K,3))*YB))
   10   CONTINUE
```

```
C
C         If this is a coarse cell use simple injection
C         of fine grid DU's for value of flux balance.
      ELSE
         DO 20 K=1,4
   20    DELU(K) = Q(K+6,INC)
      END IF
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: EULERWAL
C-----------------------------------------------------------------
      SUBROUTINE EULERWAL(LEV)
C
C         This subroutine preforms a stream line intergration
C         of the euler eqn. in natural coordinates to inforce
C         the solid wall no normal flow boundary condition.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'MET.INC'
C
      DIMENSION PIP1(257),PEX(257),HOEX(257),TX(257),TY(257),
     1          UT(257),DUT(257),QAVE(4)
      DIMENSION QN1(4),QN2(4),QAVE2(4),QSAV(6,257)
C
C         Calculate pressures at ring of points just inside the
C         flow ajoining the boundary
      DO 10 I=1,IPBBMX
C
      J1 = ABS(IP(3,IPBB(1,I)))
C
      PIP1(I) = GM1*(Q(6,J1)-0.5*(Q(4,J1)**2+Q(5,J1)**2)/Q(3,J1))
   10 CONTINUE
C
C         Calculate extrapolated values at surface
      II   = 1
      IIP1 = 2
C
      DO 20 I=1,IPBBMX+1
      IF(I.LE.IPBBMX) THEN
        JPM1 = ABS(IP(1,IPBB(1,I)))
        JP   = ABS(IP(4,IPBB(1,I)))
        JPP1 = ABS(IP(4,IPBB(2,I)))
      ELSE
        JPM1 = ABS(IP(1,IPBB(1,1)))
        JP   = ABS(IP(4,IPBB(1,1)))
        JPP1 = ABS(IP(4,IPBB(2,1)))
      END IF
C
C         calculate normal vector
```

```
      TMP1 = Q(1,JP)-Q(1,JPM1)
      TMP2 = Q(2,JP)-Q(2,JPM1)
      DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
      TX1 = TMP1/DS1
      TY1 = TMP2/DS1
      TMP1 = Q(1,JPP1)-Q(1,JP)
      TMP2 = Q(2,JPP1)-Q(2,JP)
      DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
      TX2 = TMP1/DS2
      TY2 = TMP2/DS2
C
      TMP1 = DS1+DS2
      TMP2 = DS2/(DS1*TMP1)
      TMP3 = (DS2-DS1)/(DS1*DS2)
      TMP4 = DS1/(DS2*TMP1)
C
      RC = 0.5*TMP1/SQRT((TX2-TX1)**2+(TY2-TY1)**2)

      DXDS = -Q(1,JPM1)*TMP2+Q(1,JP)*TMP3+Q(1,JPP1)*TMP4
      DYDS = -Q(2,JPM1)*TMP2+Q(2,JP)*TMP3+Q(2,JPP1)*TMP4
      TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
      DX = DXDS/TMP1
      DY = DYDS/TMP1
      IF(I.EQ.1) THEN
        DXTMP = DX
        DYTMP = DY
        DX = TX2
        DY = TY2
        RC = 1.E+20
      ELSE IF(I.EQ.IPBBMX+1) THEN
        DX = TX1
        DY = TY1
        RC = 1.E+20
      END IF
C
      DNX = -DY
      DNY =  DX
C
C          Search for intersection of normal line and ring
      AP1 = Q(1,JP)
      AP2 = DNX
      BP1 = Q(2,JP)
      BP2 = DNY
C
   11 CONTINUE
      JN   = ABS(IP(3,IPBB(1,II)))
      JNP1 = ABS(IP(3,IPBB(1,IIP1)))
C
      A1 = Q(1,JN)
      A2 = Q(1,JNP1)-A1
      B1 = Q(2,JN)
      B2 = Q(2,JNP1)-B1
C
```

```
      DEL = AP2*B2-BP2*A2
      T = (AP2*(BP1-B1)-BP2*(AP1-A1))/DEL
C
      ITMP = 0
      IF(T.LT.0.0) THEN
        ITMP = 1
        IIP1 = II
        IF(II.GT.1) THEN
          II = II-1
        ELSE
          II = IPBBMX
        END IF
      ELSE IF(T.GT.1.0) THEN
        ITMP = 1
        II = IIP1
        IF(IIP1.LT.IPBBMX) THEN
          IIP1 = IIP1+1
        ELSE
          IIP1 = 1
        END IF
      END IF
      IF(ITMP.EQ.1) GO TO 11
C
      S = (A2*(BP1-B1)-B2*(AP1-A1))/DEL
C
      PINT = PIP1(II)+T*(PIP1(IIP1)-PIP1(II))
      TMP1 = (Q(6,JN)+PIP1(II))/Q(3,JN)
      TMP2 = (Q(6,JNP1)+PIP1(IIP1))/Q(3,JNP1)
      HOINT = TMP1+T*(TMP2-TMP1)
C
      DO 12 K=1,6
      QSAV(K,I) = Q(K,JN)+T*(Q(K,JNP1)-Q(K,JN))
   12 CONTINUE
C
C         Extrapolate to surface using normal momentum eqn
C         for pressure and zeroth order extrapolation for
C         total enthalpy
      PEX(I) = PINT
      HOEX(I) = HOINT
      IF(I.GT.1) THEN
        TX(I) = DX
        TY(I) = DY
      ELSE
        TX(1) = DXTMP
        TY(1) = DYTMP
      END IF
      UT(I) = (DX*Q(4,JP)+DY*Q(5,JP))/Q(3,JP)
      PEX(I) = PINT-Q(3,JP)*UT(I)*UT(I)*S/RC
   20 CONTINUE
C
C         Correct t/e pressure to average of upper and lower
C         points just upstream of t/e and set tangent to bisector
      PEX(IPBBMX+1)  = 0.5*(PEX(1)+PEX(IPBBMX+1))
      HOEX(IPBBMX+1) = 0.5*(HOEX(1)+HOEX(IPBBMX+1))
```

```
      TX(IPBBMX+1)   = -TY(1)
      TY(IPBBMX+1)   = TX(1)
      JP = ABS(IP(4,IPBB(1,1)))
      UT(IPBBMX+1)   = (TX(IPBBMX+1)*Q(4,JP)
     1                 +TY(IPBBMX+1)*Q(5,JP))/Q(3,JP)
      PEX(1)  = PEX(IPBBMX+1)
      HOEX(1) = HOEX(IPBBMX+1)
      TX(1)   = -TX(IPBBMX+1)
      TY(1)   = -TY(IPBBMX+1)
      UT(1)   = -UT(IPBBMX+1)
C
C         Solve streamline euler eqn.
C
C         Zero wall DU's
      DO 30 I=1,IPBBMX+1
      DUT(I) = 0.0
   30 CONTINUE
C
C        SWEEP CELL BY CELL AND CALCULATE DU'S
      DO 40 I=1,IPBBMX
      IP1 = I+1
      J1 = ABS(IP(1,IPBB(2,I)))
      J2 = ABS(IP(4,IPBB(2,I)))
      DS = SQRT((Q(1,J2)-Q(1,J1))**2+(Q(2,J2)-Q(2,J1))**2)
C
      DO 32 K=1,4
      KP2 = K+2
      QAVE(K) = 0.5*(Q(KP2,J1)+Q(KP2,J2))
   32 CONTINUE
C
      UTEMP = QAVE(2)/QAVE(1)
      VTEMP = QAVE(3)/QAVE(1)
      A2    = GAMMA*GM1*(QAVE(4)/QAVE(1)
     1                   -0.5*(UTEMP*UTEMP+VTEMP*VTEMP))
      A     = SQRT(A2)
C
      DT = CFL*DS/(SQRT(UTEMP**2+VTEMP**2)+A)
C
      CHNU = 0.5*((UT(I)**2-UT(IP1)**2)
     1    +(1./Q(3,J1)+1./Q(3,J2))*(PEX(I)-PEX(IP1)))*DT/DS
      CHNE = 0.5*(UT(I)+UT(IP1))*CHNU*DT/DS
C
      DUT(I)   = DUT(I)  +0.5*(CHNU-CHNE)
      DUT(IP1) = DUT(IP1)+0.5*(CHNU+CHNE)
   40 CONTINUE
C
      DUT(1) = DUT(1)-DUT(IPBBMX+1)
C
C         Calculate new conditions and correct DQ's
      DO 50 I=1,IPBBMX
CCC      IF(IPBB(3,I).NE.4) GOTO 50
      JP = ABS(IP(4,IPBB(1,I)))
      UT(I) = UT(I)+DUT(I)
C
```

```
      RNEW = PEX(I)/(GM1DG*(H0EX(I)-0.5*UT(I)*UT(I)))
      UNEW = TX(I)*UT(I)
      VNEW = TY(I)*UT(I)
      ENEW = RNEW*H0EX(I)-PEX(I)
C
      Q(7,JP)  = RNEW      -Q(3,JP)
      Q(8,JP)  = RNEW*UNEW -Q(4,JP)
      Q(9,JP)  = RNEW*VNEW -Q(5,JP)
      Q(10,JP) = ENEW      -Q(6,JP)
   50 CONTINUE
C
C         This section smooths the solid wall boundary points.
C         Points are always smoothed on the lowest level in
C         which the two adjoining cell to the boundary exist.
C         This is consistent with the internal point smoothing.
C            For the solid wall boundary two formulations are
C         possible:
C
C           IBCOND = 0  no smoothing applied, just return
C
C                  = 1  1-D tangent smoothing model
C                       The same as the farfield boundary with a
C                       ramp increase in smoothing around the t.e.
C
C                  = 2  Standard internal smoothing model using
C                       reflected points.
C                       A standard internal smoothing using extrapolated
C                       information to define an imaginary line of points
C                       inside the wall.  In this case the smoothing is not
C                       increased in the t.e. region.
C
C         Constants
      IBCOND = 0
      JTESMTH = 5
      TECOEF  = 4.0
      IF(IBCOND.EQ.0) RETURN
      IF(IBCOND.EQ.2) GO TO 140
C
C         Type 1: 1D smoothing formulation
      DO 130 I=1,IPBBMX
CCC      IF(IPBB(3,I).NE.4) GO TO 130
C
C          Is point to be smoothed on this level?
      ICONT1 = 0
      IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C          If yes, calculate and add contributions form
C          both cells surrounding the cell
      IF(ICONT1.EQ.1) THEN
```

```
C
C           First cell:
          CALL CELPOINT(IPBB(1,I))
          CALL METRC4
          CALL CTIME
          AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Ramp smoothing near t.e.
          IF(I.LE.JTESMTH+1) THEN
            AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
          ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
            AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
     1             /FLOAT(JTESMTH))*AVIS
          END IF
C
          DO 124 K=1,4
          KP2 = K+2
          KP6 = K+6
            Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(Q(KP2,I1)-Q(KP2,I4))
  124     CONTINUE
C
C           Second cell:
          CALL CELPOINT(IPBB(2,I))
          CALL METRC4
          CALL CTIME
          AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Ramp soothing near t.e.
          IF(I.LE.JTESMTH+1) THEN
            AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
          ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
            AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
     1             /FLOAT(JTESMTH))*AVIS
          END IF
C
          DO 126 K=1,4
          KP2 = K+2
          KP6 = K+6
            Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(Q(KP2,I4)-Q(KP2,I1))
  126     CONTINUE
        END IF
  130 CONTINUE
C
        RETURN
C
C           Type 2:  Reflection wall smoothing
  140 CONTINUE
        DO 170 I=1,IPBBMX
        IP1 = I+1
        IM1 = I-1
        IF(IPBB(3,I).NE.4) GO TO 170
C
C           Is this point to be smoothed on this level?
        ICONT1 = 0
```

```
      IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C         If Yes, calculate and add contributions form both cells
      IF(ICONT1.EQ.1) THEN
C
C           Calculate surface tangent vector (dx,dy)
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
C
C           First cell:
        CALL CELPOINT(IPBB(1,I))
C
        DXDXI = 0.5*(QSAV(1,I)+Q(1,I4)-QSAV(1,IM1)-Q(1,I1))
        DYDXI = 0.5*(QSAV(2,I)+Q(2,I4)-QSAV(2,IM1)-Q(2,I1))
        DXDET = 0.5*(QSAV(1,IM1)+QSAV(1,I)-Q(1,I1)-Q(1,I4))
        DYDET = 0.5*(QSAV(2,IM1)+QSAV(2,I)-Q(2,I1)-Q(2,I4))
        DV = DXDXI*DYDET-DXDET*DYDXI
        DL = SQRT(DXDET*DXDET+DYDET*DYDET)
        DM = SQRT(DXDXI*DXDXI+DYDXI*DYDXI)
C
C         FIND MIN OF DX/(|U|+A) AND DY/(|V|+A)
        DO 150 K=1,4
        KP2 = K+2
        QAVE(K) = 0.25*(Q(KP2,I1)+QSAV(KP2,IM1)
     1                  +QSAV(KP2,I)+Q(KP2,I4))
  150   CONTINUE
        UTEMP = QAVE(2)/QAVE(1)
        VTEMP = QAVE(3)/QAVE(1)
        A2    = GAMMA*GM1*(QAVE(4)/QAVE(1)
```

```
     1                    -0.5*(UTEMP*UTEMP+VTEMP*VTEMP))
        A     = SQRT(A2)
C
        DTA = DV/(ABS(UTEMP*DYDET-VTEMP*DXDET)+A*DL)
        DTB = DV/(ABS(UTEMP*DYDXI-VTEMP*DXDXI)+A*DM)
        DT = CFL*MIN(DTA,DTB)
C
        AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Extrapolate r,p,h0 and reflect u,v
        P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
        P3 = GM1*(QSAV(6,I)-0.5*(QSAV(4,I)**2+QSAV(5,I)**2)/QSAV(3,I))
        PI4 = P3-2.*(P3-P4)
        H4 = (Q(6,I4)+P4)/Q(3,I4)
        H3 = (QSAV(6,I)+P3)/QSAV(3,I)
        HI4 = H3
        VELT3 = (QSAV(4,I)*DX+QSAV(5,I)*DY)/QSAV(3,I)
        VELN3 = (-QSAV(4,I)*DY+QSAV(5,I)*DX)/QSAV(3,I)
        UI4 = VELT3*DX+VELN3*DY
        VI4 = VELT3*DY-VELN3*DX
        RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
        EI4 = RI4*HI4-PI4
        P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
        P2 = GM1*(QSAV(6,IM1)-0.5*(QSAV(4,IM1)**2
     1                            +QSAV(5,IM1)**2)/QSAV(3,IM1))
        PI1 = P2-2.*(P2-P1)
        H1 = (Q(6,I1)+P1)/Q(3,I1)
        H2 = (QSAV(6,IM1)+P2)/QSAV(3,IM1)
        HI1 = H2
        VELT2 = (QSAV(4,IM1)*DX+QSAV(5,IM1)*DY)/QSAV(3,IM1)
        VELN2 = (-QSAV(4,IM1)*DY+QSAV(5,IM1)*DX)/QSAV(3,IM1)
        UI1 = VELT2*DX+VELN2*DY
        VI1 = VELT2*DY-VELN2*DX
        RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
        EI1 = RI1*HI1-PI1
C
C           Find reflected cell center values
        QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
        QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
        QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
        QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C           Add contribution
        DO 164 K=1,4
        KP2 = K+2
        KP6 = K+6
        QAVE1 = 0.25*(Q(KP2,I1)+QSAV(KP2,IM1)+QSAV(KP2,I)+Q(KP2,I4))
C
        Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I4))
  164   CONTINUE
C
C           Second Cell:
        CALL CELPOINT(IPBB(2,I))
C
```

```
      DXDXI = 0.5*(QSAV(1,IP1)+Q(1,I4)-QSAV(1,I)-Q(1,I1))
      DYDXI = 0.5*(QSAV(2,IP1)+Q(2,I4)-QSAV(2,I)-Q(2,I1))
      DXDET = 0.5*(QSAV(1,I)+QSAV(1,IP1)-Q(1,I1)-Q(1,I4))
      DYDET = 0.5*(QSAV(2,I)+QSAV(2,IP1)-Q(2,I1)-Q(2,I4))
      DV = DXDXI*DYDET-DXDET*DYDXI
      DL = SQRT(DXDET*DXDET+DYDET*DYDET)
      DM = SQRT(DXDXI*DXDXI+DYDXI*DYDXI)
C
C       FIND MIN OF DX/(|U|+A) AND DY/(|V|+A)
      DO 165 K=1,4
      KP2 = K+2
      QAVE(K) = 0.25*(Q(KP2,I1)+QSAV(KP2,I)+QSAV(KP2,IP1)+Q(KP2,I4))
  165 CONTINUE
      UTEMP = QAVE(2)/QAVE(1)
      VTEMP = QAVE(3)/QAVE(1)
      A2    = GAMMA*GM1*(QAVE(4)/QAVE(1)
     1                  -0.5*(UTEMP*UTEMP+VTEMP*VTEMP))
      A     = SQRT(A2)
C
      DTA = DV/(ABS(UTEMP*DYDET-VTEMP*DXDET)+A*DL)
      DTB = DV/(ABS(UTEMP*DYDXI-VTEMP*DXDXI)+A*DM)
      DT = CFL*MIN(DTA,DTB)
C
      AVIS = AVISCF*DT*(DL+DM)/DV
C
C         Extrapolate r,p,h0 and reflect u,v
      P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
      P3 = GM1*(QSAV(6,IP1)-0.5*(QSAV(4,IP1)**2
     1                          +QSAV(5,IP1)**2)/QSAV(3,IP1))
      PI4 = P3-2.*(P3-P4)
      H4 = (Q(6,I4)+P4)/Q(3,I4)
      H3 = (QSAV(6,IP1)+P3)/QSAV(3,IP1)
      HI4 = H3
      VELT3 = (QSAV(4,IP1)*DX+QSAV(5,IP1)*DY)/QSAV(3,IP1)
      VELN3 = (-QSAV(4,IP1)*DY+QSAV(5,IP1)*DX)/QSAV(3,IP1)
      UI4 = VELT3*DX+VELN3*DY
      VI4 = VELT3*DY-VELN3*DX
      RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
      EI4 = RI4*HI4-PI4
      P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
      P2 = GM1*(QSAV(6,I)-0.5*(QSAV(4,I)**2+QSAV(5,I)**2)/QSAV(3,I))
      PI1 = P2-2.*(P2-P1)
      H1 = (Q(6,I1)+P1)/Q(3,I1)
      H2 = (QSAV(6,I)+P2)/QSAV(3,I)
      HI1 = H2
      VELT2 = (QSAV(4,I)*DX+QSAV(5,I)*DY)/QSAV(3,I)
      VELN2 = (-QSAV(4,I)*DY+QSAV(5,I)*DX)/QSAV(3,I)
      UI1 = VELT2*DX+VELN2*DY
      VI1 = VELT2*DY-VELN2*DX
      RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
      EI1 = RI1*HI1-PI1
C
C         Find reflected cell center values
      QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
```

```
        QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
        QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
        QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C           Add contribution
        DO 166 K=1,4
        KP2 = K+2
        KP6 = K+6
        QAVE1 = 0.25*(Q(KP2,I1)+QSAV(KP2,I)+QSAV(KP2,IP1)+Q(KP2,I4))
C
        Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I1))
  166   CONTINUE
      END IF
  170 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C           SUBROUTINE: EULERWAL2
C-----------------------------------------------------------------
      SUBROUTINE EULERWAL2(LEV)
C
C         This subroutine preforms a stream line intergration
C         of the euler eqn. in natural coordinates to inforce
C         the solid wall no normal flow boundary condition.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'LUNITS.INC'
C
      DIMENSION PIP1(257),PEX(257),HOEX(257),
     1          UT(257),DUT(257),QAVE(4)
      DIMENSION QN1(4),QN2(4),QAVE2(4),QSAV(6,257)
C
C         Calculate pressures at ring of points just inside the
C         flow ajoining the boundary
      DO 10 I=1,IPBBMX
C
      J1 = ABS(IP(3,IPBB(1,I)))
C
      PIP1(I) = GM1*(Q(6,J1)-0.5*(Q(4,J1)**2+Q(5,J1)**2)/Q(3,J1))
   10 CONTINUE
C
C         Calculate extrapolated values at surface
      II   = 1
      IIP1 = 2
C
      DO 20 I=1,IPBBMX+1
      IF(I.LE.IPBBMX) THEN
        JP   = ABS(IP(4,IPBB(1,I)))
      ELSE
```

```
        JP   = ABS(IP(4,IPBB(1,1)))
      END IF
C
      II = IWP1(I)
      IF(II.LT.IPBBMX) THEN
        IIP1 = II+1
      ELSE
        IIP1 = 1
      END IF
      JN = ABS(IP(3,IPBB(1,II)))
      JNP1 = ABS(IP(3,IPBB(1,IIP1)))
C
      PINT = PIP1(II)+TSCL(I)*(PIP1(IIP1)-PIP1(II))
      TMP1 = (Q(6,JN)+PIP1(II))/Q(3,JN)
      TMP2 = (Q(6,JNP1)+PIP1(IIP1))/Q(3,JNP1)
      HOINT = TMP1+TSCL(I)*(TMP2-TMP1)
C
      DO 12 K=1,6
      QSAV(K,I) = Q(K,JN)+TSCL(I)*(Q(K,JNP1)-Q(K,JN))
   12 CONTINUE
C
C         Extrapolate to surface using normal momentum eqn
C         for pressure and zeroth order extrapolation for
C         total enthalpy
      PEX(I) = PINT
      HOEX(I) = HOINT
      IF(I.EQ.1.OR.I.EQ.IPBBMX+1) THEN
        DX = TX(1)-TX(IPBBMX+1)
        DY = TY(1)-TY(IPBBMX+1)
        TMP = SQRT(DX*DX+DY*DY)
        DX = DX/TMP
        DY = DY/TMP
      ELSE
        DX = TX(I)
        DY = TY(I)
      END IF
      UT(I) = (DX*Q(4,JP)+DY*Q(5,JP))/Q(3,JP)
      PEX(I) = PINT-Q(3,JP)*UT(I)*UT(I)*SSCL(I)/RC(I)
   20 CONTINUE
C
C         Correct t/e pressure to average of upper and lower
C         points just upstream of t/e and set tangent to bisector
      PEX(IPBBMX+1)  = 0.5*(PEX(1)+PEX(IPBBMX+1))
      HOEX(IPBBMX+1) = 0.5*(HOEX(1)+HOEX(IPBBMX+1))
      DX = TX(1)-TX(IPBBMX+1)
      DY = TY(1)-TY(IPBBMX+1)
      TMP = SQRT(DX*DX+DY*DY)
      DX = DX/TMP
      DY = DY/TMP
      JP = ABS(IP(4,IPBB(1,1)))
      UT(IPBBMX+1)   = (-DX*Q(4,JP)
     1                  -DY*Q(5,JP))/Q(3,JP)
      PEX(1)  = PEX(IPBBMX+1)
      HOEX(1) = HOEX(IPBBMX+1)
```

```
      UT(1)   = -UT(IPBBMX+1)
C
C         Solve streamline euler eqn.
C
C         Zero wall DU's
      DO 30 I=1,IPBBMX+1
      DUT(I) = 0.0
   30 CONTINUE
C
C        SWEEP CELL BY CELL AND CALCULATE DU'S
      DO 40 I=1,IPBBMX
      IP1 = I+1
      J1 = ABS(IP(1,IPBB(2,I)))
      J2 = ABS(IP(4,IPBB(2,I)))
      DS = SQRT((Q(1,J2)-Q(1,J1))**2+(Q(2,J2)-Q(2,J1))**2)
C
      DO 32 K=1,4
      KP2 = K+2
      QAVE(K) = 0.5*(Q(KP2,J1)+Q(KP2,J2))
   32 CONTINUE
C
      UTEMP = QAVE(2)/QAVE(1)
      VTEMP = QAVE(3)/QAVE(1)
      A2    = GAMMA*GM1*(QAVE(4)/QAVE(1)
     1                  -0.5*(UTEMP*UTEMP+VTEMP*VTEMP))
      A     = SQRT(A2)
C
      DT = CFL*DS/(SQRT(UTEMP**2+VTEMP**2)+A)
C
      CHNU = 0.5*((UT(I)**2-UT(IP1)**2)
     1    +(1./Q(3,J1)+1./Q(3,J2))*(PEX(I)-PEX(IP1)))*DT/DS
      CHNF = 0.5*(UT(I)+UT(IP1))*CHNU*DT/DS
C
      DUT(I)   = DUT(I)  +0.5*(CHNU-CHNF)
      DUT(IP1) = DUT(IP1)+0.5*(CHNU+CHNF)
   40 CONTINUE
C
      DUT(1) = DUT(1)-DUT(IPBBMX+1)
C
C         Calculate new conditions and correct DQ's
      DO 50 I=1,IPBBMX
CCC      IF(IPBB(3,I).NE.4) GOTO 50
      JP = ABS(IP(4,IPBB(1,I)))
      UT(I) = UT(I)+DUT(I)
C
      IF(I.EQ.1) THEN
        DX = TX(1)-TX(IPBBMX+1)
        DY = TY(1)-TY(IPBBMX+1)
        TMP = SQRT(DX*DX+DY*DY)
        DX = DX/TMP
        DY = DY/TMP
      ELSE
        DX = TX(I)
        DY = TY(I)
```

```
      END IF
      RNEW = PEX(I)/(GM1DG*(H0EX(I)-0.5*UT(I)*UT(I)))
      UNEW = DX*UT(I)
      VNEW = DY*UT(I)
      ENEW = RNEW*H0EX(I)-PEX(I)
C
      Q(7,JP)  = RNEW      -Q(3,JP)
      Q(8,JP)  = RNEW*UNEW -Q(4,JP)
      Q(9,JP)  = RNEW*VNEW -Q(5,JP)
      Q(10,JP) = ENEW      -Q(6,JP)
   50 CONTINUE
C
C         This section smooths the solid wall boundary points.
C         Points are always smoothed on the lowest level in
C         which the two adjoining cell to the boundary exist.
C         This is consistent with the internal point smoothing.
C             For the solid wall boundary two formulations are
C         possible:
C
C           IBCOND = 0  no smoothing applied, just return
C
C                  = 1  1-D tangent smoothing model
C                       The same as the farfield boundary with a
C                       ramp increase in smoothing around the t.e.
C
C                  = 2  Standard internal smoothing model using
C                       reflected points.
C                       A standard internal smoothing using extrapolated
C                       information to define an imaginary line of points
C                       inside the wall.  In this case the smoothing is not
C                       increased in the t.e. region.
C
C         Constants
      IBCOND = 0
      JTESMTH = 5
      TECOEF  = 4.0
      IF(IBCOND.EQ.0) RETURN
      IF(IBCOND.EQ.2) GO TO 140
C
C         Type 1: 1D smoothing formulation
      DO 130 I=1,IPBBMX
CCC      IF(IPBB(3,I).NE.4) GO TO 130
C
C         Is point to be smoothed on this level?
      ICONT1 = 0
      IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C         If yes, calculate and add contributions form
C         both cells surrounding the cell
```

```
      IF(ICONT1.EQ.1) THEN
C
C           First cell:
        CALL CELPOINT(IPBB(1,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Ramp smoothing near t.e.
        IF(I.LE.JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
        ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
     1             /FLOAT(JTESMTH))*AVIS
        END IF
C
        DO 124 K=1,4
        KP2 = K+2
        KP6 = K+6
          Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(Q(KP2,I1)-Q(KP2,I4))
  124   CONTINUE
C
C           Second cell:
        CALL CELPOINT(IPBB(2,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
C           Ramp soothing near t.e.
        IF(I.LE.JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
        ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
          AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
     1             /FLOAT(JTESMTH))*AVIS
        END IF
C
        DO 126 K=1,4
        KP2 = K+2
        KP6 = K+6
          Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(Q(KP2,I4)-Q(KP2,I1))
  126   CONTINUE
      END IF
  130 CONTINUE
C
      RETURN
C
C         Type 2:  Reflection wall smoothing
  140 CONTINUE
      DO 170 I=1,IPBBMX
      IP1 = I+1
      IM1 = I-1
      IF(IPBB(3,I).NE.4) GO TO 170
C
C         Is this point to be smoothed on this level?
```

```
      ICONT1 = 0
      IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C         If Yes, calculate and add contributions form both cells
      IF(ICONT1.EQ.1) THEN
C
C           Calculate surface tangent vector (dx,dy)
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
C
C           First cell:
        CALL CELPOINT(IPBB(1,I))
C
        DXDXI = 0.5*(QSAV(1,I)+Q(1,I4)-QSAV(1,IM1)-Q(1,I1))
        DYDXI = 0.5*(QSAV(2,I)+Q(2,I4)-QSAV(2,IM1)-Q(2,I1))
        DXDET = 0.5*(QSAV(1,IM1)+QSAV(1,I)-Q(1,I1)-Q(1,I4))
        DYDET = 0.5*(QSAV(2,IM1)+QSAV(2,I)-Q(2,I1)-Q(2,I4))
        DV = DXDXI*DYDET-DXDET*DYDXI
        DL = SQRT(DXDET*DXDET+DYDET*DYDET)
        DM = SQRT(DXDXI*DXDXI+DYDXI*DYDXI)
C
C         FIND MIN OF DX/(|U|+A) AND DY/(|V|+A)
        DO 150 K=1,4
        KP2 = K+2
        QAVE(K) = 0.25*(Q(KP2,I1)+QSAV(KP2,IM1)
     1                  +QSAV(KP2,I)+Q(KP2,I4))
  150   CONTINUE
        UTEMP = QAVE(2)/QAVE(1)
        VTEMP = QAVE(3)/QAVE(1)
```

```
      A2    = GAMMA*GM1*(QAVE(4)/QAVE(1)
     1                  -0.5*(UTEMP*UTEMP+VTEMP*VTEMP))
      A     = SQRT(A2)
C
      DTA = DV/(ABS(UTEMP*DYDET-VTEMP*DXDET)+A*DL)
      DTB = DV/(ABS(UTEMP*DYDXI-VTEMP*DXDXI)+A*DM)
      DT = CFL*MIN(DTA,DTB)
C
      AVIS = AVISCF*DT*(DL+DM)/DV
C
C         Extrapolate r,p,h0 and reflect u,v
      P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
      P3 = GM1*(QSAV(6,I)-0.5*(QSAV(4,I)**2+QSAV(5,I)**2)/QSAV(3,I))
      PI4 = P3-2.*(P3-P4)
      H4 = (Q(6,I4)+P4)/Q(3,I4)
      H3 = (QSAV(6,I)+P3)/QSAV(3,I)
      HI4 = H3
      VELT3 = (QSAV(4,I)*DX+QSAV(5,I)*DY)/QSAV(3,I)
      VELN3 = (-QSAV(4,I)*DY+QSAV(5,I)*DX)/QSAV(3,I)
      UI4 = VELT3*DX+VELN3*DY
      VI4 = VELT3*DY-VELN3*DX
      RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
      EI4 = RI4*HI4-PI4
      P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
      P2 = GM1*(QSAV(6,IM1)-0.5*(QSAV(4,IM1)**2
     1                          +QSAV(5,IM1)**2)/QSAV(3,IM1))
      PI1 = P2-2.*(P2-P1)
      H1 = (Q(6,I1)+P1)/Q(3,I1)
      H2 = (QSAV(6,IM1)+P2)/QSAV(3,IM1)
      HI1 = H2
      VELT2 = (QSAV(4,IM1)*DX+QSAV(5,IM1)*DY)/QSAV(3,IM1)
      VELN2 = (-QSAV(4,IM1)*DY+QSAV(5,IM1)*DX)/QSAV(3,IM1)
      UI1 = VELT2*DX+VELN2*DY
      VI1 = VELT2*DY-VELN2*DX
      RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
      EI1 = RI1*HI1-PI1
C
C         Find reflected cell center values
      QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
      QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
      QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
      QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C         Add contribution
      DO 164 K=1,4
      KP2 = K+2
      KP6 = K+6
      QAVE1 = 0.25*(Q(KP2,I1)+QSAV(KP2,IM1)+QSAV(KP2,I)+Q(KP2,I4))
C
      Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I4))
  164 CONTINUE
C
C         Second Cell:
      CALL CELPOINT(IPBB(2,I))
```

```
C
      DXDXI = 0.5*(QSAV(1,IP1)+Q(1,I4)-QSAV(1,I)-Q(1,I1))
      DYDXI = 0.5*(QSAV(2,IP1)+Q(2,I4)-QSAV(2,I)-Q(2,I1))
      DXDET = 0.5*(QSAV(1,I)+QSAV(1,IP1)-Q(1,I1)-Q(1,I4))
      DYDET = 0.5*(QSAV(2,I)+QSAV(2,IP1)-Q(2,I1)-Q(2,I4))
      DV = DXDXI*DYDET-DXDET*DYDXI
      DL = SQRT(DXDET*DXDET+DYDET*DYDET)
      DM = SQRT(DXDXI*DXDXI+DYDXI*DYDXI)
C
C       FIND MIN OF DX/(|U|+A) AND DY/(|V|+A)
      DO 165 K=1,4
      KP2 = K+2
      QAVE(K) = 0.25*(Q(KP2,I1)+QSAV(KP2,I)+QSAV(KP2,IP1)+Q(KP2,I4))
  165 CONTINUE
      UTEMP = QAVE(2)/QAVE(1)
      VTEMP = QAVE(3)/QAVE(1)
      A2    = GAMMA*GM1*(QAVE(4)/QAVE(1)
     1                -0.5*(UTEMP*UTEMP+VTEMP*VTEMP))
      A     = SQRT(A2)
C
      DTA = DV/(ABS(UTEMP*DYDET-VTEMP*DXDET)+A*DL)
      DTB = DV/(ABS(UTEMP*DYDXI-VTEMP*DXDXI)+A*DM)
      DT = CFL*MIN(DTA,DTB)
C
      AVIS = AVISCF*DT*(DL+DM)/DV
C
C         Extrapolate r,p,h0 and reflect u,v
      P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
      P3 = GM1*(QSAV(6,IP1)-0.5*(QSAV(4,IP1)**2
     1                          +QSAV(5,IP1)**2)/QSAV(3,IP1))
      PI4 = P3-2.*(P3-P4)
      H4 = (Q(6,I4)+P4)/Q(3,I4)
      H3 = (QSAV(6,IP1)+P3)/QSAV(3,IP1)
      HI4 = H3
      VELT3 = (QSAV(4,IP1)*DX+QSAV(5,IP1)*DY)/QSAV(3,IP1)
      VELN3 = (-QSAV(4,IP1)*DY+QSAV(5,IP1)*DX)/QSAV(3,IP1)
      UI4 = VELT3*DX+VELN3*DY
      VI4 = VELT3*DY-VELN3*DX
      RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
      EI4 = RI4*HI4-PI4
      P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
      P2 = GM1*(QSAV(6,I)-0.5*(QSAV(4,I)**2+QSAV(5,I)**2)/QSAV(3,I))
      PI1 = P2-2.*(P2-P1)
      H1 = (Q(6,I1)+P1)/Q(3,I1)
      H2 = (QSAV(6,I)+P2)/QSAV(3,I)
      HI1 = H2
      VELT2 = (QSAV(4,I)*DX+QSAV(5,I)*DY)/QSAV(3,I)
      VELN2 = (-QSAV(4,I)*DY+QSAV(5,I)*DX)/QSAV(3,I)
      UI1 = VELT2*DX+VELN2*DY
      VI1 = VELT2*DY-VELN2*DX
      RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
      EI1 = RI1*HI1-PI1
C
C         Find reflected cell center values
```

```
        QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
        QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
        QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
        QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C           Add contribution
        DO 166 K=1,4
        KP2 = K+2
        KP6 = K+6
        QAVE1 = 0.25*(Q(KP2,I1)+QSAV(KP2,I)+QSAV(KP2,IP1)+Q(KP2,I4))
C
        Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I1))
  166   CONTINUE
      END IF
  170 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: FARFDBC2
C-----------------------------------------------------------------
      SUBROUTINE FARFDBC2(CL,CD)
C
C         This subroutine calculates the far field boundary
C         conditions using a local characteristic analysis
C         tangent and normal to the boundary.  Both uniform
C         freestream or far field vortex boundary conditions
C         are possible.  The selection is made by setting the
C         following switch:
C           IFDTYPE = 0  for uniform freestreem conditions
C                     1  for vortex farfield conditions with
C                        the strength of the point vortex based
C                        on an integration of surface pressure
C                        to set the lift.
C
C         Note: for supersonic flows the uniform freestream
C               flow condition is automatically set since this
C               boundary condition is only correct for subsonic
C               flows.
C
C         Note: RAD for vortex farfield boundary assumes the vortex
C               is located at the quarter chord of the airfoil.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'LUNITS.INC'
      DIMENSION UBAR(4)
C
C         Constants
      IFDTYPE = 1
      PHI   = 3.141592654
C
C         Calculate Lift Force Coefficients through an
```

```
C         integration of the surface pressures of airfoil.
      IF(IFDTYPE.EQ.1) THEN
C
        CHORD = 0.0
        CFN  = 0.0
        CFT  = 0.0
C
        DO 5 I=1,IPBBMX
        J1 = ABS(IP(1,IPBB(2,I)))
        J2 = ABS(IP(4,IPBB(2,I)))
C
        IF(I.EQ.1) THEN
          TX1 = Q(1,J1)
          TY1 = Q(2,J1)
        END IF
        TCHORD = (TX1-Q(1,J2))**2+(TY1-Q(2,J2))**2
        IF(TCHORD.GT.CHORD) CHORD = TCHORD
C
        DX = Q(1,J2)-Q(1,J1)
        DY = Q(2,J2)-Q(2,J1)
        DS = SQRT(DX*DX+DY*DY)
C
        P1 = GM1*(Q(6,J1)
     1      -0.5*(Q(4,J1)*Q(4,J1)+Q(5,J1)*Q(5,J1))/Q(3,J1))
        P2 = GM1*(Q(6,J2)
     1      -0.5*(Q(4,J2)*Q(4,J2)+Q(5,J2)*Q(5,J2))/Q(3,J2))
        TMP = P1+P2
        CFN = CFN+TMP*DX
        CFT = CFT+TMP*DY
    5   CONTINUE
C
        CHORD = SQRT(CHORD)
        QFS = ROFS*(UFS*UFS+VFS*VFS)*CHORD
        CFN = -CFN/QFS
        CFT = CFT/QFS
C
        ALPHAR = 3.14159*ALPHA/180.0
        CL = CFN*COS(ALPHAR)-CFT*SIN(ALPHAR)
        CD = CFN*SIN(ALPHAR)+CFT*COS(ALPHAR)
C
C         SET AIRFOIL CENTER AT 1/4 CHORD
        XQC = TX1-0.75*CHORD
        YQC = TY1
      ELSE
        CL = 0.0
      END IF
C
C        Sweep around farfield boundary and correct DU's
C        using Characteristic analysis tangent and normal
C        to the boundary.
C
      DO 10 I=1,IPBUMX
      IF(IPBU(3,I).EQ.2) THEN
        IF(IPBU(1,I).NE.0) THEN
```

```
            J1 = ABS(IP(3,IPBU(1,I)))
            J2 = ABS(IP(4,IPBU(1,I)))
          ELSE
            J1 = ABS(IP(2,IPBU(2,I)))
            J2 = ABS(IP(1,IPBU(2,I)))
          END IF
        ELSE
          WRITE(LU1,*)' ERROR IN UDBC2C IPBU(3,I) NOT 1'
        END IF
C
C         Calculate boundary normal vector
C         Note: Present analysis assumes eta lines run
C               Normal to the far field boundary.
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        TMP3 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        DX = TMP1/TMP3
        DY = TMP2/TMP3
C
C         Calculate local radius and direction
        TMP1 = XQC-Q(1,J1)
        TMP2 = YQC-Q(2,J1)
        RAD  = SQRT(TMP1**2+TMP2**2)
        DRX  = TMP1/RAD
        DRY  = TMP2/RAD
C
C         Calculate extrapolated quantities from
C         the predicted values of Q at the boundary.
        REX   = Q(3,J1)+Q(7,J1)
        UEX   = (Q(4,J1)+Q(8,J1))/REX
        VEX   = (Q(5,J1)+Q(9,J1))/REX
        EEX   = Q(6,J1)+Q(10,J1)
        QSQEX =UEX*UEX+VEX*VEX
        PEX   =GM1*(EEX-0.5*REX*QSQEX)
        IF(PEX.LE.0.0) WRITE(LU1,*)'** PEX<0 AT UP I=',I
        AEX   =SQRT(GAMMA*PEX/REX)
C
        QNEX  = UEX*DX+VEX*DY
        QTEX  =-UEX*DY+VEX*DX
C
C         Set barred or frozen quantities of linearization
C         based on the extrapolated conditions.
        RBAR = REX
        ABAR = AEX
C
C         Calculate the free stream conditions without
C         the vortex.
        QNFS = UFS*DX+VFS*DY
        QTFS =-UFS*DY+VFS*DX
        QFS  = SQRT(QNFS**2+QTFS**2)
C
C         Set far field conditions to either free stream
C         or calculate and set to vortex farfield conditions
```

```
C
C         Set vortex farfield condition
      IF(IFDTYPE.EQ.1.AND.AMFS.LE.1) THEN
        COSFD = (UFS*DRX+VFS*DRY)/QFS
        SINFD = (-UFS*DRY+VFS*DRX)/QFS
        BETA  = SQRT(1.0-AMFS*AMFS)
        TMP1  = 1.0/(COSFD**2+BETA*BETA*SINFD*SINFD)
        DQVORT = QFS*CHORD*CL*BETA*TMP1/(4.0*PHI*RAD)
        QNFD  = QNFS+DQVORT*(-DRY*DX+DRX*DY)
        QTFD  = QTFS+DQVORT*(DRY*DY+DRX*DX)
        QFD   = SQRT(QNFD**2+QTFD**2)
        PFD   = (APFS**GM1DG+GM1D2G*ROFS*(QFS**2-QFD**2)
     1            /(APFS**(1.0/GAMMA)))**(GAMMA/GM1)
        ROFD  = ROFS*((PFD/APFS)**(1.0/GAMMA))
C
C           Otherwise set farfield conditions to freestream
      ELSE
        QNFD  = QNFS
        QTFD  = QTFS
        PFD   = APFS
        ROFD  = ROFS
      END IF
C
C         Calculate corrected farfield flow conditions
C         based on whether it is supersonic or subsonic
C         and inflow or outflow
C
C           Subsonic inflow
      IF(QNEX.GE.0.0.AND.QNEX.LE.ABAR) THEN
        PNEW = 0.5*(PFD+PEX+RBAR*ABAR*(QNFD-QNEX))
        QTNEW = QTFD
        QNNEW = QNFD+(PFD-PNEW)/(RBAR*ABAR)
        RNEW  = ROFD+(PNEW-PFD)/(ABAR*ABAR)
C
C           Subsonic outflow
C           note: sets the downstream characteristic
      ELSE IF(QNEX.GE.-ABAR.AND.QNEX.LT.0.0) THEN
        PNEW  = 0.5*(PFD+PEX+RBAR*ABAR*(QNFD-QNEX))
        QTNEW = QTEX
        QNNEW = QNEX+(PNEW-PEX)/(RBAR*ABAR)
        RNEW  = REX+(PNEW-PEX)/(ABAR*ABAR)
C
C           Supersonic inflow
      ELSE IF(QNEX.GT.ABAR) THEN
        PNEW  = PFD
        QTNEW = QTFD
        QNNEW = QNFD
        RNEW  = ROFD
C
C           Supersonic outflow
      ELSE IF(QNEX.LT.-ABAR) THEN
        PNEW  = PEX
        QTNEW = QTEX
        QNNEW = QNEX
```

```
        RNEW  = REX
      END IF
C
      ENEW = PNEW/GM1+0.5*RNEW*(QNNEW*QNNEW+QTNEW*QTNEW)
C
C         Calculate corrected DQ's
      Q(7,J1)  = RNEW-Q(3,J1)
      Q(8,J1)  = RNEW*(QNNEW*DX-QTNEW*DY)-Q(4,J1)
      Q(9,J1)  = RNEW*(QNNEW*DY+QTNEW*DX)-Q(5,J1)
      Q(10,J1) = ENEW-Q(6,J1)
C
   10 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: FINDFG
C-----------------------------------------------------------------
      SUBROUTINE FINDFG(I,M)
C
C         SUBROUTINE FINDFG CALCULATES F AND G AT POINT I
C         FROM U AND LEAVES THEM IN LOCATION M OF F AND G
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'SOLV.INC'
      INCLUDE 'GAM.INC'
C
      W1 = Q(4,I)*Q(4,I)/Q(3,I)
      W2 = Q(5,I)*Q(5,I)/Q(3,I)
      PTMP  = GM1*(Q(6,I)-0.5*(W1+W2))
      HTMP  = (Q(6,I)+PTMP)/Q(3,I)
C
      F(1,M) = Q(4,I)
      F(2,M) = W1+PTMP
      F(3,M) = Q(4,I)*Q(5,I)/Q(3,I)
      F(4,M) = Q(4,I)*HTMP
C
      G(1,M) = Q(5,I)
      G(2,M) = F(3,M)
      G(3,M) = W2+PTMP
      G(4,M) = Q(5,I)*HTMP
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: GAMMAS
C-----------------------------------------------------------------
      SUBROUTINE GAMMAS
C
C         This subroutine calculates constants containing gamma
C         for later use in other routines.
C
      INCLUDE 'GAM.INC'
C
```

```
      GAMMA = 1.4
      GM1   = GAMMA-1.0
      GM3   = GAMMA-3.0
      GM1D2 = GM1/2.0
      GM1DG = GM1/GAMMA
      GM1D2G= GM1D2/GAMMA
      GP1DG = (GAMMA+1.)/GAMMA
      GP1D2G= GP1DG/2.0
      HTOT  = 1.0/GM1
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: GEOWAL
C-----------------------------------------------------------------
      SUBROUTINE GEOWAL
C
C         This subroutine calculates the surface tangent, radius of
C         curvature, and scaling distances for calcuating extrapolated
C         flow values form the line of nodes next to the wall.
C           Note: this routine as written only applies to airfoils
C                 with solid wall pointers generated by geocreat
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'LUNITS.INC'
C
C         Calculate extrapolated values at surface
      II   = 1
      IIP1 = 2
C
      DO 20 I=1,IPBBMX+1
      IF(I.LE.IPBBMX) THEN
        JPM1 = ABS(IP(1,IPBB(1,I)))
        JP   = ABS(IP(4,IPBB(1,I)))
        JPP1 = ABS(IP(4,IPBB(2,I)))
      ELSE
        JPM1 = ABS(IP(1,IPBB(1,1)))
        JP   = ABS(IP(4,IPBB(1,1)))
        JPP1 = ABS(IP(4,IPBB(2,1)))
      END IF
C
C         calculate normal vector
      TMP1 = Q(1,JP)-Q(1,JPM1)
      TMP2 = Q(2,JP)-Q(2,JPM1)
      DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
      TX1 = TMP1/DS1
      TY1 = TMP2/DS1
      TMP1 = Q(1,JPP1)-Q(1,JP)
      TMP2 = Q(2,JPP1)-Q(2,JP)
      DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
```

```
      TX2 = TMP1/DS2
      TY2 = TMP2/DS2
C
      TMP1 = DS1+DS2
      TMP2 = DS2/(DS1*TMP1)
      TMP3 = (DS2-DS1)/(DS1*DS2)
      TMP4 = DS1/(DS2*TMP1)
C
      RC(I) = 0.5*TMP1/SQRT((TX2-TX1)**2+(TY2-TY1)**2)
C
      DXDS = -Q(1,JPM1)*TMP2+Q(1,JP)*TMP3+Q(1,JPP1)*TMP4
      DYDS = -Q(2,JPM1)*TMP2+Q(2,JP)*TMP3+Q(2,JPP1)*TMP4
      TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
      TX(I) = DXDS/TMP1
      TY(I) = DYDS/TMP1
      IF(I.EQ.1) THEN
        TX(I) = TX2
        TY(I) = TY2
        RC(I) = 1.E+20
      ELSE IF(I.EQ.IPBBMX+1) THEN
        TX(I) = TX1
        TY(I) = TY1
        RC(I) = 1.E+20
      END IF
C
      DNX = -TY(I)
      DNY =  TX(I)
C
C          Search for intersection of normal line and ring
      AP1 = Q(1,JP)
      AP2 = DNX
      BP1 = Q(2,JP)
      BP2 = DNY
C
   11 CONTINUE
      JN   = ABS(IP(3,IPBB(1,II)))
      JNP1 = ABS(IP(3,IPBB(1,IIP1)))
C
      A1 = Q(1,JN)
      A2 = Q(1,JNP1)-A1
      B1 = Q(2,JN)
      B2 = Q(2,JNP1)-B1
C
      DEL = AP2*B2-BP2*A2
      T = (AP2*(BP1-B1)-BP2*(AP1-A1))/DEL
C
      ITMP = 0
      IF(T.LT.0.0) THEN
        ITMP = 1
        IIP1 = II
        IF(II.GT.1) THEN
          II = II-1
        ELSE
```

```
          II = IPBBMX
        END IF
      ELSE IF(T.GT.1.0) THEN
        ITMP = 1
        II = IIP1
        IF(IIP1.LT.IPBBMX) THEN
          IIP1 = IIP1+1
        ELSE
          IIP1 = 1
        END IF
      END IF
      IF(ITMP.EQ.1) GO TO 11
C
      IWP1(I) = II
      TSCL(I) = T
      SSCL(I) = (A2*(BP1-B1)-B2*(AP1-A1))/DEL
C
   20 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: GTIME
C-----------------------------------------------------------------
      SUBROUTINE GTIME(LEV)
C
C         This subroutine calculates the maximum stable global
C         time step over the current level.  This is done by
C         calling CTIME for each cell ( which determines the
C         cell time step based on local flow properties and
C         the CFL No.) and saves the minimum value.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'INPT.INC'
      DIMENSION QAVE(4)
C
C         Find MIN of DX/(|U|+A) and DY/(|V|+A) for each cell
      DO 1 I = LEVP(1,LEV),LEVP(2,LEV)
C
C         Set local cell pointers
      I1  = ABS(IP(1,I))
      I2  = ABS(IP(2,I))
      I3  = ABS(IP(3,I))
      I4  = ABS(IP(4,I))
      INC = IP(5,I)
C
C          Find cell time step
      CALL METRC4
      CALL CTIME
C
C          Compare with current minimum
```

```
      IF(I.NE.LEVP(1,LEV)) THEN
        DTMIN = MIN(DTMIN,DT)
      ELSE
        DTMIN = DT
      END IF
    1 CONTINUE
C
C          Set final value of time step
      DT    = DTMIN
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: INBC4
C-----------------------------------------------------------------
      SUBROUTINE INBC4(LEV)
C
C         THIS SUBROUTINE CALCULATES THE SUBDOMAIN-GLOBAL
C         INTERFACE BOUNDARY CONDITIONS FROM THE GLOBAL
C         LEVEL SOLUTION.
C
      INCLUDE 'MAIN.INC'
C
C          INTERPOLATE INTERFACE BOUNDARY POINTS
      IF(IPBIMX(1,LEV).EQ.0) RETURN
      DO 5 I=IPBIMX(1,LEV),IPBIMX(2,LEV)
      J1 = IPBI(1,I)
      J2 = IPBI(2,I)
      J3 = IPBI(3,I)
      DO 5 K=3,6
    5 Q(K,J2) = 0.5*(Q(K,J1)+Q(K,J3))
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: INFACBC
C-----------------------------------------------------------------
      SUBROUTINE INFACBC(LEV)
C
C         THIE SUBROUTINE CORRECTS THE INTERFACE
C         BOUNDARY DU'S.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'SOLV.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'POINT.INC'
      DIMENSION DELUSAV(4)
C
      IF(IPBIMX(1,LEV-1).EQ.0) RETURN
C
C        NEW CORECTION SWEEP
      DO 55 ICEL=LEVP(1,LEV),LEVP(2,LEV)
      IF(IP(5,ICEL).EQ.0) GOTO 55
```

```
      ITYPE1 = 0
      ITYPE2 = 0
      ITYPE3 = 0
      ITYPE4 = 0
      NOCELL = 0
C
C        FIND TO CELLS
      DO 15 I = IPBIMX(1,LEV-1),IPBIMX(2,LEV-1)
      J2 = IPBI(2,I)
      IF(J2.EQ.IP(6,ICEL)) THEN
        ITYPE1 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(7,ICEL)) THEN
        ITYPE2 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(8,ICEL)) THEN
        ITYPE3 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(9,ICEL)) THEN
        ITYPE4 = 1
        NOCELL = 1
      END IF
   15 CONTINUE
      IF(NOCELL.EQ.0) GOTO 55
C
C         CALCULATE DV AND DT FOR TOTAL CELL
      CALL CELPOINT(ICEL)
      CALL METRC4
      CALL CTIME
      CALL DELTU
C
      DO 17 K=1,4
   17 DELUSAV(K) = -DELU(K)
C
C         FLUX BALANCE ON SUBCELL 1
      I1 = ABS(IP(1,ICEL))
      I2 = IP(6,ICEL)
      I3 = IP(5,ICEL)
      I4 = IP(9,ICEL)
      INC = 0
      CALL DELTU
C
      DO 18 K=1,4
   18 DELUSAV(K) = DELUSAV(K)+DELU(K)
C
C         FLUX BALANCE IN SUBSCELL 2
      I1 = IP(6,ICEL)
      I2 = ABS(IP(2,ICEL))
      I3 = IP(7,ICEL)
      I4 = IP(5,ICEL)
      INC = 0
      CALL DELTU
C
      DO 19 K=1,4
```

```
   19 DELUSAV(K) = DELUSAV(K)+DELU(K)
C
C         FLUX BALANCE IN SUBSCELL 3
      I1 = IP(5,ICEL)
      I2 = IP(7,ICEL)
      I3 = ABS(IP(3,ICEL))
      I4 = IP(8,ICEL)
      INC = 0
      CALL DELTU
C
      DO 20 K=1,4
   20 DELUSAV(K) = DELUSAV(K)+DELU(K)
C
C         FLUX BALANCE IN SUBSCELL 4
      I1 = IP(9,ICEL)
      I2 = IP(5,ICEL)
      I3 = IP(8,ICEL)
      I4 = ABS(IP(4,ICEL))
      INC = 0
      CALL DELTU
C
      DO 21 K=1,4
   21 DELU(K) = DELUSAV(K)+DELU(K)
C
C         CALCULATE DELF AND DELG
      CALL CELPOINT(ICEL)
      CALL METRC4
      INC = 0
      CALL DELTFG
C
C         DISTRIBUTE DELTA'S
      DO 50 K=1,4
      KP6 = K+6
      IF(ITYPE4.EQ.1.OR.ITYPE1.EQ.1)
     1  Q(KP6,I1) = Q(KP6,I1)+(DELU(K)-DELF(K)-DELG(K))/4.0
      IF(ITYPE1.EQ.1.OR.ITYPE2.EQ.1)
     1  Q(KP6,I2) = Q(KP6,I2)+(DELU(K)-DELF(K)+DELG(K))/4.0
      IF(ITYPE2.EQ.1.OR.ITYPE3.EQ.1)
     1  Q(KP6,I3) = Q(KP6,I3)+(DELU(K)+DELF(K)+DELG(K))/4.0
      IF(ITYPE3.EQ.1.OR.ITYPE4.EQ.1)
     1  Q(KP6,I4) = Q(KP6,I4)+(DELU(K)+DELF(K)-DELG(K))/4.0
   50 CONTINUE
C
   55 CONTINUE
C
      RETURN
      END
C-------------------------------------------------------------
C          SUBROUTINE: INFACBC2
C-------------------------------------------------------------
      SUBROUTINE INFACBC2(LEV)
C
C         THIS SUBROUTINE CALCULATES THE COARSE GRID DU'S
C         FOR THE FINE MESH INTERFACE AND CORRECTS THE INTERFACE
```

```
C         BOUNDARY DU'S ON THE COARSE MESH.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'SOLV.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'POINT.INC'
      DIMENSION DELUSAV(4),QI(4,2,257)
C
      IF(IPBIMX(1,LEV).EQ.0) GOTO 100
C
C        ZERO INTERFACE DU'S
      DO 5 I=IPBIMX(1,LEV),IPBIMX(2,LEV)
      J1 = IPBI(1,I)
      J2 = IPBI(2,I)
      J3 = IPBI(3,I)
      DO 5 K=7,10
      Q(K,J1) = 0.0
      Q(K,J2) = 0.0
      Q(K,J3) = 0.0
    5 CONTINUE
C
C        NEW CORECTION SWEEP
      DO 55 ICEL=LEVP(1,LEV+1),LEVP(2,LEV+1)
      IF(IP(5,ICEL).EQ.0) GOTO 55
      ITYPE1 = 0
      ITYPE2 = 0
      ITYPE3 = 0
      ITYPE4 = 0
      NOCELL = 0
C
C        FIND TO CELLS
      DO 15 I = IPBIMX(1,LEV),IPBIMX(2,LEV)
      J2 = IPBI(2,I)
      IF(J2.EQ.IP(6,ICEL)) THEN
        ITYPE1 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(7,ICEL)) THEN
        ITYPE2 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(8,ICEL)) THEN
        ITYPE3 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(9,ICEL)) THEN
        ITYPE4 = 1
        NOCELL = 1
      END IF
   15 CONTINUE
      IF(NOCELL.EQ.0) GOTO 55
C
C         CALCULATE DV AND DT FOR TOTAL CELL
      CALL CELPOINT(ICEL)
      CALL METRC4
      CALL CTIME
```

```
C
C         FLUX BALANCE ON SUBCELL 1
      I1 = ABS(IP(1,ICEL))
      I2 = IP(6,ICEL)
      I3 = IP(5,ICEL)
      I4 = IP(9,ICEL)
      INC = 0
      CALL DELTU
C
      DO 18 K=1,4
   18 DELUSAV(K) = DELU(K)
C
C         FLUX BALANCE IN SUBSCELL 2
      I1 = IP(6,ICEL)
      I2 = ABS(IP(2,ICEL))
      I3 = IP(7,ICEL)
      I4 = IP(5,ICEL)
      INC = 0
      CALL DELTU
C
      DO 19 K=1,4
   19 DELUSAV(K) = DELUSAV(K)+DELU(K)
C
C         FLUX BALANCE IN SUBSCELL 3
      I1 = IP(5,ICEL)
      I2 = IP(7,ICEL)
      I3 = ABS(IP(3,ICEL))
      I4 = IP(8,ICEL)
      INC = 0
      CALL DELTU
C
      DO 20 K=1,4
   20 DELUSAV(K) = DELUSAV(K)+DELU(K)
C
C         FLUX BALANCE IN SUBSCELL 4
      I1 = IP(9,ICEL)
      I2 = IP(5,ICEL)
      I3 = IP(8,ICEL)
      I4 = ABS(IP(4,ICEL))
      INC = 0
      CALL DELTU
C
      DO 21 K=1,4
   21 DELU(K) = DELUSAV(K)+DELU(K)
C
C         CALCULATE DELF AND DELG
      CALL CELPOINT(ICEL)
      CALL METRC4
      INC = 0
      CALL DELTFG
C
C         DISTRIBUTE DELTA'S
      DO 50 K=1,4
      KP6 = K+6
```

```
      IF(ITYPE4.EQ.1.OR.ITYPE1.EQ.1)
     1  Q(KP6,I1) = Q(KP6,I1)+(DELU(K)-DELF(K)-DELG(K))/4.0
      IF(ITYPE1.EQ.1.OR.ITYPE2.EQ.1)
     1  Q(KP6,I2) = Q(KP6,I2)+(DELU(K)-DELF(K)+DELG(K))/4.0
      IF(ITYPE2.EQ.1.OR.ITYPE3.EQ.1)
     1  Q(KP6,I3) = Q(KP6,I3)+(DELU(K)+DELF(K)+DELG(K))/4.0
      IF(ITYPE3.EQ.1.OR.ITYPE4.EQ.1)
     1  Q(KP6,I4) = Q(KP6,I4)+(DELU(K)+DELF(K)-DELG(K))/4.0
   50 CONTINUE
C
   55 CONTINUE
C
C         STORE NEW DU'S IN QI AND ZERO DU'S
      DO 60 I=IPBIMX(1,LEV),IPBIMX(2,LEV)
      J1 = IPBI(1,I)
      J2 = IPBI(3,I)
      DO 60 K=1,4
      KP6 = K+6
      QI(K,1,I) = Q(KP6,J1)
      QI(K,2,I) = Q(KP6,J2)
      Q(KP6,J1) = 0.0
      Q(KP6,J2) = 0.0
   60 CONTINUE
C
C         CORRECTION OF COURSE GRID DU'S
  100 IF(LEV.EQ.1) RETURN
      IF(IPBIMX(1,LEV-1).EQ.0) RETURN
C
C        NEW CORECTION SWEEP
      DO 155 ICEL=LEVP(1,LEV),LEVP(2,LEV)
      IF(IP(5,ICEL).EQ.0) GOTO 155
      ITYPE1 = 0
      ITYPE2 = 0
      ITYPE3 = 0
      ITYPE4 = 0
      NOCELL = 0
C
C        FIND TO CELLS
      DO 115 I = IPBIMX(1,LEV-1),IPBIMX(2,LEV-1)
      J2 = IPBI(2,I)
      IF(J2.EQ.IP(6,ICEL)) THEN
        ITYPE1 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(7,ICEL)) THEN
        ITYPE2 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(8,ICEL)) THEN
        ITYPE3 = 1
        NOCELL = 1
      ELSE IF(J2.EQ.IP(9,ICEL)) THEN
        ITYPE4 = 1
        NOCELL = 1
      END IF
  115 CONTINUE
```

```
      IF(NOCELL.EQ.0) GOTO 155
C
C         CALCULATE DV AND DT FOR TOTAL CELL
      CALL CELPOINT(ICEL)
      CALL METRC4
      CALL CTIME
      CALL DELTU
C
      DO 117 K=1,4
  117 DELU(K) = -DELU(K)
C
      CALL DELTFG
C
C         DISTRIBUTE DELTA'S
      DO 150 K=1,4
      KP6 = K+6
      IF(ITYPE4.EQ.1.OR.ITYPE1.EQ.1)
     1  Q(KP6,I1) = Q(KP6,I1)+(DELU(K)-DELF(K)-DELG(K))/4.0
      IF(ITYPE1.EQ.1.OR.ITYPE2.EQ.1)
     1  Q(KP6,I2) = Q(KP6,I2)+(DELU(K)-DELF(K)+DELG(K))/4.0
      IF(ITYPE2.EQ.1.OR.ITYPE3.EQ.1)
     1  Q(KP6,I3) = Q(KP6,I3)+(DELU(K)+DELF(K)+DELG(K))/4.0
      IF(ITYPE3.EQ.1.OR.ITYPE4.EQ.1)
     1  Q(KP6,I4) = Q(KP6,I4)+(DELU(K)+DELF(K)-DELG(K))/4.0
  150 CONTINUE
C
  155 CONTINUE
C
      DO 160 I=IPBIMX(1,LEV-1),IPBIMX(2,LEV-1)
      J1 = IPBI(1,I)
      J2 = IPBI(3,I)
      DO 160 K=1,4
      KP6 = K+6
      Q(KP6,J1) = Q(KP6,J1)+QI(K,1,I)
      Q(KP6,J2) = Q(KP6,J2)+QI(K,2,I)
      QI(K,1,I) = 0.0
      QI(K,2,I) = 0.0
  160 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: INITIA
C-----------------------------------------------------------------
      SUBROUTINE INITIA
C
C         This subroutine calculates the freestream quantities
C         and if ISTART=1 initializes the flow field to uniform
C         flow based on ALPHA and AMFS using isentropic relations.
C         If uniform flow is set the solid wall points are corrected
C         for a zero flux through the boundary by holding the pressure
C         and energy constant and rotating the velocity vector to
C         the local wall tangent.   Note if ISTART=0 the flow is
C         left as read in the INPUT subroutine.
```

```
C
C           IF ISTART = 0   THEN UNIFORM FLOW
C                       1   THEN RESTART
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'LUNITS.INC'
      DIMENSION FSU(4)
C
C         Calculate surface tangent, radius of curvature and
C         extrapolation scalings
      CALL GEOWAL
C
C         CALCULATE FREE STREAM VECTOR U
      ALPHAR = ALPHA*3.14159/180.0
      TMP  = 1.0+GM1D2*AMFS*AMFS
      ROFS = TMP**(-1./GM1)
      APFS = (TMP**(-1./GM1DG))/GAMMA
      UFS  = AMFS*COS(ALPHAR)/SQRT(TMP)
      VFS  = AMFS*SIN(ALPHAR)/SQRT(TMP)
      AFS  = 1.0/SQRT(TMP)
C
      FSU(1) = ROFS
      FSU(2) = ROFS*UFS
      FSU(3) = ROFS*VFS
      FSU(4) = APFS/GM1+ROFS*(UFS*UFS+VFS*VFS)/2.0
C
C         RETURN IF RESTART
      IF(ISTART.EQ.1) RETURN
C
C         INITIALIZE FLOW FIELD TO FREE STREAM
      DO 1  I = 1,IQMAX
      DO 1  K = 1,4
    1 Q(K+2,I) = FSU(K)
C
C         CORRECT WALL PROPERTIES
      DO 2 I=1,IPBBMX
C
C         SET POINTERS & CALCULATE WALL TANGENT
      IF(IPBB(3,I).EQ.4) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
```

```
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
      ELSE IF(IPBB(3,I).EQ.5) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        THETA1 = ATAN2((Q(2,J2)-Q(2,J1)),(Q(1,J2)-Q(1,J1)))
        THETA2 = ATAN2((Q(2,J2)-Q(2,J3)),(Q(1,J2)-Q(1,J3)))
        THETA  = 0.5*(THETA1+THETA2)
C
        DX = COS(THETA)
        DY = SIN(THETA)
      ELSE
        WRITE(LU1,*)' ERROR INITIA IPBB(3,I) NOT 4'
      END IF
C
      QFS = SQRT(UFS*UFS+VFS*VFS)
      SIGN = (UFS*DX+VFS*DY)
      SIGN = SIGN/ABS(SIGN)
      TU = SIGN*QFS*DX
      TV = SIGN*QFS*DY
      TR = APFS/(GM1DG*(HTOT-0.5*(TU*TU+TV*TV)))
      TE = TR*HTOT-APFS
C
C         IF EULER CALCULATION (INSSWT=0) MAKE FLOW TANGENT
      IF(INSSWT.EQ.0) THEN
CC        Q(3,J2)   = TR
CC        Q(4,J2) = TR*TU
CC        Q(5,J2) = TR*TV
CC        Q(6,J2) = TE
C
C         IF NAVIER-STOKES CALCULATION (INSSWT=1) SET ZERO FLOW
      ELSE IF(INSSWT.EQ.1) THEN
        Q(3,J2) = GAMMA*APFS/(AFS**2)
        Q(4,J2) = 0.0
        Q(5,J2) = 0.0
        Q(6,J2) = Q(3,J2)*HTOT-APFS
      END IF
    2 CONTINUE
C
C         OUTPUT OF INITIAL FLOW
      WRITE(LU1,1000)
      WRITE(1,1004) ROFS,UFS,VFS,APFS
      IF (IPRNT2.EQ.0) RETURN
      WRITE(LU1,*) ' INITIAL Q VALUES'
      DO 50 K=1,6
```

```
   50 WRITE(LU1,1001) (Q(K,I), I=1,IQMAX)
C
 1000 FORMAT(///,10X,'INITIAL FLOW FIELD  U1/U2/U3/U4',/)
 1001 FORMAT(1X,(10E12.4))
 1004 FORMAT(1X,'ROFS,UFS,VFS,APFS=',4E12.4,/)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: INJECT
C-----------------------------------------------------------------
      SUBROUTINE INJECT(LEV,INJTYPE)
C
C         THIS SUBROUTINE INJECTS FINE MESH DU'S INTO
C         THE COARSE GRID USING A WEIGHTED DISTRIBUTION
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'MET.INC'
C
      IF(INJTYPE.EQ.0) RETURN
C
C         FORM TYPE 1: SIMPLE ALGEBRAIC WEIGHTING
C                 |1  2  1|
C            1/16*|2  4  2|
C                 |1  2  1|
      IF(INJTYPE.NE.1) GO TO 100
      TMP = 1./16.
      DO 50 I=LEVP(1,LEV),LEVP(2,LEV)
C
C         CHECK FOR FINER GRID
      IF(IP(5,I).EQ.0) GO TO 50
C
      CALL CELPOINT(I)
C
      DO 40 K=7,10
      Q(K,INC) = TMP*(2.*(2.*Q(K,INC)
     1                   +Q(K,IN1)+Q(K,IN2)+Q(K,IN3)+Q(K,IN4))
     2                +Q(K,I1)+Q(K,I2)+Q(K,I3)+Q(K,I4))
   40 CONTINUE
C
   50 CONTINUE
      RETURN
C
C         FORM TYPE 2: VOLUME WEIGHTING
  100 IF(INJTYPE.NE.2) RETURN
      DO 150 I=LEVP(1,LEV),LEVP(2,LEV)
C
C         CHECK FOR FINER GRID
      IF(IP(5,I).EQ.0) GO TO 150
C
      CALL CELPOINT(I)
C
```

```
C         CALCULATE VOLUMES
      VA = -0.5*((Q(1,INC)-Q(1,I1))*(Q(2,IN4)-Q(2,IN1))
     1           -(Q(1,IN4)-Q(1,IN1))*(Q(2,INC)-Q(2,I1)))
      VB = -0.5*((Q(1,IN2)-Q(1,IN1))*(Q(2,INC)-Q(2,I2))
     1           -(Q(1,INC)-Q(1,I2))*(Q(2,IN2)-Q(2,IN1)))
      VC = -0.5*((Q(1,I3)-Q(1,INC))*(Q(2,IN3)-Q(2,IN2))
     1           -(Q(1,IN3)-Q(1,IN2))*(Q(2,I3)-Q(2,INC)))
      VD = -0.5*((Q(1,IN3)-Q(1,IN4))*(Q(2,I4)-Q(2,INC))
     1           -(Q(1,I4)-Q(1,INC))*(Q(2,IN3)-Q(2,IN4)))
      VT = (VA+VB+VC+VD)
C
      DO 140 K=7,10
      Q(K,INC) = 0.25*(Q(K,INC)
     1          +( Q(K,I1)*VA+Q(K,I2)*VB+Q(K,I3)*VC+Q(K,I4)*VD
     2            +Q(K,IN1)*(VA+VB)+Q(K,IN2)*(VB+VC)
     3            +Q(K,IN3)*(VC+VD)+Q(K,IN4)*(VD+VA) )/VT)
  140 CONTINUE
C
  150 CONTINUE

C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: INJECT5
C-----------------------------------------------------------------
      SUBROUTINE INJECT5(LEV,INJTYPE)
C
C         THIS SUBROUTINE INJECTS FINE MESH DU'S INTO
C         THE COARSE GRID USING A WEIGHTED DISTRIBUTION
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'SOLV.INC'
C
      DIMENSION QSAVE(4)
      DIMENSION QAVE(4),DF(4),DG(4)
C
      IF(INJTYPE.EQ.0) RETURN
C
C         FORM TYPE 1: SIMPLE ALGEBRAIC WEIGHTING
C                 |1  2  1|
C            1/16*|2  4  2|
C                 |1  2  1|
      IF(INJTYPE.NE.1) GO TO 100
      TMP = 1./16.
      DO 50 I=LEVP(1,LEV),LEVP(2,LEV)
C
C         CHECK FOR FINER GRID
      IF(IP(5,I).EQ.0) GO TO 50
C
      CALL CELPOINT(I)
```

```
C
      DO 40 K=7,10
      Q(K,INC) = TMP*(2.*(2.*Q(K,INC)
     1                    +Q(K,IN1)+Q(K,IN2)+Q(K,IN3)+Q(K,IN4))
     2                +Q(K,I1)+Q(K,I2)+Q(K,I3)+Q(K,I4))
   40 CONTINUE
C
   50 CONTINUE
      RETURN
C
C         FORM TYPE 2: VOLUME WEIGHTING
  100 IF(INJTYPE.NE.2) GO TO 200
      DO 150 I=LEVP(1,LEV),LEVP(2,LEV)
C
C         CHECK FOR FINER GRID
      IF(IP(5,I).EQ.0) GO TO 150
C
      CALL CELPOINT(I)
C
C         CALCULATE VOLUMES
      VA = -0.5*((Q(1,INC)-Q(1,I1))*(Q(2,IN4)-Q(2,IN1))
     1          -(Q(1,IN4)-Q(1,IN1))*(Q(2,INC)-Q(2,I1)))
      VB = -0.5*((Q(1,IN2)-Q(1,IN1))*(Q(2,INC)-Q(2,I2))
     1          -(Q(1,INC)-Q(1,I2))*(Q(2,IN2)-Q(2,IN1)))
      VC = -0.5*((Q(1,I3)-Q(1,INC))*(Q(2,IN3)-Q(2,IN2))
     1          -(Q(1,IN3)-Q(1,IN2))*(Q(2,I3)-Q(2,INC)))
      VD = -0.5*((Q(1,IN3)-Q(1,IN4))*(Q(2,I4)-Q(2,INC))
     1          -(Q(1,I4)-Q(1,INC))*(Q(2,IN3)-Q(2,IN4)))
      VT = (VA+VB+VC+VD)
C
      DO 140 K=7,10
      Q(K,INC) = 0.25*(Q(K,INC)
     1          +( Q(K,I1)*VA+Q(K,I2)*VB+Q(K,I3)*VC+Q(K,I4)*VD
     2            +Q(K,IN1)*(VA+VB)+Q(K,IN2)*(VB+VC)
     3            +Q(K,IN3)*(VC+VD)+Q(K,IN4)*(VD+VA) )/VT)
  140 CONTINUE
C
  150 CONTINUE
C
      RETURN
C
C         FORM TYPE 3: DONE AT THIS TIME
  200 GO TO 300
C
C         TYPE 4: DISTRIBUTION OF DU'S
C         SET TYPE OF CELL AVERAGING BEFORE DISTRIBUTION
C         BASED ON THE FOLLOWING SWITCH,
C           IT4SWT = 1  AVERAGE OF NODES 1-4
C                    2  AVERAGE OF NODES 1+3
C                    3  AVERAGE OF NODES 2+4
C                    4  AVERAGE OF NODES 2+3+4
  300 IF(INJTYPE.NE.4) GO TO 400
      IT4SWT = 2
C
```

```
      DO 330 I = LEVP(1,LEV),LEVP(2,LEV)
C
      CALL CELPOINT(I)
C
C         CALCULATE VOLUMES
      VA = -0.5*((Q(1,INC)-Q(1,I1))*(Q(2,IN4)-Q(2,IN1))
     1          -(Q(1,IN4)-Q(1,IN1))*(Q(2,INC)-Q(2,I1)))
      VB = -0.5*((Q(1,IN2)-Q(1,IN1))*(Q(2,INC)-Q(2,I2))
     1          -(Q(1,INC)-Q(1,I2))*(Q(2,IN2)-Q(2,IN1)))
      VC = -0.5*((Q(1,I3)-Q(1,INC))*(Q(2,IN3)-Q(2,IN2))
     1          -(Q(1,IN3)-Q(1,IN2))*(Q(2,I3)-Q(2,INC)))
      VD = -0.5*((Q(1,IN3)-Q(1,IN4))*(Q(2,I4)-Q(2,INC))
     1          -(Q(1,I4)-Q(1,INC))*(Q(2,IN3)-Q(2,IN4)))
      VT = (VA+VB+VC+VD)
C
C         CELL 1
      I1 = ABS(IP(1,I))
      I2 = IP(6,I)
      I3 = IP(5,I)
      I4 = IP(9,I)
      INC = 0
C
      CALL METRC4
      CALL CTIME
C
      DO 305 K=1,4
      KP6 = K+6
      IF(IT4SWT.EQ.1) THEN
        DELU(K) = 0.25*(Q(KP6,I1)+Q(KP6,I2)+Q(KP6,I3)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.2) THEN
        DELU(K) = 0.5*(Q(KP6,I1)+Q(KP6,I3))
      ELSE IF(IT4SWT.EQ.3) THEN
        DELU(K) = 0.5*(Q(KP6,I2)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.4) THEN
        DELU(K) = (Q(KP6,I1)+Q(KP6,I2)+Q(KP6,I4))/3.
      END IF
  305 CONTINUE
C
      CALL DELTFG
C
      DO 307 K=1,4
  307 QSAVE(K) = (DELU(K)+DELF(K)+DELG(K))*DV/VT
C
C         CELL 2
      I1 = I2
      I4 = I3
      I2 = ABS(IP(2,I))
      I3 = IP(7,I)
      INC = 0
C
      CALL METRC4
      CALL CTIME
C
      DO 310 K=1,4
```

```
      KP6 = K+6
      IF(IT4SWT.EQ.1) THEN
        DELU(K) = 0.25*(Q(KP6,I1)+Q(KP6,I2)+Q(KP6,I3)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.2) THEN
        DELU(K) = 0.5*(Q(KP6,I2)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.3) THEN
        DELU(K) = 0.5*(Q(KP6,I1)+Q(KP6,I3))
      ELSE IF(IT4SWT.EQ.4) THEN
        DELU(K) = (Q(KP6,I1)+Q(KP6,I2)+Q(KP6,I3))/3.
      END IF
  310 CONTINUE
C
      CALL DELTFG
C
      DO 312 K=1,4
  312 QSAVE(K) = QSAVE(K)+(DELU(K)+DELF(K)-DELG(K))*DV/VT
C
C         CELL 3
      I1 = I4
      I2 = I3
      I3 = ABS(IP(3,I))
      I4 = IP(8,I)
      INC = 0
C
      CALL METRC4
      CALL CTIME
C
      DO 315 K=1,4
      KP6 = K+6
      IF(IT4SWT.EQ.1) THEN
        DELU(K) = 0.25*(Q(KP6,I1)+Q(KP6,I2)+Q(KP6,I3)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.2) THEN
        DELU(K) = 0.5*(Q(KP6,I1)+Q(KP6,I3))
      ELSE IF(IT4SWT.EQ.3) THEN
        DELU(K) = 0.5*(Q(KP6,I2)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.4) THEN
        DELU(K) = (Q(KP6,I2)+Q(KP6,I3)+Q(KP6,I4))/3.
      END IF
  315 CONTINUE
C
      CALL DELTFG
C
      DO 317 K=1,4
  317 QSAVE(K) = QSAVE(K)+(DELU(K)-DELF(K)-DELG(K))*DV/VT
C
C
C         CELL 4
      I2 = I1
      I3 = I4
      I1 = IP(9,I)
      I4 = ABS(IP(4,I))
      INC = 0
C
      CALL METRC4
```

```
      CALL CTIME
C
      DO 320 K=1,4
      KP6 = K+6
      IF(IT4SWT.EQ.1) THEN
        DELU(K) = 0.25*(Q(KP6,I1)+Q(KP6,I2)+Q(KP6,I3)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.2) THEN
        DELU(K) = 0.5*(Q(KP6,I2)+Q(KP6,I4))
      ELSE IF(IT4SWT.EQ.3) THEN
        DELU(K) = 0.5*(Q(KP6,I1)+Q(KP6,I3))
      ELSE IF(IT4SWT.EQ.4) THEN
        DELU(K) = (Q(KP6,I1)+Q(KP6,I3)+Q(KP6,I4))/3.
      END IF
  320 CONTINUE
C
      CALL DELTFG
C
      DO 322 K=1,4
      KP6 = K+6
  322 Q(KP6,I2) = QSAVE(K)+(DELU(K)-DELF(K)+DELG(K))*DV/VT
C
  330 CONTINUE
C
      RETURN
C
C         TYPE 5: NOTHING HERE
  400 GO TO 500
C
C         TYPE 6: NI DISTRIBUTION OF CORNER DU'S FROM SMALL CELL CENTER
  500 IF(INJTYPE.NE.6) RETURN
      DO 530 I = LEVP(1,LEV),LEVP(2,LEV)
C
      IF(IP(5,I).EQ.0) GO TO 530
C
      CALL CELPOINT(I)
C
C         CALCULATE VOLUMES
      VA = -0.5*((Q(1,INC)-Q(1,I1))*(Q(2,IN4)-Q(2,IN1))
     1          -(Q(1,IN4)-Q(1,IN1))*(Q(2,INC)-Q(2,I1)))
      VB = -0.5*((Q(1,IN2)-Q(1,IN1))*(Q(2,INC)-Q(2,I2))
     1          -(Q(1,INC)-Q(1,I2))*(Q(2,IN2)-Q(2,IN1)))
      VC = -0.5*((Q(1,I3)-Q(1,INC))*(Q(2,IN3)-Q(2,IN2))
     1          -(Q(1,IN3)-Q(1,IN2))*(Q(2,I3)-Q(2,INC)))
      VD = -0.5*((Q(1,IN3)-Q(1,IN4))*(Q(2,I4)-Q(2,INC))
     1          -(Q(1,I4)-Q(1,INC))*(Q(2,IN3)-Q(2,IN4)))
      VT = (VA+VB+VC+VD)
C
C         CELL 1
      I1 = ABS(IP(1,I))
      I2 = IP(6,I)
      I3 = IP(5,I)
      I4 = IP(9,I)
      INC = 0
C
```

```
      CALL METRC4
      CALL CTIME
C
      PCHECK = Q(7,I1)+Q(8,I1)+Q(9,I1)+Q(10,I1)
C
      DO 505 K=1,4
      KP6 = K+6
      IF(PCHECK.NE.0.0) THEN
        DELU(K) = Q(KP6,I1)
      ELSE
        DELU(K) = Q(KP6,I3)
      END IF
  505 CONTINUE
C
      CALL DELTFG
C
      DO 507 K=1,4
  507 QSAVE(K) = (DELU(K)+DELF(K)+DELG(K))*DV/VT
C
C         CELL 2
      I1 = I2
      I4 = I3
      I2 = ABS(IP(2,I))
      I3 = IP(7,I)
      INC = 0
C
      CALL METRC4
      CALL CTIME
C
      PCHECK = Q(7,I2)+Q(8,I2)+Q(9,I2)+Q(10,I2)
C
      DO 510 K=1,4
      KP6 = K+6
      IF(PCHECK.NE.0.0) THEN
        DELU(K) = Q(KP6,I2)
      ELSE
        DELU(K) = Q(KP6,I4)
      END IF
  510 CONTINUE
C
      CALL DELTFG
C
      DO 512 K=1,4
  512 QSAVE(K) = QSAVE(K)+(DELU(K)+DELF(K)-DELG(K))*DV/VT
C
C         CELL 3
      I1 = I4
      I2 = I3
      I3 = ABS(IP(3,I))
      I4 = IP(8,I)
      INC = 0
C
      CALL METRC4
      CALL CTIME
```

```
C
      PCHECK = Q(7,I3)+Q(8,I3)+Q(9,I3)+Q(10,I3)
C
      DO 515 K=1,4
      KP6 = K+6
      IF(PCHECK.NE.0.0) THEN
        DELU(K) = Q(KP6,I3)
      ELSE
        DELU(K) = Q(KP6,I1)
      END IF
  515 CONTINUE
C
      CALL DELTFG
C
      DO 517 K=1,4
  517 QSAVE(K) = QSAVE(K)+(DELU(K)-DELF(K)-DELG(K))*DV/VT
C
C
C         CELL 4
      I2 = I1
      I3 = I4
      I1 = IP(9,I)
      I4 = ABS(IP(4,I))
      INC = 0
C
      CALL METRC4
      CALL CTIME
C
      PCHECK = Q(7,I4)+Q(8,I4)+Q(9,I4)+Q(10,I4)
C
      DO 520 K=1,4
      KP6 = K+6
      IF(PCHECK.NE.0.0) THEN
        DELU(K) = Q(KP6,I4)
      ELSE
        DELU(K) = Q(KP6,I2)
      END IF
  520 CONTINUE
C
      CALL DELTFG
C
      DO 522 K=1,4
      KP6 = K+6
  522 Q(KP6,I2) = QSAVE(K)+(DELU(K)-DELF(K)+DELG(K))*DV/VT
C
  530 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: INPUT2
C-----------------------------------------------------------------
      SUBROUTINE INPUT2
C
```

```
C         THIS SUBROUTINE READS REQUIRED INPUT PARAMETERS FROM THE TERMINAL
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'LUNITS.INC'
C
      CHARACTER TLABEL1*10,TLABEL2*100,IN_NAME*50
C
C         SET UP LOGICAL UNIT NUMBERS
C     INPUT:
C           LU3 = POINTER SYSTEM AND FLOW START FILE
C           LU5 = INTERACTIVE INPUT
C           LU7 = STOP COMAND FILE
C     OUTPUT:
C           LU1 = SOLUTION SUMMARY
C           LU2 = RESIDUAL FILE
C           LU4 = POINTER AND SOLUTION FILE
C           LU6 = INTERACTIVE PROMPTS
C
      LU1 = 1
      LU2 = 2
      LU3 = 3
      LU4 = 4
      LU5 = 5
      LU6 = 6
      LU7 = 7
C
C        READ RUN CONDITIONS
      WRITE(LU6,*)' ENTER RUN NAME (RLABEL1<10)'
      READ(LU5,1020)RLABEL1
      WRITE(LU6,*)' ENTER RUN COMMENTS (RLABEL2<100)'
      READ(LU5,1020)RLABEL2
 1020 FORMAT(A)
      WRITE(LU6,*)' ENTER FREE STREAM MACH NO., AMFS'
      READ(LU5,*)AMFS
      WRITE(LU6,*)' ENTER ANGLE OF ATTACK ALPHA'
      READ(LU5,*)ALPHA
      WRITE(LU6,*)' ENTER CFL NO.'
      READ(LU5,*)CFL
      WRITE(LU6,*)' ENTER TYPE OF TIME STEP'
      WRITE(LU6,*)'   0 = SINGLE TIME STEP FOR SWEEP'
      WRITE(LU6,*)'   2 = TIME STEP FOR EACH CELL'
      READ(LU5,*)ITIM
      WRITE(LU6,*)' ENTER ARTIFICIAL VISCOSITY COEF. 0.<AVISCF<0.1'
      READ(LU5,*)AVISCF
      WRITE(LU6,*)' ENTER NUMBER OF ITERATIONS: NSTART,NMAX'
      READ(LU5,*) NSTART,NMAX
      WRITE(LU6,*)' ENTER CONVERGENCE CUT OFF DELSTP'
      READ(LU5,*)DELSTP
      WRITE(LU6,*)' ENTER LEVEL TO CHECK CONVERGENCE ON, LSTOP'
      READ(LU5,*)LSTOP
      WRITE(LU6,*)' DO YOU WANT THE INITIAL FLOW PRINTED??'
      WRITE(LU6,*)'  0=NO   1=YES'
```

```
      READ(LU5,*)IPRNT2
      WRITE(LU6,*) ' ENTER TYPE OF INITIAL SOLUTION'
      WRITE(LU6,*) '  0 = UNIFORM FLOW'
      WRITE(LU6,*) '  1 = RESTART'
      READ(LU5,*) ISTART
      WRITE(LU6,*)' ENTER NAVIER-STOKES SWITCH INSSWT= 1:YES, 0:NO'
      READ(LU5,*)INSSWT
      IF(INSSWT.EQ.1) THEN
        WRITE(LU6,*)' ENTER REO, PR, TREF FOR NAVIER-STOKES SUBDOMAIN'
        READ(LU5,*)REO,PR,TREF
        CSTAR = 110.0/TREF
      ELSE
        REO = 1
        PR = 1
        TREF = 1
        CSTAR = 1
      END IF
C
C         INPUT OF GRID AND POINTER SYSTEM
      OPEN(UNIT=LU3,TYPE='OLD',FORM='UNFORMATTED',
     1      READONLY)
C
      READ(LU3) GLABEL1,GLABEL2,TLABEL1,TLABEL2
      READ(LU3) NICONST,NRCONST
      READ(LU3) (ICONST(K), K=1,NICONST)
      READ(LU3) (RCONST(K), K=1,NRCONST)
      READ(LU3) LMAX,IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
      READ(LU3) ((IPBIMX(M,N), M=1,2), N=1,LMAX)
      READ(LU3) ((LEVP(M,N), M=1,2), N=1,LMAX)
C
      DO 10 LEV = 1,LMAX
      READ(LU3) ((IP(M,N), M=1,9), N=LEVP(1,LEV),LEVP(2,LEV))
   10 CONTINUE
C
      DO 15 LEV=1,LMAX
      IF(IPBIMX(2,LEV).NE.0)
     1  READ(LU3) ((IPBI(M,N), M=1,3),
     2              N=IPBIMX(1,LEV),IPBIMX(2,LEV))
   15 CONTINUE
C
      READ(LU3) ((IPBU(M,N), M=1,3), N=1,IPBUMX)
      READ(LU3) ((IPBD(M,N), M=1,3), N=1,IPBDMX)
      READ(LU3) ((IPBT(M,N), M=1,3), N=1,IPBTMX)
      READ(LU3) ((IPBB(M,N), M=1,3), N=1,IPBBMX)
C
      DO 20 K=1,6
   20 READ(LU3) (Q(K,I), I=1,IQMAX)
C
      CLOSE(UNIT=LU3)
C
C         SET GRID CONSTANTS
      IE    = ICONST(1)
      JE    = ICONST(2)
C
```

```
C         OUTPUT OF INPUT DATA
      WRITE(LU1,1000) RLABEL1,GLABEL1,RLABEL2,GLABEL2
      WRITE(LU1,1001)
      WRITE(LU1,1004) AMFS,ALPHA,CFL,AVISCF,ITIM
      WRITE(LU1,1005) NSTART,NMAX,LSTOP,DELSTP,ISWT
      WRITE(LU1,1006) ISTART,IPRNT1,IPRNT2,REO,TREF
      WRITE(LU1,1007)
      WRITE(LU1,1008) LMAX,IQMAX,IPBUMX,IPBBMX,IPBDMX
      WRITE(LU1,1009) IPBTMX,IE,JE,IC1,IC2
      WRITE(LU1,1010) IF1,IF2,JC2,DELTA,AK
C
 1000 FORMAT(//,5X,A10,2X,A30,/,5X,A100,/,5X,A100)
 1001 FORMAT(//,5X,'INPUT PARAMETERS',/)
 1004 FORMAT(5X,'AMFS     =',E11.4,5X,'ALPHA    =',E11.4,
     1       5X,'CFL      =',E11.4,5X,'AVISCF   =',E11.4,
     2       5X,'ITIM     =',I3)
 1005 FORMAT(5X,'NSTART   =',I6,5X,5X,'NMAX     =',I6,5X,
     1       5X,'LSTOP    =',I6,5X,5X,'DELSTP   =',E11.4,
     2       5X,'ISWT     =',I3)
 1006 FORMAT(5X,'ISTART   =',I4,7X,5X,'IPRNT1   =',I4,7X,
     1       5X,'IPRNT2   =',I4,7X,5X,'REO      =',E11.4,
     2       5X,'TREF     =',E11.4)
 1007 FORMAT(//,5X,'GRID PARAMETERS')
 1008 FORMAT(5X,'LMAX     =',I4,7X,5X,'IQMAX    =',I6,5X,
     1       5X,'IPBUMX   =',I6,5X,5X,'IPBBMX   =',I6,5X,
     2       5X,'IPBDMX   =',I6)
 1009 FORMAT(5X,'IPBTMX   =',I6,5X,5X,'IE       =',I4,7X,
     1       5X,'JE       =',I4,7X,5X,'IC1      =',I4,7X,
     2       5X,'IC2      =',I4,7X)
 1010 FORMAT(5X,'IF1      =',I4,7X,5X,'IF2      =',I4,7X,
     1       5X,'JF2      =',I4,7X,5X,'DELTA    =',E11.4,
     2       5X,'AK       =',E11.4)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: INTERPT
C-----------------------------------------------------------------
      SUBROUTINE INTERPT(LEV,IFORM)
C
C         THIS SUBROUTINE INTERPOLATES THE COARSE GRID
C         SOLUTION TO THE LOCALLY FINEST GRID.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'LUNITS.INC'
C
C         FORM TYPE 1: CENTERED INTERPOLATION
C
      IF(IFORM.NE.1) GO TO 100
C
C         INTERPOLATION TO FINE GRID DU
```

```
      DO 50 LL=1,LEV-1
      L=LEV-LL+1
C
C         UPDATE BOUNDARIES
      IF(INSSWT.EQ.0) THEN
        CALL SDWALBC
CC        CALL EULERWAL2(L)
      ELSE IF(INSSWT.EQ.1) THEN
        CALL NSSDWAL
      END IF
      CALL FARFDBC2(CLN,CDN)
C
      DO 50 I=LEVP(1,L),LEVP(2,L)
C
C         CHECK FOR FINER GRID
      IF(IP(5,I).EQ.0) GO TO 45
C
C         INTERPOLATE CELL
      CALL CELPOINT(I)
C
      EDT(INC) = EDT(I1)
      EDT(IN1) = EDT(I1)
      EDT(IN2) = EDT(I1)
      EDT(IN3) = EDT(I1)
      EDT(IN4) = EDT(I1)
C
      DO 40 K=7,10
      Q(K,IN1) = Q(K,I1)+0.5*(Q(K,I2)-Q(K,I1))
      Q(K,IN2) = Q(K,I2)+0.5*(Q(K,I3)-Q(K,I2))
      Q(K,IN3) = Q(K,I4)+0.5*(Q(K,I3)-Q(K,I4))
      Q(K,IN4) = Q(K,I1)+0.5*(Q(K,I4)-Q(K,I1))
      Q(K,INC) = Q(K,IN1)+0.5*(Q(K,IN3)-Q(K,IN1))
   40 CONTINUE
C
   45 CONTINUE
C
   50 CONTINUE
C
      RETURN
C
C         FORM TYPE 2: WEIGHTED INTERPOLATION
C
  100 IF(IFORM.NE.2) GO TO 200
C
C         INTERPOLATION TO FINE GRID DU
      DO 150 LL=1,LEV-1
      L=LEV-LL+1
C
C         UPDATE BOUNDARIES
      IF(INSSWT.EQ.0) THEN
        CALL SDWALBC
      ELSE IF(INSSWT.EQ.1) THEN
        CALL NSSDWAL
      END IF
```

```
      CALL FAREDBC2
C
      DO 150 I=LEVP(1,L),LEVP(2,L)
C
C         CHECK FOR FINER GRID
      IF(IP(5,I).EQ.0) GO TO 145
C
C         INTERPOLATE CELL
      CALL CELPOINT(I)
C
C
C         CALCULATE SCALINGS FOR NONUNIFORM GRIDS
      TMP1 = SQRT((Q(1,IN1)-Q(1,I1))**2+(Q(2,IN1)-Q(2,I1))**2)
      TMP2 = SQRT((Q(1,I2)-Q(1,IN1))**2+(Q(2,I2)-Q(2,IN1))**2)
      TMP3 = SQRT((Q(1,IN2)-Q(1,I2))**2+(Q(2,IN2)-Q(2,I2))**2)
      TMP4 = SQRT((Q(1,I3)-Q(1,IN2))**2+(Q(2,I3)-Q(2,IN2))**2)
      TMP5 = SQRT((Q(1,IN3)-Q(1,I3))**2+(Q(2,IN3)-Q(2,I3))**2)
      TMP6 = SQRT((Q(1,I4)-Q(1,IN3))**2+(Q(2,I4)-Q(2,IN3))**2)
      TMP7 = SQRT((Q(1,IN4)-Q(1,I4))**2+(Q(2,IN4)-Q(2,I4))**2)
      TMP8 = SQRT((Q(1,I1)-Q(1,IN4))**2+(Q(2,I1)-Q(2,IN4))**2)
      SCAL1 = TMP1/(TMP1+TMP2)
      SCAL2 = TMP3/(TMP3+TMP4)
      SCAL3 = TMP6/(TMP5+TMP6)
      SCAL4 = TMP8/(TMP7+TMP8)
      SCAL5 = 0.5*(SCAL2+SCAL4)
C
      EDT(INC) = EDT(I1)
      EDT(IN1) = EDT(I1)
      EDT(IN2) = EDT(I1)
      EDT(IN3) = EDT(I1)
      EDT(IN4) = EDT(I1)
C
      DO 140 K=7,10
      Q(K,IN1) = Q(K,I1)+SCAL1*(Q(K,I2)-Q(K,I1))
      Q(K,IN2) = Q(K,I2)+SCAL2*(Q(K,I3)-Q(K,I2))
      Q(K,IN3) = Q(K,I4)+SCAL3*(Q(K,I3)-Q(K,I4))
      Q(K,IN4) = Q(K,I1)+SCAL4*(Q(K,I4)-Q(K,I1))
      Q(K,INC) = Q(K,IN1)+SCAL5*(Q(K,IN3)-Q(K,IN1))
  140 CONTINUE
C
  145 CONTINUE
C
  150 CONTINUE
C
      RETURN
C
C         ERROR IN FORM TYPE CHOOSEN
C
  200 WRITE(LU1,*)' WRONG FORM IN INTERPT IFORM=',IFORM
      STOP
      END
C-------------------------------------------------------------
C          SUBROUTINE: METRC4
C-------------------------------------------------------------
```

```
      SUBROUTINE METRC4
C
C         THIS SUBROUTINE CALCULATES THE METRICS FOR THE
C         CENTER OF CELL (I) USING BILINEAR INTERPOLATION
C
      INCLUDE 'MET.INC'
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
C
      DXDXI = 0.5*(Q(1,I3)+Q(1,I4)-Q(1,I2)-Q(1,I1))
      DYDXI = 0.5*(Q(2,I3)+Q(2,I4)-Q(2,I2)-Q(2,I1))
      DXDET = 0.5*(Q(1,I2)+Q(1,I3)-Q(1,I1)-Q(1,I4))
      DYDET = 0.5*(Q(2,I2)+Q(2,I3)-Q(2,I1)-Q(2,I4))
C
C         CALCULATE JACOBIAN
      DV = DXDXI*DYDET-DXDET*DYDXI
      DL = SQRT(DXDET*DXDET+DYDET*DYDET)
      DM = SQRT(DXDXI*DXDXI+DYDXI*DYDXI)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: METRC5
C-----------------------------------------------------------------
      SUBROUTINE METRC5(DXX1,DYX1,DXE2,DYE2,
     1                  DXX3,DYX3,DXE4,DYE4,VT)
C
C         THIS SUBROUTINE CALCULATES THE METRICS FOR THE
C         CENTER OF CELL (I) USING BILINEAR INTERPOLATION
C
      INCLUDE 'MET.INC'
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
C
      DIMENSION VT(3000)
C
      IF(INC.EQ.0) THEN
       DXDXI = 0.5*(Q(1,I3)+Q(1,I4)-Q(1,I2)-Q(1,I1))
       DYDXI = 0.5*(Q(2,I3)+Q(2,I4)-Q(2,I2)-Q(2,I1))
       DXDET = 0.5*(Q(1,I2)+Q(1,I3)-Q(1,I1)-Q(1,I4))
       DYDET = 0.5*(Q(2,I2)+Q(2,I3)-Q(2,I1)-Q(2,I4))
C
       DV = DXDXI*DYDET-DXDET*DYDXI
      ELSE
       DXX1 = Q(1,INC)-Q(1,IN1)
       DYX1 = Q(2,INC)-Q(2,IN1)
       DXE2 = Q(1,IN2)-Q(1,INC)
       DYE2 = Q(2,IN2)-Q(2,INC)
       DXX3 = Q(1,IN3)-Q(1,INC)
       DYX3 = Q(2,IN3)-Q(2,INC)
       DXE4 = Q(1,INC)-Q(1,IN4)
       DYE4 = Q(2,INC)-Q(2,IN4)
       DXDXI = DXX1+DXX3
       DYDXI = DYX1+DYX3
```

```
      DXDET = DXE2+DXE4
      DYDET = DYE2+DYE4
      DV = VT(INC)
     END IF
C
     DL = SQRT(DXDET*DXDET+DYDET*DYDET)
     DM = SQRT(DXDXI*DXDXI+DYDXI*DYDXI)
C
     RETURN
     END
C-----------------------------------------------------------------
C          SUBROUTINE: NI
C-----------------------------------------------------------------
     SUBROUTINE NI
C
C         SUBROUTINE NI DEFINES THE GRID CYCLING
C         FOR THE GENERAL CELL ORIENTED NI SOLVER
C         'NISTEP' WHICH SOLVES THE GOVERNING EQN'S
C         ON EACH LEVEL.  THIS SUBROUTINE THEN CHECKS
C         FOR CONVERGENCE ON THE DEFINED LEVEL.
C
     INCLUDE 'MAIN.INC'
     INCLUDE 'INPT.INC'
     INCLUDE 'LUNITS.INC'
C
     WRITE(LU1,1001)
C
     N  = NSTART-1
     ISTOP = 0
   1 N  = N+1
C
C         RELAXATION SWEEP ON EACH GRID LEVEL FINE TO COARSE
     DO 5 LEV = 1,LMAX
C
C         SOLVE EQN'S
     CALL NISTEP5(N,LEV)
     IF(N.EQ.NSTART.AND.LEV.EQ.1) THEN
       WRITE(LU1,1000)N,IMAX,DELMAX
       DO 2 K=1,5
   2   DELMAX1(K) = DELMAX(K)
     END IF
CCC
CCC         WRITE TEMP RESTART FILE
CCC       TMPREST = FLOAT(N)/100.-FLOAT(N/100)
CCC       IF(LEV.EQ.LSTOP.AND.TMPREST.EQ.0) CALL OUTRESTT
C
C         CHECK FOR CONVERGENCE
     IF(ISTOP.EQ.1.AND.LEV.EQ.1) GOTO 10
     IF(LEV.EQ.LSTOP.AND.DELMAX(5).LE.DELSTP) ISTOP = 1
   5 CONTINUE
C
     IF(N.GE.NMAX-1) ISTOP = 1
     OPEN(UNIT=LU7,READONLY,TYPE='OLD')
     READ(LU7,*)JSTOP
```

```
      IF(JSTOP.EQ.1) ISTOP = 1
      CLOSE(UNIT=LU7)
      GOTO 1
C
   10 WRITE(LU1,1000)N,IMAX,DELMAX
      NFINSH = N
C
 1000 FORMAT(2(2X,I5),6E12.5)
 1001 FORMAT(///,'CONVERGENCE HISTORY',/,4X,'N',5X,'IMAX',
     1       2X,'DELMAX(U1)',2X,'DELMAX(U2)',2X,'DELMAX(U3)',
     2       2X,'DELMAX(U4)',2X,'D(U2/DT)AV')
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: NISTEP5
C-----------------------------------------------------------------
      SUBROUTINE NISTEP5(N,LEV)
C
C         This subroutine solves the Euler eqns.
C         using a cell oreinted version of Ni's Method
C         over grid level LEV.  This subroutine as written
C         performs either a fine mesh cell distribution or
C         a coarse mesh cell acceleration distribution depending
C         of the type of each cell.
C             In addition this particular version saves a
C         representative dt for each node in EDT(i) for use
C         in the error norm calculation.   This same time step
C         then acts as a indicator as to wether the node is to
C         be updated (i.e. if EDT(i)=0.0 then the node has not
C         been distributed to or interpolated to and therefore
C         should not be updated).
C             This subroutine contains a switch which will include
C         the Navier-Stokes terms on level 1 based on the following
C         switch:
C            INSSWT = 0  For Euler solver.
C                     1  For Navier-Stokes terms on level 1.
C                        Note: In this case no smoothing is applied
C                              on level 1.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'SOLV.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'LUNITS.INC'
C
C         Inject changes from the next finer level based on
C         one of the following weighting formulae:
C             IFORM = 0    FOR SIMPLE INJECTION OF VALUE AT INC
C                     1    ALGEBRAIC WEIGHTING
C                     2    AREA WEIGHTING
C                     3    NOTHING DONE AT THIS TIME
C                     4    SPECIAL DISTRIBUTION INJECTION
C                     5    NOTHING DONE AT THIS TIME
```

```
C                     6    Ni's Distribution (modified form 4)
      IF(LEV.GT.1) CALL INJECT5(LEV,6)
C
C         Initialize DU and EDT before sweep
C           If LEV = 1 then all DU's and EDT's zeroed,
      IF(LEV.EQ.1) THEN
        DO 5 I=1,IQMAX
        EDT(I) = 0.0
        DO 5 K=7,10
    5   Q(K,I) = 0.0
C
C           Otherwize zero Du and EDT only at cell nodes.
      ELSE
        DO 7 I=LEVP(1,LEV),LEVP(2,LEV)
        DO 7 J=1,4
        JP = ABS(IP(J,I))
        EDT(JP) = 0.0
        DO 7 K=7,10
    7   Q(K,JP) = 0.0
C
C           In addition zero boundary du's so application
C           of boundary conditions on coarser levels will
C           only make changes at coarse nodes.
        DO 8 I=1,IPBBMX
        JP = ABS(IP(1,IPBB(2,I)))
        DO 8 K=7,10
    8   Q(K,JP) = 0.0
C
        DO 9 I=1,IPBUMX
        JP = ABS(IP(2,IPBB(2,I)))
        DO 9 K=7,10
    9   Q(K,JP) = 0.0
C
      END IF
C
C         Initialize embedded mesh interface nodes from
C         coarser mesh.  This  subroutine may also be used
C         to initalize interface DU's with embedded mesh
C         interface corrections.
      CALL INBC4(LEV)
C
C         If global time step is used calculate DT here
C         based on minimum DT for current level.
      IF(ITIM.EQ.0)  CALL GTIME(LEV)
C
C         Initialize error norms to zero.
      DO 10 K=1,5
   10 DELMAX(K) = 0.0
      DELUMAX = 0.0
C
C         Start of relaxation sweep for DU
C         over current level.
      DO 30 I = LEVP(1,LEV),LEVP(2,LEV)
C
```

```
C         Setup node pointers for cell.
      CALL CELPOINT(I)
C
C         Calculate cell metrics, volume, and other distances
      CALL METRC4
C
C         Calculate time step for local CFL calculations
C         based on current cell.
      IF(ITIM.EQ.2) CALL CTIME
C
C         Store cell DT in EDT(i) for residual calculations
C         Note: set in this way the final value of EDT is
C               the value of the last cell to be calculated
C               which contains this node.  It is note an
C               average.
      EDT(I1) = DT
      EDT(I2) = DT
      EDT(I3) = DT
      EDT(I4) = DT
C
C         Perform flux balance on cell for DELU(k)
C         then calculate distribution weightings
C         DELF and DELG for cell center.
C          Note: If INC = 0 this is a coarse cell
C                and injection is used.
      CALL DELTU
      CALL DELTFG
C
C         If level 1 is Navier-Stokes region calculate terms
      IF(INSSWT.EQ.1.AND.LEV.EQ.1) CALL DELTRS
C
C         Calculate artifical viscosity coefficient
C         if any of the cell nodes is to be smoothed.
      IF(IVIS.GT.0) THEN
        AVIS = AVISCF*DT*(DL+DM)/DV
      END IF
C
C         Distribute cell changes to nodes and if
C         the node is to be smoothed then add smoothing.
      DO 20 K=1,4
      KP6 = K+6
C
C              Distribution step
      Q(KP6,I1) = Q(KP6,I1)+(DELU(K)-DELF(K)-DELG(K))/4.0
      Q(KP6,I2) = Q(KP6,I2)+(DELU(K)-DELF(K)+DELG(K))/4.0
      Q(KP6,I3) = Q(KP6,I3)+(DELU(K)+DELF(K)+DELG(K))/4.0
      Q(KP6,I4) = Q(KP6,I4)+(DELU(K)+DELF(K)-DELG(K))/4.0
C
C              Smoothing step
      IF(INSSWT.EQ.1.AND.LEV.EQ.1) GO TO 20
      IF(IVIS.EQ.0) GO TO 20
      KP2 = K+2
C
      QBAR = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
```

```
      Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(QBAR-Q(KP2,I1))*VIS1
      Q(KP6,I2) = Q(KP6,I2)+0.25*AVIS*(QBAR-Q(KP2,I2))*VIS2
      Q(KP6,I3) = Q(KP6,I3)+0.25*AVIS*(QBAR-Q(KP2,I3))*VIS3
      Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(QBAR-Q(KP2,I4))*VIS4
C
   20 CONTINUE
C
C         Calculate Maximum cell RU residual
C         and its cell location.
      IF(DELUMAX.LT.DELU(2)/DT) THEN
        DELUMAX = DELU(2)/DT
        JMAX = I
      END IF
C
   30 CONTINUE
C
C         Zero embedded mesh interface points and
C         calculate interface corrections to be add
C         to interface points on the next coarser
C         level.
      CALL INFACBC2(LEV)
C
C         Double solid wall boundary DU's.
      CALL WALLDBL
C
C         Correct smoothing at all boundary points
C         (i.e. solid wall and farfield points at
C         this time.).
      CALL BDSMTH(LEV)
C
C         Interpolate DU's from current level to
C         the finest level in each mesh region.
C            IFORM = 1 For centered interpolation (i.e. algebraic)
C                    2 For interpolation based on physical lengths
      IF(LEV.NE.1) CALL INTERPT(LEV,1)
C
C         Apply boundary conditions to all
C         boundary points.
      IF(INSSWT.EQ.0) THEN
        CALL SDWALBC
CC          CALL EULERWAL(LEV)
      ELSE IF(INSSWT.EQ.1) THEN
        CALL NSSDWAL
      END IF
      CALL FARFDBC2(CLN,CDN)
C
C         Update solution for all points
C         that have been changed and calculate
C         node error norms.
      NUMPTS = 0
      DO 60 I = 1,IQMAX
      IF(EDT(I).EQ.0.0) GO TO 60
      NUMPTS = NUMPTS+1
      DO 55 K = 1,4
```

```
      KP6 = K+6
      KP2 = K+2
      IF (DELMAX(K).LT.ABS(Q(KP6,I)/EDT(I))) THEN
         DELMAX(K) = ABS(Q(KP6,I)/EDT(I))
         IF (K.EQ.2) THEN
            IMAX = I
         END IF
      END IF
   55 Q(KP2,I) = Q(KP2,I)+Q(KP6,I)
      DELMAX(5) = DELMAX(5)+ABS(Q(8,I)/EDT(I))
   60 CONTINUE
C
      DELMAX(5) = DELMAX(5)/FLOAT(NUMPTS)
C
C         Write out error norms to plot file if
C         LEV is less than or equal to LSTOP.
      IF(LEV.LE.LSTOP) WRITE(LU2,1000) N,IMAX,DELMAX,JMAX,DELUMAX,
     1                                 CLN,CDN
 1000 FORMAT(2(2X,I5),5E12.5,2X,I5,E12.5,2X,E12.5,2X,E12.5)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: NSSDWAL
C-----------------------------------------------------------------
      SUBROUTINE NSSDWAL
C
C         THIS SUBROUTINE CALCULATES DU FOR WALL
C         BOUNDARY POINTS FOR THE NAVIER-STOKES EQN.
C         USING NORMAL EXTRAPOLATION OF PRESSURE AND TEMPERATURE
C         I.E. ADIABATIC WALL CONDITION.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'LUNITS.INC'
C
C         BOTTOM WALL
      DO 10 I=1,IPBBMX
C
C         SET POINTERS & CALCULATE WALL TANGENT
      IF(IPBB(3,I).EQ.4) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
        J4 = ABS(IP(3,IPBB(1,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
```

```
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
        IBCOND = 1
      ELSE IF(IPBB(3,I).EQ.5) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
        J4 = ABS(IP(3,IPBB(1,I)))
C
        THETA1 = ATAN2((Q(2,J2)-Q(2,J1)),(Q(1,J2)-Q(1,J1)))
        THETA2 = ATAN2((Q(2,J2)-Q(2,J3)),(Q(1,J2)-Q(1,J3)))
        THETA  = 0.5*(THETA1+THETA2)
CC        THETA = THETA1
C
        DX = COS(THETA)
        DY = SIN(THETA)
        IBCOND = 1
      ELSE
        WRITE(LU1,*)' ERROR WALBC9C NOT VALID WALL TYPE I=',I
      END IF
      IF(IBCOND.EQ.0) GOTO 10
C
C         CALCULATION OF DWI,DWN,A0
      RTMP1 = Q(3,J4)
      UTMP1 = Q(4,J4)/RTMP1
      VTMP1 = Q(5,J4)/RTMP1
      ETMP1 = Q(6,J4)
      PTMP1 = GM1*(ETMP1-0.5*RTMP1*(UTMP1*UTMP1+VTMP1*VTMP1))
      TTMP1 = GAMMA*PTMP1/RTMP1
      IF(PTMP1.LT.0.0) THEN
         WRITE(LU1,*) '** PTMP1<0.0 IN SDWALBC AT BOTTOM I=',I
         STOP
      END IF
C
C         CALCULATION OF CORRECTED DELTA'S
      Q(7,J2)  = GAMMA*PTMP1/TTMP1-Q(3,J2)
      Q(8,J2)  = 0.0-Q(4,J2)
      Q(9,J2)  = 0.0-Q(5,J2)
      Q(10,J2) = PTMP1/GM1-Q(6,J2)
C
   10 CONTINUE
C
      RETURN
      END
C-------------------------------------------------------------
```

```
C          SUBROUTINE: OUTPUT3
C-----------------------------------------------------------------
      SUBROUTINE OUTPUT3
C
C         THIS SUBROUTINE CREATES THE OUTPUT FILE
C         CALL REST.DAT WHICH IS READ BY EULER
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'LUNITS.INC'
C
C         CALCULATION OF LIFT FORCE COEFFICIENTS
      CORD = 0.0
      CFN  = 0.0
      CFT  = 0.0
C
      DO 5 I=1,IPBBMX
      J1 = ABS(IP(1,IPBB(2,I)))
      J2 = ABS(IP(4,IPBB(2,I)))
C
      IF(I.EQ.1) THEN
        TX1 = Q(1,J1)
        TY1 = Q(2,J1)
      END IF
      TCORD = (TX1-Q(1,J2))**2+(TY1-Q(2,J2))**2
      IF(TCORD.GT.CORD) CORD = TCORD
C
      DX = Q(1,J2)-Q(1,J1)
      DY = Q(2,J2)-Q(2,J1)
      DS = SQRT(DX*DX+DY*DY)
C
      P1 = GM1*(Q(6,J1)
     1    -0.5*(Q(4,J1)*Q(4,J1)+Q(5,J1)*Q(5,J1))/Q(3,J1))
      P2 = GM1*(Q(6,J2)
     1    -0.5*(Q(4,J2)*Q(4,J2)+Q(5,J2)*Q(5,J2))/Q(3,J2))
      TMP = P1+P2
      CFN = CFN+TMP*DX
      CFT = CFT+TMP*DY
    5 CONTINUE
C
      CORD = SQRT(CORD)
      QFS = ROFS*(UFS*UFS+VFS*VFS)*CORD
      CFN = -CFN/QFS
      CFT = CFT/QFS
C
      ALPHAR = 3.14159*ALPHA/180.0
      CL = CFN*COS(ALPHAR)-CFT*SIN(ALPHAR)
      CD = CFN*SIN(ALPHAR)+CFT*COS(ALPHAR)
C
C         CALCULATE SPECTRIAL RADIUS
      SRAD = (DELMAX(5)/DELMAX1(5))**(1./(NFINSH-NSTART))
C
```

```
      OPEN(UNIT=LU4,TYPE='NEW',FORM='UNFORMATTED')
C
C         SET CONSTANTS
      ICONST(11) = NSTART
      ICONST(12) = NFINSH
      ICONST(13) = ITIM
      ICONST(14) = ISTART
      ICONST(15) = LSTOP
      NICONST = 50
C
      RCONST(1)  = AMFS
      RCONST(2)  = ALPHA
      RCONST(3)  = CFL
      RCONST(4)  = AVISCF
      RCONST(5)  = ROFS
      RCONST(6)  = UFS
      RCONST(7)  = VFS
      RCONST(8)  = APFS
      RCONST(9)  = CORD
      RCONST(10) = CFN
      RCONST(11) = CFT
      RCONST(12) = CL
      RCONST(13) = CD
      RCONST(14) = CM
      RCONST(15) = DELMAX1(1)
      RCONST(16) = DELMAX1(2)
      RCONST(16) = DELMAX1(3)
      RCONST(18) = DELMAX1(4)
      RCONST(19) = DELMAX1(5)
      RCONST(20) = DELMAX(1)
      RCONST(21) = DELMAX(2)
      RCONST(22) = DELMAX(3)
      RCONST(23) = DELMAX(4)
      RCONST(24) = DELMAX(5)
      RCONST(25) = SRAD
      NRCONST = 50
C
      WRITE(LU1,1004)
      WRITE(LU1,1005)ROFS,UFS,VFS,CORD
      WRITE(LU1,1006)CFN,CFT,CL,CD
 1004 FORMAT(//,5X,'SECTION LIFT PROPERTIES',/)
 1005 FORMAT(5X,'ROFS     =',F10.7,5X,'UFS      =',F10.7,
     1       5X,'VFS      =',F10.7,5X,'CHORD    =',F10.7)
 1006 FORMAT(5X,'CFN      =',F10.7,5X,'CFT      =',F10.7,
     1       5X,'CL       =',F10.7,5X,'CD       =',F10.7)
C
C         WRITE OUT GRID POINTERS
C
      WRITE(LU4) GLABEL1,GLABEL2,RLABEL1,RLABEL2
      WRITE(LU4) NICONST,NRCONST
      WRITE(LU4) (ICONST(K), K=1,NICONST)
      WRITE(LU4) (RCONST(K), K=1,NRCONST)
      WRITE(LU4) LMAX,IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
      WRITE(LU4) ((IPBIMX(M,N), M=1,2), N=1,LMAX)
```

```
      WRITE(LU4) ((LEVP(M,N), M=1,2), N=1,LMAX)
C
      DO 10 LEV = 1,LMAX
      WRITE(LU4) ((IP(M,N), M=1,9), N=LEVP(1,LEV),LEVP(2,LEV))
   10 CONTINUE
C
      DO 15 LEV=1,LMAX
      IF(IPBIMX(2,LEV).NE.0)
     1  WRITE(LU4) ((IPBI(M,N), M=1,3), N=IPBIMX(1,LEV),IPBIMX(2,LEV))
   15 CONTINUE
C
      WRITE(LU4) ((IPBU(M,N), M=1,3), N=1,IPBUMX)
      WRITE(LU4) ((IPBD(M,N), M=1,3), N=1,IPBDMX)
      WRITE(LU4) ((IPBT(M,N), M=1,3), N=1,IPBTMX)
      WRITE(LU4) ((IPBB(M,N), M=1,3), N=1,IPBBMX)
C
      DO 8 K =1,6
    8 WRITE(LU4) (Q(K,I), I=1,IQMAX)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE:OUTRESTT
C-----------------------------------------------------------------
      SUBROUTINE OUTRESTT
C
C         THIS SUBROUTINE CREATES A TEMPORARY OUTPUT FILE
C         CALL TREST.DAT WHICH IS READ BY EULER
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'GAM.INC'
C
C         CALCULATION OF LIFT FORCE COEFFICIENTS
      CORD = 0.0
      CFN  = 0.0
      CFT  = 0.0
C
      DO 5 I=1,IPBBMX
      J1 = ABS(IP(1,IPBB(2,I)))
      J2 = ABS(IP(4,IPBB(2,I)))
C
      IF(I.EQ.1) THEN
        TX1 = Q(1,J1)
        TY1 = Q(2,J1)
      END IF
      TCORD = (TX1-Q(1,J2))**2+(TY1-Q(2,J2))**2
      IF(TCORD.GT.CORD) CORD = TCORD
C
      DX = Q(1,J2)-Q(1,J1)
      DY = Q(2,J2)-Q(2,J1)
      DS = SQRT(DX*DX+DY*DY)
C
```

```
      P1 = GM1*(Q(6,J1)
     1    -0.5*(Q(4,J1)*Q(4,J1)+Q(5,J1)*Q(5,J1))/Q(3,J1))
      P2 = GM1*(Q(6,J2)
     1    -0.5*(Q(4,J2)*Q(4,J2)+Q(5,J2)*Q(5,J2))/Q(3,J2))
      TMP = P1+P2
      CFN = CFN+TMP*DX
      CFT = CFT+TMP*DY
    5 CONTINUE
C
      CORD = SQRT(CORD)
      QFS = ROFS*(UFS*UFS+VFS*VFS)*CORD
      CFN = -CFN/QFS
      CFT = CFT/QFS
C
      ALPHAR = 3.14159*ALPHA/180.0
      CL = CFN*COS(ALPHAR)-CFT*SIN(ALPHAR)
      CD = CFN*SIN(ALPHAR)+CFT*COS(ALPHAR)
C
C         CALCULATE SPECTRIAL RADIUS
      SRAD = (DELMAX(5)/DELMAX1(5))**(1./(NFINSH-NSTART))
C
      OPEN(UNIT=8,NAME='TREST.DAT',TYPE='OLD',FORM='UNFORMATTED')
C
C         SET CONSTANTS
      ICONST(11) = NSTART
      ICONST(12) = NFINSH
      ICONST(13) = ITIM
      ICONST(14) = ISTART
      ICONST(15) = LSTOP
      NICONST = 50
C
      RCONST(1)  = AMFS
      RCONST(2)  = ALPHA
      RCONST(3)  = CFL
      RCONST(4)  = AVISCF
      RCONST(5)  = ROFS
      RCONST(6)  = UFS
      RCONST(7)  = VFS
      RCONST(8)  = APFS
      RCONST(9)  = CORD
      RCONST(10) = CFN
      RCONST(11) = CFT
      RCONST(12) = CL
      RCONST(13) = CD
      RCONST(14) = CM
      RCONST(15) = DELMAX1(1)
      RCONST(16) = DELMAX1(2)
      RCONST(16) = DELMAX1(3)
      RCONST(18) = DELMAX1(4)
      RCONST(19) = DELMAX1(5)
      RCONST(20) = DELMAX(1)
      RCONST(21) = DELMAX(2)
      RCONST(22) = DELMAX(3)
      RCONST(23) = DELMAX(4)
```

```
      RCONST(24) = DELMAX(5)
      RCONST(25) = SRAD
      NRCONST = 50
C
 1004 FORMAT(//,5X,'SECTION LIFT PROPERTIES',/)
 1005 FORMAT(5X,'ROFS     =',F10.7,5X,'UFS      =',F10.7,
     1       5X,'VFS      =',F10.7,5X,'CHORD    =',F10.7)
 1006 FORMAT(5X,'CFN      =',F10.7,5X,'CFT      =',F10.7,
     1       5X,'CL       =',F10.7,5X,'CD       =',F10.7)
C
C         WRITE OUT GRID POINTERS
C
      WRITE(8) GLABEL1,GLABEL2,RLABEL1,RLABEL2
      WRITE(8) NICONST,NRCONST
      WRITE(8) (ICONST(K), K=1,NICONST)
      WRITE(8) (RCONST(K), K=1,NRCONST)
      WRITE(8) LMAX,IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
      WRITE(8) ((IPBIMX(M,N), M=1,2), N=1,LMAX)
      WRITE(8) ((LEVP(M,N), M=1,2), N=1,LMAX)
C
      DO 10 LEV = 1,LMAX
      WRITE(8) ((IP(M,N), M=1,9), N=LEVP(1,LEV),LEVP(2,LEV))
   10 CONTINUE
C
      DO 15 LEV=1,LMAX
      IF(IPBIMX(2,LEV).NE.0)
     1  WRITE(8) ((IPBI(M,N), M=1,3), N=IPBIMX(1,LEV),IPBIMX(2,LEV))
   15 CONTINUE
C
      WRITE(8) ((IPBU(M,N), M=1,3), N=1,IPBUMX)
      WRITE(8) ((IPBD(M,N), M=1,3), N=1,IPBDMX)
      WRITE(8) ((IPBT(M,N), M=1,3), N=1,IPBTMX)
      WRITE(8) ((IPBB(M,N), M=1,3), N=1,IPBBMX)
C
      DO 8 K =1,6
    8 WRITE(8) (Q(K,I), I=1,IQMAX)
C
      CLOSE(UNIT=8)
C
      RETURN
      END
C-------------------------------------------------------------
C          SUBROUTINE: SDWALBC
C-------------------------------------------------------------
      SUBROUTINE SDWALBC
C
C         THIS SUBROUTINE CALCULATES DU FOR WALL
C         BOUNDARY POINTS USING A SIMPLE WAVE
C         BC FOR THE 4 EQN EULER PROBLEM.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'LUNITS.INC'
```

```
C
C         BOTTOM WALL
      DO 10 I=1,IPBBMX
C
C         SET POINTERS & CALCULATE WALL TANGENT
      IF(IPBB(3,I).EQ.4) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
        IBCOND = 1
      ELSE IF(IPBB(3,I).EQ.5) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        THETA1 = ATAN2((Q(2,J2)-Q(2,J1)),(Q(1,J2)-Q(1,J1)))
        THETA2 = ATAN2((Q(2,J2)-Q(2,J3)),(Q(1,J2)-Q(1,J3)))
        THETA  = 0.5*(THETA1+THETA2)
CC        THETA = THETA1
C
        DX = COS(THETA)
        DY = SIN(THETA)
        IBCOND = 0
      ELSE
        WRITE(LU1,*)' ERROR WALBC9C NOT VALID WALL TYPE I=',I
      END IF
      IF(IBCOND.EQ.0) GOTO 10
C
C         CALCULATION OF DWT,DWN,A0
      RTMP1 = Q(3,J2)+Q(7,J2)
      UTMP1 = (Q(4,J2)+Q(8,J2))/RTMP1
      VTMP1 = (Q(5,J2)+Q(9,J2))/RTMP1
      ETMP1 = Q(6,J2)+Q(10,J2)
      PTMP1 = GM1*(ETMP1-0.5*RTMP1*(UTMP1*UTMP1+VTMP1*VTMP1))
      IF(PTMP1.LT.0.0) THEN
```

```
         WRITE(LU1,*) '** PTMP1<0.0 IN SDWALBC AT BOTTOM I=',I
         STOP
       END IF
       IF(IBCOND.EQ.1) THEN
         A01   = SQRT(GAMMA*PTMP1/RTMP1)
         UWT1 = DX*UTMP1+DY*VTMP1
         DWN1 =-DY*UTMP1+DX*VTMP1
C
         UNEW = UWT1*DX
         VNEW = UWT1*DY
         TMP = 1.0-0.5*GM1*DWN1/A01
         IF(TMP.LT.0.0) THEN
            WRITE(LU1,*)' **TMP<0.0 IN SDWALBC AT BOTTOM I=',I
            STOP
         END IF
         RNEW = RTMP1*((TMP)**(2.0/GM1))
         PNEW = PTMP1*((TMP)**(2.0/GM1DG))
       ELSE IF(IBCOND.EQ.2) THEN
         UNEW = 0.0
         VNEW = 0.0
         PNEW = (PTMP1**GM1DG+GM1D2G*RTMP1*(UTMP1**2+VTMP1**2)
      1                        /(PTMP1**(1./GAMMA)))**(1.0/GM1DG)
         RNEW = RTMP1*((PNEW/PTMP1)**(1./GAMMA))
       ELSE IF(IBCOND.EQ.3) THEN
         UNEW = 0.0
         VNEW = 0.0
         PTMPA = GM1*(Q(6,J1)-0.5*(Q(4,J1)**2+Q(5,J1)**2)/Q(3,J1))
         PTMPB = GM1*(Q(6,J3)-0.5*(Q(4,J3)**2+Q(5,J3)**2)/Q(3,J3))
         PNEWA = (PTMPA**GM1DG+GM1D2G*(Q(4,J1)**2+Q(5,J1)**2)/Q(3,J1)
      1                        /(PTMPA**(1./GAMMA)))**(1.0/GM1DG)
         PNEWB = (PTMPB**GM1DG+GM1D2G*(Q(4,J3)**2+Q(5,J3)**2)/Q(3,J3)
      1                        /(PTMPB**(1./GAMMA)))**(1.0/GM1DG)
         PNEW = 0.5*(PNEWA+PNEWB)
         RNEW = RTMP1*((PNEW/PTMPA)**(1./GAMMA))
       ELSE IF(IBCOND.EQ.4) THEN
         QNEW = SQRT(UTMP1**2+VTMP1**2)
         UNEW = QNEW*DX
         VNEW = QNEW*DY
         PNEW = PTMP1
         RNEW = RTMP1
       END IF
C
C         CALCULATION OF CORRECTED DELTA'S
       Q(7,J2)  = RNEW-Q(3,J2)
       Q(8,J2)  = RNEW*UNEW-Q(4,J2)
       Q(9,J2)  = RNEW*VNEW-Q(5,J2)
       Q(10,J2) = PNEW/GM1+0.5*RNEW*(UNEW*UNEW+VNEW*VNEW)
      1           -Q(6,J2)
C
   10 CONTINUE
C
       RETURN
       END
C-------------------------------------------------------------
```

```
C          SUBROUTINE: WALLDBL
C-----------------------------------------------------------------
      SUBROUTINE WALLDBL
C
C         THIS SUBROUTINE DOUBLES THE SOLID WALL
C         DU'S BEFORE INTERPOLATION AND APPLICATION
C         OF BOUNDARY CONDITIONS.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'LUNITS.INC'
C
C         BOTTOM WALL POINTS
      DO 5 I=1,IPBBMX
      IF(IPBB(3,I).EQ.4) THEN
        IF(IPBB(1,I).NE.0) THEN
          J2 = ABS(IP(4,IPBB(1,I)))
        ELSE
          J2 = ABS(IP(1,IPBB(2,I)))
        END IF
      ELSE IF(IPBB(3,I).EQ.5) THEN
        IF(IPBB(1,I).NE.0) THEN
          J2 = ABS(IP(4,IPBB(1,I)))
        ELSE
          J2 = ABS(IP(1,IPBB(2,I)))
        END IF
      ELSE
        WRITE(LU1,*)' ERROR WALLDBL IPBB(3,I) NOT 4 OR 5'
      END IF
      DO 5 K=7,10
    5 Q(K,J2) = 2.0*Q(K,J2)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: ZERO4
C-----------------------------------------------------------------
      SUBROUTINE ZERO4(LEV)
C
C         THIE SUBROUTINE SETS SUBDOMAIN-GLOBAL
C         BOUNDARY DU'S TO ZERO.
C
      INCLUDE 'MAIN.INC'
C
      IF(IPBIMX(1,LEV).EQ.0) RETURN
C
      DO 5 I=IPBIMX(1,LEV),IPBIMX(2,LEV)
      DO 5 J=1,3
      JPOINT = IPBI(J,I)
      DO 5 K=7,10
    5 Q(K,JPOINT) = 0.0
C
      RETURN
      END
C-----------------------------------------------------------------
```

```
C          LINK COMMAND FILE: AFLLINK.COM
C--------------------------------------------------------------
SET1 := BDSMTH,CELPOINT,CTIME,DELTFG,DELTU,METRC5,-
        INFACBC2,INTERPT,NISTEP5,INJECT5,EULERWAL2,
SET2 := FINDFG,GAMMAS,GEOWAL,GTIME,INBC4,INITIA,INFACBC,
SET3 := INPUT2,METRC4,NI,
SET4 := OUTPUT3,FARFDBC2,SDWALBC,WALLDBL,-
        ZERO4,DELTRS,NSSDWAL,OUTRESTT
LINK EULERCELL,'SET1' 'SET2' 'SET3' 'SET4'
C--------------------------------------------------------------
C          DATA FILE: AIRFOIL.TMP
C--------------------------------------------------------------
RUN 216:
EULER, NI INJECTION(6), CHAR S/W, VORTEX DS CHAR. QUARTER CHORD
.85          ENTER FREE STREAM MACH NO., AMFS
1.0          ENTER ANGLE OF ATTACK
.95          ENTER CFL NO.
2            ENTER TYPE OF TIME STEP 0 = SINGLE TIME 2 = EACH CELL
.08          ENTER ARTIFICIAL VISCOSITY COEF. 0.<AVISCF<0.1
1  2000 1500      ENTER NUMBER OF ITERATIONS: NSTART,NMAX
1E-5         ENTER CONVERGENCE CUT OFF DELSTP
1            ENTER LEVEL TO CHECK CONVERGENCE ON, LSTOP
0            DO YOU WANT THE INITIAL FLOW PRINTED??  0=NO   1=YES
0            ENTER TYPE OF INITIAL SOLUTION 0 = UNIFORM FLOW 1 = RESTART
[CFD.USAB.EULERCELL.GRIDFOIL]NACA0012.015
0            ENTER NAVIER-STOKES SWITCH INSSWT= 1:YES, 0:NO
2.342E6 .72 288.0    ENTER RE0,PR,TREF FOR NAVIER-STOKES SUBDOMAIN
C--------------------------------------------------------------
C**************************************************************
C--------------------------------------------------------------
```

APPENDIX F

2-D AIRFOIL EULER CODE FOR C-TYPE MESHES

This appendix contains listings of the two computer codes used to generate the 2-D transonic airfoil solutions on C-type meshes presented in this thesis. The first program called GEOCREATC reads in a global C-type mesh as input and then interactively generates a file which contains the pointer system and mesh coordinates for the complete mesh structure. The second program, EULERCELL, is the multiple-grid Euler solver. EULERCELL requires two files as input, the pointer file defining the embedded mesh structure and a second file containing the flow conditions and control parameters (a sample of this file is given at the end of this appendix). Since most of the subroutines used by GEOCREATC and EULERCELL are the same as those used by the corresponding O-mesh codes, only those subroutines which are different are presented here. All others may be taken directly from the O-mesh codes.

```
12345678901234567890123456789012345678901234567890123456789012345
C-----------------------------------------------------------------
C*****************************************************************
C-----------------------------------------------------------------
C
C-----------------------------------------------------------------
C          PROGRAM: GEOCREATC
C-----------------------------------------------------------------
C
C         THIS PROGRAM GENERATES THE GRID AND POINTERS
C         FOR A AIRFOIL WITH A CGRID.
C
      CALL GEOIN3
C
      CALL GEOPONTC
C
      CALL GEOOUT3
C
      STOP
      END
C-----------------------------------------------------------------
C          INCLUDE FILE: GEOCOM.COM
C-----------------------------------------------------------------
      COMMON/GEOCOM/IE,JE,IEM1,JEM1,LMAX,ISUBD,
     1              DELTA,AK,Y0,IC1,IC2,JC2,IF2,JF2,
     2              X(257,65),Y(257,65),XS(49,9),YS(49,9),
     3              IPC(257,65),IPS(49,9),IQMAX,Q(2,8424),
     4              ILEVP(2,5),LEVP(2,5),IP(9,10816),
     5              IPBIMX(2,5),IPBUMX,IPBDMX,IPBTMX,IPBBMX,
     6              IPBI(3,257),IPBU(3,257),IPBD(3,9),
     7              IPBT(3,33),IPBB(3,257),
     8              LEVSET(257,65),IPSET(257,65),
     9              ICONST(50),RCONST(50),LU1,LU2,LU3,LU5,LU6
      COMMON/GEOLAB/GLABEL1,GLABEL2,RLABEL1,RLABEL2,
     1             IN_NAME,OUT_NAME
      CHARACTER GLABEL1*30,GLABEL2*100,RLABEL1*10,RLABEL2*100,
     1          IN_NAME*15,OUT_NAME*15
      COMMON/GAM/ GAMMA,HTOT,
     1            GM1,GM1D2,GM1DG,GM1D2G,
     2            GP1DG,GP1D2G,GM3
C-----------------------------------------------------------------
C          SUBROUTINE FILE: GEOPONTC
C-----------------------------------------------------------------
      SUBROUTINE GEOPONTC
C
C         THIS SUBROUTINE GENERATES THE POINTERS
C         FOR THE GIVEN GRID AND BOUNDARY CONDITIONS.
C         THIS IS A EXTENDED VERSION OF GEOPONT3.
C
      INCLUDE 'GEOCOM.COM'
C
C         FIND T/E LOCATION
      I = 0
      ITE = 0
```

```
    5 I = I+1
      IF(Y(I,1).EQ.Y(IE-I+1,1)) THEN
        ITE = I
        GO TO 5
      END IF
      WRITE(LU6,*)' DOUBLE T/E SETUP'
CCC
      ITE =ITE-1
CCC
      WRITE(LU6,*)' ITE =',ITE
C
C         SETUP AND FILLING OF Q-VECTOR
      IND = 0
      DO 10 L=1,LMAX
      LSKIP = 2**(L-1)
      DO 10 I=1,IE,LSKIP
      DO 10 J=1,JE,LSKIP
      IF(LEVSET(I,J).EQ.L) THEN
        IF(J.GT.1.OR.I.LE.IE-ITE) THEN
          IND = IND+1
          Q(1,IND) = X(I,J)
          Q(2,IND) = Y(I,J)
          IPSET(I,J) = IND
        ELSE
          IPSET(I,J) = IPSET(IE-I+1,J)
        END IF
      END IF
   10 CONTINUE
C
      IQMAX = IND
      WRITE(LU6,*) ' IQMAX=' , IQMAX
C
C         GENERATE POINTERS
      IND = 0
C
      DO 30 LEV=1,LMAX
      ILEVP(1,LEV) = IND+1
      LSKIP = 2**(LEV-1)
      LSKIP2 = LSKIP/2
      DO 25 I=1,IEM1,LSKIP
      DO 25 J=1,JEM1,LSKIP
      ICOUNT = 0
      IF(LEVSET(I,J).LE.LEV)              ICOUNT = ICOUNT+1
      IF(LEVSET(I,J+LSKIP).LE.LEV)        ICOUNT = ICOUNT+1
      IF(LEVSET(I+LSKIP,J+LSKIP).LE.LEV) ICOUNT = ICOUNT+1
      IF(LEVSET(I+LSKIP,J).LE.LEV)        ICOUNT = ICOUNT+1
      IF(ICOUNT.EQ.4) THEN
        IND = IND+1
        IP(1,IND) = -IPSET(I,J)
        IP(2,IND) = -IPSET(I,J+LSKIP)
        IP(3,IND) = -IPSET(I+LSKIP,J+LSKIP)
        IP(4,IND) = -IPSET(I+LSKIP,J)
        IF(LEV.EQ.1) THEN
          IF(J.GT.1) THEN
```

```
            IF(I.GT.1) THEN
              IF(LEVSET(I-1,J).EQ.1.AND.LEVSET(I,J-1).EQ.1)
     1         IP(1,IND) = -IP(1,IND)
            END IF
            IF(I.LT.IEM1) THEN
              IF(LEVSET(I+2,J).EQ.1.AND.LEVSET(I+1,J-1).EQ.1)
     1         IP(4,IND) = -IP(4,IND)
            END IF
          ELSE
            IF(I.LE.ITE.OR.I.GE.IE-ITE+1) THEN
              IF(I.NE.1) IP(1,IND) = -IP(1,IND)
            END IF
            IF(I.LT.ITE.OR.I.GE.IE-ITE) THEN
              IF(I.NE.IEM1) IP(4,IND) = -IP(4,IND)
            END IF
          END IF
          IF(J.LT.JEM1) THEN
            IF(I.GT.1) THEN
              IF(LEVSET(I-1,J+1).EQ.1.AND.LEVSET(I,J+2).EQ.1)
     1         IP(2,IND) = -IP(2,IND)
            END IF
            IF(I.LT.IEM1) THEN
              IF(LEVSET(I+2,J+1).EQ.1.AND.LEVSET(I+1,J+2).EQ.1)
     1         IP(3,IND) = -IP(3,IND)
            END IF
          END IF
          DO 17 K=5,9
   17     IP(K,IND) = 0
        ELSE
          IF(LEVSET(I+LSKIP2,J+LSKIP2).EQ.LEV) THEN
            IF(J.GT.1) THEN
              IF(I.GT.1) THEN
                IF(LEVSET(I-LSKIP,J).LE.LEV
     1            .AND.LEVSET(I,J-LSKIP).LE.LEV)
     1           IP(1,IND) = -IP(1,IND)
              END IF
              IF(I.LT.IE-LSKIP) THEN
                IF(LEVSET(I+2*LSKIP,J).LE.LEV
     1            .AND.LEVSET(I+LSKIP,J-LSKIP).LE.LEV)
     1           IP(4,IND) = -IP(4,IND)
              END IF
            ELSE
C
C     **** ADD EMBEDDED MESH SYMMETRY CONDITIONS
C
              WRITE(LUG,*)' *** ERROR SYMMETRY CONDITION MISSING'
            END IF
            IF(J.LT.JE-LSKIP) THEN
              IF(I.GT.1) THEN
                IF(LEVSET(I-LSKIP,J+LSKIP).LE.LEV
     1            .AND.LEVSET(I,J+2*LSKIP).LE.LEV)
     1           IP(2,IND) = -IP(2,IND)
              END IF
              IF(I.LT.IE-LSKIP) THEN
```

```
              IF(LEVSET(I+2*LSKIP,J+LSKIP).LE.LEV
     1          .AND.LEVSET(I+LSKIP,J+2*LSKIP).LE.LEV)
     1         IP(3,IND) = -IP(3,IND)
            END IF
          END IF
        ELSE IF(LEVSET(I+LSKIP2,J+LSKIP2).EQ.LEV-1) THEN
          IF(J.GT.1) THEN
            IF(I.GT.1) THEN
              IF(LEVSET(I-LSKIP,J).EQ.LEV
     1          .OR.LEVSET(I,J-LSKIP).EQ.LEV)
     1         IP(1,IND) = -IP(1,IND)
            END IF
            IF(I.LT.IE-LSKIP) THEN
              IF(LEVSET(I+2*LSKIP,J).EQ.LEV
     1          .OR.LEVSET(I+LSKIP,J-LSKIP).EQ.LEV)
     1         IP(4,IND) = -IP(4,IND)
            END IF
          ELSE
C
C     ***** ADD EMBEDDED MESH SYMMETRY CONDITIONS
C
            WRITE(LU6,*)' *** ERROR SYMMETRY CONDITION MISSING'
          END IF
          IF(J.LT.JE-LSKIP) THEN
            IF(I.GT.1) THEN
              IF(LEVSET(I-LSKIP,J+LSKIP).EQ.LEV
     1          .OR.LEVSET(I,J+2*LSKIP).EQ.LEV)
     1         IP(2,IND) = -IP(2,IND)
            END IF
            IF(I.LT.IE-LSKIP) THEN
              IF(LEVSET(I+2*LSKIP,J+LSKIP).EQ.LEV
     1          .OR.LEVSET(I+LSKIP,J+2*LSKIP).EQ.LEV)
     1         IP(3,IND) = -IP(3,IND)
            END IF
          END IF
        END IF
        IF(LEVSET(I+LSKIP2,J+LSKIP2).LT.LEV)
     1       IP(5,IND) = IPSET(I+LSKIP2,J+LSKIP2)
        IF(LEVSET(I,J+LSKIP2).LT.LEV)
     1       IP(6,IND) = IPSET(I,J+LSKIP2)
        IF(LEVSET(I+LSKIP2,J+LSKIP).LT.LEV)
     1       IP(7,IND) = IPSET(I+LSKIP2,J+LSKIP)
        IF(LEVSET(I+LSKIP,J+LSKIP2).LT.LEV)
     1       IP(8,IND) = IPSET(I+LSKIP,J+LSKIP2)
        IF(LEVSET(I+LSKIP2,J).LT.LEV)
     1       IP(9,IND) = IPSET(I+LSKIP2,J)
      END IF
    END IF
 25 CONTINUE
    ILEVP(2,LEV) = IND
    WRITE(LU6,*)' ILEVP(2,LEV)=',ILEVP(2,LEV)
 30 CONTINUE
C
C         INTERPOLATION BC POINTER
```

```
C
      WRITE(LUG,*)' *** NOTE: NO SYMMETRY CONDITIONS HERE'
C
      IND = 0
      DO 47 LEV = 2,LMAX
      LEVM1 = LEV-1
      LSKIP = 2**(LEV-1)
      LSKIP2 = LSKIP/2
      IPBIMX(1,LEV-1) = IND+1
      DO 45 I=1,IEM1,LSKIP
      DO 45 J=1,JEM1,LSKIP
      IF(LEVSET(I+LSKIP2,J+LSKIP2).EQ.LEV) THEN
        IF(LEVSET(I,J+LSKIP2).EQ.LEV-1) THEN
          IND = IND+1
          IPBI(1,IND) = IPSET(I,J+LSKIP)
          IPBI(2,IND) = IPSET(I,J+LSKIP2)
          IPBI(3,IND) = IPSET(I,J)
        END IF
        IF(LEVSET(I+LSKIP2,J+LSKIP).EQ.LEVM1) THEN
          IND = IND+1
          IPBI(1,IND) = IPSET(I+LSKIP,J+LSKIP)
          IPBI(2,IND) = IPSET(I+LSKIP2,J+LSKIP)
          IPBI(3,IND) = IPSET(I,J+LSKIP)
        END IF
        IF(LEVSET(I+LSKIP,J+LSKIP2).EQ.LEVM1) THEN
          IND = IND+1
          IPBI(1,IND) = IPSET(I+LSKIP,J)
          IPBI(2,IND) = IPSET(I+LSKIP,J+LSKIP2)
          IPBI(3,IND) = IPSET(I+LSKIP,J+LSKIP)
        END IF
        IF(LEVSET(I+LSKIP2,J).EQ.LEVM1) THEN
          IND = IND+1
          IPBI(1,IND) = IPSET(I,J)
          IPBI(2,IND) = IPSET(I+LSKIP2,J)
          IPBI(3,IND) = IPSET(I+LSKIP,J)
        END IF
      END IF
   45 CONTINUE
      IF(IPBIMX(1,LEV-1).GT.IND) THEN
        IPBIMX(1,LEV-1) = 0
        IPBIMX(2,LEV-1) = 0
      ELSE
        IPBIMX(2,LEV-1) = IND
      END IF
   47 CONTINUE
C
C         FARFIELD & SOLID WALL BC POINTERS
C
C         DEFINE FARFIELD POINTS
      IND = 0
      J = 0
   49 J = J+1
      IF(IPSET(1,J).GT.0) THEN
        IND = IND+1
```

```
        IPBU(2,IND) = IPSET(1,J)
        IF(J.GT.1) THEN
          IPBU(3,IND-1) = IPBU(2,IND)
          IPBU(1,IND)   = IPBU(2,IND-1)
        END IF
      END IF
      IF(J.LT.JE) GO TO 49
C
      I = 1
   50 I = I+1
      IF(IPSET(I,JE).GT.0) THEN
        IND = IND+1
        IPBU(2,IND) = IPSET(I,JE)
        IF(I.GT.2) THEN
          IPBU(3,IND-1) = IPBU(2,IND)
          IPBU(1,IND)   = IPBU(2,IND-1)
        ELSE
          IPBU(1,IND)   = IPBU(2,IND-1)
        END IF
      END IF
      IF(I.LT.IE-1) GO TO 50
C
      J = JE+1
   51 J = J-1
      IF(IPSET(1,J).GT.0) THEN
        IND = IND+1
        IPBU(2,IND) = IPSET(IE,J)
        IF(J.LT.JE) THEN
          IPBU(3,IND-1) = IPBU(2,IND)
          IPBU(1,IND)   = IPBU(2,IND-1)
        ELSE
          IPBU(3,IND-1) = IPBU(2,IND)
        END IF
      END IF
      IF(J.GT.1) GO TO 51
C
      IPBU(1,1) = IPBU(1,IND)
      IPBUMX = IND-1
C
C         DEFINE SOLID WALL POINTS
      IND = 0
      I = ITE
   52 I = I+1
      IF(IPSET(I,1).GT.0) THEN
        IND = IND+1
        IPBB(2,IND) = IPSET(I,1)
        IF(IND.GT.1) THEN
          IPBB(3,IND-1) = IPBB(2,IND)
          IPBB(1,IND)   = IPBB(2,IND-1)
        ELSE
          IPBB(1,IND)   = IPSET(ITE,1)
        END IF
      END IF
      IF(I.LT.IE-ITE+1) GO TO 52
```

```
C
      IPBBMX = IND-1
C
      DO 53 IND=1,IPBUMX
      I1 = IPBU(1,IND)
      I2 = IPBU(2,IND)
      I3 = IPBU(3,IND)
      CALL GEOBCEL(I1,I2,I3,ICEL1,ICEL2)
      IPBU(1,IND) = ICEL1
      IPBU(2,IND) = ICEL2
      IF(ICEL1.NE.0.AND.ICEL2.NE.0) THEN
        IF(ABS(IP(2,ICEL1)).EQ.ABS(IP(1,ICEL2))) THEN
          IPBU(3,IND) = 1
        ELSE IF(ABS(IP(3,ICEL1)).EQ.ABS(IP(2,ICEL2))) THEN
          IPBU(3,IND) = 2
        ELSE IF(ABS(IP(4,ICEL1)).EQ.ABS(IP(3,ICEL2))) THEN
          IPBU(3,IND) = 3
        ELSE IF(ABS(IP(1,ICEL1)).EQ.ABS(IP(4,ICEL2))) THEN
          IPBU(3,IND) = 4
        ELSE
          WRITE(LU6,*)' ** ERROR IN DEFINING IPBU TYPE 1'
        END IF
      ELSE IF(I3.EQ.0) THEN
        IPBU(3,IND) = 6
      ELSE IF(I1.EQ.0) THEN
        IPBU(3,IND) = 7
      ELSE
        WRITE(LU6,*)' ** ERROR IN DEFINING IPBU TYPE 2'
      END IF
      IF(IND.EQ.1) IPBU(3,IND) = 5
   53 CONTINUE
C
      DO 54 IND=1,IPBBMX
      I1 = IPBB(1,IND)
      I2 = IPBB(2,IND)
      I3 = IPBB(3,IND)
      CALL GEOBCEL(I1,I2,I3,ICEL1,ICEL2)
      IPBB(1,IND) = ICEL1
      IPBB(2,IND) = ICEL2
      IPBB(3,IND) = 4
   54 CONTINUE
C
C         OTHER POINTERS
      IPBDMX = 0
      IPBIMX = 0
C
      RETURN
      END
C-----------------------------------------------------------------
C*****************************************************************
C-----------------------------------------------------------------
```

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
C-------------------------------------------------------------------
C*******************************************************************
C-------------------------------------------------------------------
C
C-------------------------------------------------------------------
C          PROGRAM: C-MESH EULERCEL
C-------------------------------------------------------------------
C         PROGRAM EULERCELL SOLVES THE 2-D EULER EQN'S
C         USING A CELL POINTER BASED VERSION OF
C         NI'S METHOD.  IT INCLUDES THE CAPABILITY
C         OF ANY NUMBER OF SUBDOMAINS.
C
C
C         READ INPUT PROPERTIES
      CALL INPUT2
C
C         CALCULATE CONSTANTS CONTAINING GAMMA
      CALL GAMMAS
C
C         INITIALIZE FLOW FIELD TO UNIFORM FLOW
      CALL INITIA
C
C         SOLVE EULER EQN'S USING NI'S METHOD
      CALL NI
C
C         OUTPUT FINAL SOLUTION
      CALL OUTPUT3
C
      STOP
      END
C-------------------------------------------------------------------
C          INCLUDE FILE: GAM.INC
C-------------------------------------------------------------------
      COMMON/GAM/ GAMMA,HTOI,
     1            GM1,GM1D2,GM1DG,GM1D2G,
     2            GP1DG,GP1D2G,GM3
C-------------------------------------------------------------------
C          INCLUDE FILE: INPT.INC
C-------------------------------------------------------------------
      COMMON/INPT/ AMFS,CFL,AVISCF,EXITP,ITIM,
     1             ISTART,NSTART,NMAX,LMAX,
     2             LSTOP,DELSTP,IPRNT1,IPRNT2,
     3             WCFS(4),DELTA,AK,YO,
     4             IE,JE,IC1,IC2,JC2,IF2,JF2,
     5             ALPHA,ROFS,APFS,UFS,VFS,
     6             NFINSH,DELMAX1(5),
     7             ICONST(50),RCONST(50),
     8             INSSWT,RE0,PR,CSTAR,TREF
      COMMON/INPTLAB/GLABEL1,GLABEL2,RLABEL1,RLABEL2
      CHARACTER GLABEL1*30,GLABEL2*100,RLABEL1*10,RLABEL2*100
C-------------------------------------------------------------------
C          INCLUDE FILE: LUNITS.INC
C-------------------------------------------------------------------
```

```
      COMMON/LUNITS/LU1,LU2,LU3,LU4,LU5,LU6,LU7
C-----------------------------------------------------------------
C           INCLUDE FILE: MAIN.INC
C-----------------------------------------------------------------
      COMMON/MAIN/T,DT,DTE(82),DELMAX(5),IMAX,
     1            IQMAX,LEVP(2,5),IPBIMX(2,5),IPBUMX,IPBDMX,
     2            IPBTMX,IPBBMX,Q(10,8424),IP(9,10816),IPBI(3,257),
     3            IPBU(3,257),IPBD(3,9),IPBT(3,33),IPBB(3,257),
     4            QIB(4,257),EDT(8424)
C-----------------------------------------------------------------
C           INCLUDE FILE: MET.INC
C-----------------------------------------------------------------
      COMMON/MET/ DV,DL,DM,DXDXI,DYDXI,DXDET,DYDET,
     1            DXIDX,DXIDY,DETDX,DETDY,AJAC
      COMMON/MET2/ IWP1(258),TX(258),TY(258),
     1             RC(258),TSCL(258),SSCL(258)
C-----------------------------------------------------------------
C           INCLUDE FILE: POINT.INC
C-----------------------------------------------------------------
      COMMON/POINT/ I1,I2,I3,I4,INC,IN1,IN2,IN3,IN4,
     1              VIS1,VIS2,VIS3,VIS4,IVIS
C-----------------------------------------------------------------
C           INCLUDE FILE: SOLV.INC
C-----------------------------------------------------------------
      COMMON/SOLV/ F(4,4),G(4,4),DELU(4),DELF(4),DELG(4)
C-----------------------------------------------------------------
C           SUBROUTINE FILE: BDSMTH
C-----------------------------------------------------------------
      SUBROUTINE BDSMTH(LEV)
C
C         **** SPECIAL C-MESH FORMULATION ****
C         **** SHOULD WORK WITH O-MESH   ****
C
C         This subroutine smooths the far field and solid
C         wall boundary points.  Points are always smoothed
C         on the lowest level in which the two adjoining cell
C         to the boundary exist.  This is consistent with the
C         internal point smoothing.
C             For the far field boundary the smoothing used is
C         the corresponding one model applied along the boundary.
C             For the solid wall boundary two formulations are
C         possible:
C           Type 1:  The same as the farfield boundary with a
C                    ramp increase in smoothing around the t.e.
C           Type 2:  A standard internal smoothing using extrapolated
C                    information to define an imaginary line of points
C                    inside the wall.  In this case the smoothing is not
C                    increased in the t.e. region.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'GAM.INC'
```

```
C
      DIMENSION QN1(4),QN2(4),QAVE2(4)
C
C        Far Field boundary point smoothing using a 1-D smoothing
C        tangent to the boundary
      DO 10 I=1,IPBUMX
C
C        Is point to be smoothed on this level?
      ICONT1 = 0
      IF(IPBU(1,I).GE.LEVP(1,LEV).AND.IPBU(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBU(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBU(2,I).GE.LEVP(1,LEV).AND.IPBU(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBU(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C        If it is then calculate and add contributions
C        for each cell surrounding the point.
      IF(ICONT1.EQ.1) THEN
C
        IF(IPBU(3,I).EQ.1) THEN
          J1 = ABS(IP(1,IPBU(1,I)))
          J2 = ABS(IP(2,IPBU(1,I)))
          J3 = ABS(IP(2,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.2) THEN
          J1 = ABS(IP(2,IPBU(1,I)))
          J2 = ABS(IP(3,IPBU(1,I)))
          J3 = ABS(IP(3,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.3) THEN
          J1 = ABS(IP(3,IPBU(1,I)))
          J2 = ABS(IP(4,IPBU(1,I)))
          J3 = ABS(IP(4,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.4) THEN
          J1 = ABS(IP(4,IPBU(1,I)))
          J2 = ABS(IP(1,IPBU(1,I)))
          J3 = ABS(IP(1,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.5) THEN
          J1 = ABS(IP(3,IPBU(1,I)))
          J2 = ABS(IP(4,IPBU(1,I)))
          J3 = ABS(IP(2,IPBU(2,I)))
        ELSE IF(IPBU(3,I).GT.5) THEN
          GO TO 10
        END IF
C
C         First cell:
        CALL CELPOINT(IPBU(1,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
C
        DO 4 K=1,4
        KP2 = K+2
        KP6 = K+6
    4   Q(KP6,J2) = Q(KP6,J2)+0.125*AVIS*(Q(KP2,J1)-Q(KP2,J2))
```

```
C
C        Second Cell:
         CALL CELPOINT(IPBU(2,I))
         CALL METRC4
         CALL CTIME
         AVIS = AVISCF*DT*(DL+DM)/DV
C
         DO 6 K=1,4
         KP2 = K+2
         KP6 = K+6
    6    Q(KP6,J2) = Q(KP6,J2)+0.125*AVIS*(Q(KP2,J3)-Q(KP2,J2))
       END IF
   10 CONTINUE
C
C         Solid Wall Boundary point Smoothing
C         Possible forms:
C             IBCOND = 1  1-D tangent smoothing model
C                         with ramp increase at t.e.
C                    = 2  For reflected points and standard
C                         internal point smoothing model.
C
C         Constants
       IBCOND = 1
       JTESMTH = 5
       TECOEF  = 4.0
       IF(IBCOND.EQ.2) GO TO 40
C
C          Type 1: 1D smoothing formulation
       DO 30 I=1,IPBBMX
CCC       IF(IPBB(3,I).NE.4) GO TO 30
C
C          Is point to be smoothed on this level?
       ICONT1 = 0
       IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
         IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
       ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
      1        THEN
         IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
       END IF
C
C          If yes, calculate and add contributions form
C          both cells surrounding the cell
       IF(ICONT1.EQ.1) THEN
C
C            First cell:
         CALL CELPOINT(IPBB(1,I))
         CALL METRC4
         CALL CTIME
         AVIS = AVISCF*DT*(DL+DM)/DV
C
C            Ramp smoothing near t.e.
CCC         IF(I.LE.JTESMTH+1) THEN
CCC           AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
CCC         ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
```

```
CCC           AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
CCC     1             /FLOAT(JTESMTH))*AVIS
CCC          END IF
CCC
C
         DO 24 K=1,4
         KP2 = K+2
         KP6 = K+6
CC         IF(IPBB(3,I).EQ.4) THEN
C           Q(KP6,I4) = Q(KP6,I4)+0.125*AVIS*(Q(KP2,I1)-Q(KP2,I4))
           Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(Q(KP2,I1)-Q(KP2,I4))
CC         ELSE IF(IPBB(3,I).EQ.5) THEN
CC           Q(KP6,I4) = Q(KP6,I4)+0.0625*AVIS*(Q(KP2,I1)-Q(KP2,I4))
CC     1                           +0.0625*AVIS*(Q(KP2,I3)-Q(KP2,I4))
CC         END IF
   24    CONTINUE
C
C            Second cell:
         CALL CELPOINT(IPBB(2,I))
         CALL METRC4
         CALL CTIME
         AVIS = AVISCF*DT*(DL+DM)/DV
C
C             Ramp soothing near t.e.
CCC         IF(I.LE.JTESMTH+1) THEN
CCC           AVIS = (1.0+TECOEF*FLOAT(JTESMTH-I+1)/FLOAT(JTESMTH))*AVIS
CCC         ELSE IF(I.GE.IPBBMX-JTESMTH+1) THEN
CCC           AVIS = (1.0+TECOEF*FLOAT(I+JTESMTH-IPBBMX-1)
CCC     1             /FLOAT(JTESMTH))*AVIS
CCC         END IF
C
         DO 26 K=1,4
         KP2 = K+2
         KP6 = K+6
CC         IF(IPBB(3,I).EQ.4) THEN
C           Q(KP6,I1) = Q(KP6,I1)+0.125*AVIS*(Q(KP2,I4)-Q(KP2,I1))
           Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(Q(KP2,I4)-Q(KP2,I1))
CC         ELSE IF(IPBB(3,I).EQ.5) THEN
CC           Q(KP6,I1) = Q(KP6,I1)+0.0625*AVIS*(Q(KP2,I4)-Q(KP2,I1))
CC     1                           +0.0625*AVIS*(Q(KP2,I2)-Q(KP2,I1))
CC         END IF
   26    CONTINUE
       END IF
   30 CONTINUE
C
       RETURN
C
C          Type 2:  Reflection wall smoothing
   40 CONTINUE
       DO 70 I=1,IPBBMX
       IF(IPBB(3,I).NE.4) GO TO 70
C
C          Is this point to be smoothed on this level?
       ICONT1 = 0
```

```
      IF(IPBB(1,I).GE.LEVP(1,LEV).AND.IPBB(1,I).LE.LEVP(2,LEV))THEN
        IF(IPBB(2,I).LE.LEVP(2,LEV)) ICONT1 = 1
      ELSE IF(IPBB(2,I).GE.LEVP(1,LEV).AND.IPBB(2,I).LE.LEVP(2,LEV))
     1       THEN
        IF(IPBB(1,I).LE.LEVP(2,LEV)) ICONT1 = 1
      END IF
C
C         If Yes, calculate and add contributions form both cells
      IF(ICONT1.EQ.1) THEN
C
C           Calculate surface tangent vector (dx,dy)
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
C
C           First cell:
        CALL CELPOINT(IPBB(1,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
CCC
CCC           Find surface tangent
CC        DX = Q(1,I4)-Q(1,I1)
CC        DY = Q(2,I4)-Q(2,I1)
CC        DS = SQRT(DX*DX+DY*DY)
CC        DX = DX/DS
CC        DY = DY/DS
C
C           Extrapolate r,p,h0 and reflect u,v
        P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
        P3 = GM1*(Q(6,I3)-0.5*(Q(4,I3)*Q(4,I3)+Q(5,I3)*Q(5,I3))/Q(3,I3))
        PI4 = P3-2.*(P3-P4)
        H4 = (Q(6,I4)+P4)/Q(3,I4)
        H3 = (Q(6,I3)+P3)/Q(3,I3)
CCC        HI4 = H3-2.*(H3-H4)
```

```
        HI4 = H3
        VELT3 = (Q(4,I3)*DX+Q(5,I3)*DY)/Q(3,I3)
        VELN3 = (-Q(4,I3)*DY+Q(5,I3)*DX)/Q(3,I3)
        UI4 = VELT3*DX+VELN3*DY
        VI4 = VELT3*DY-VELN3*DX
CCC        RI4 = Q(3,I3)-2.*(Q(3,I3)-Q(3,I4))
CCC        EI4 = PI4/GM1+0.5*RI4*(UI4*UI4+VI4*VI4)
        RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
        EI4 = RI4*HI4-PI4
        P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
        P2 = GM1*(Q(6,I2)-0.5*(Q(4,I2)*Q(4,I2)+Q(5,I2)*Q(5,I2))/Q(3,I2))
        PI1 = P2-2.*(P2-P1)
        H1 = (Q(6,I1)+P1)/Q(3,I1)
        H2 = (Q(6,I2)+P2)/Q(3,I2)
CCC        HI1 = H2-2.*(H2-H1)
        HI1 = H2
        VELT2 = (Q(4,I2)*DX+Q(5,I2)*DY)/Q(3,I2)
        VELN2 = (-Q(4,I2)*DY+Q(5,I2)*DX)/Q(3,I2)
        UI1 = VELT2*DX+VELN2*DY
        VI1 = VELT2*DY-VELN2*DX
CCC        RI1 = Q(3,I2)-2.*(Q(3,I2)-Q(3,I1))
CCC        EI1 = PI1/GM1+0.5*RI1*(UI1*UI1+VI1*VI1)
        RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
        EI1 = RI1*HI1-PI1
C
C           Find reflected cell center values
        QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
        QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
        QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
        QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C           Add contribution
        DO 64 K=1,4
        KP2 = K+2
        KP6 = K+6
        QAVE1 = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
C
        Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I4))
   64   CONTINUE
C
C           Second Cell:
        CALL CELPOINT(IPBB(2,I))
        CALL METRC4
        CALL CTIME
        AVIS = AVISCF*DT*(DL+DM)/DV
CCC
CCC           Find surface tangent
CC        DX = Q(1,I4)-Q(1,I1)
CC        DY = Q(2,I4)-Q(2,I1)
CC        DS = SQRT(DX*DX+DY*DY)
CC        DX = DX/DS
CC        DY = DY/DS
C
C           Extrapolate r,p,h0 and reflect u,v
```

```
      P4 = GM1*(Q(6,I4)-0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))
      P3 = GM1*(Q(6,I3)-0.5*(Q(4,I3)*Q(4,I3)+Q(5,I3)*Q(5,I3))/Q(3,I3))
      PI4 = P3-2.*(P3-P4)
      H4 = (Q(6,I4)+P4)/Q(3,I4)
      H3 = (Q(6,I3)+P3)/Q(3,I3)
CCC      HI4 = H3-2.*(H3-H4)
      HI4 = H3
      VELT3 = (Q(4,I3)*DX+Q(5,I3)*DY)/Q(3,I3)
      VELN3 = (-Q(4,I3)*DY+Q(5,I3)*DX)/Q(3,I3)
      UI4 = VELT3*DX+VELN3*DY
      VI4 = VELT3*DY-VELN3*DX
CCC      RI4 = Q(3,I3)-2.*(Q(3,I3)-Q(3,I4))
CCC      EI4 = PI4/GM1+0.5*RI4*(UI4*UI4+VI4*VI4)
      RI4 = PI4/(GM1DG*(HI4-0.5*(UI4*UI4+VI4*VI4)))
      EI4 = RI4*HI4-PI4
      P1 = GM1*(Q(6,I1)-0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))
      P2 = GM1*(Q(6,I2)-0.5*(Q(4,I2)*Q(4,I2)+Q(5,I2)*Q(5,I2))/Q(3,I2))
      PI1 = P2-2.*(P2-P1)
      H1 = (Q(6,I1)+P1)/Q(3,I1)
      H2 = (Q(6,I2)+P2)/Q(3,I2)
CCC      HI1 = H2-2.*(H2-H1)
      HI1 = H2
      VELT2 = (Q(4,I2)*DX+Q(5,I2)*DY)/Q(3,I2)
      VELN2 = (-Q(4,I2)*DY+Q(5,I2)*DX)/Q(3,I2)
      UI1 = VELT2*DX+VELN2*DY
      VI1 = VELT2*DY-VELN2*DX
CCC      RI1 = Q(3,I2)-2.*(Q(3,I2)-Q(3,I1))
CCC      EI1 = PI1/GM1+0.5*RI1*(UI1*UI1+VI1*VI1)
      RI1 = PI1/(GM1DG*(HI1-0.5*(UI1*UI1+VI1*VI1)))
      EI1 = RI1*HI1-PI1
C
C        Find reflected cell center values
      QAVE2(1) = 0.25*(Q(3,I1)+Q(3,I4)+RI1+RI4)
      QAVE2(2) = 0.25*(Q(4,I1)+Q(4,I4)+RI1*UI1+RI4*UI4)
      QAVE2(3) = 0.25*(Q(5,I1)+Q(5,I4)+RI1*VI1+RI4*VI4)
      QAVE2(4) = 0.25*(Q(6,I1)+Q(6,I4)+EI1+EI4)
C
C        Add contribution
      DO 66 K=1,4
      KP2 = K+2
      KP6 = K+6
      QAVE1 = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
C
      Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(QAVE1+QAVE2(K)-2.*Q(KP2,I1))
   66 CONTINUE
     END IF
   70 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C        SUBROUTINE FILE: DELTRS
C-----------------------------------------------------------------
      SUBROUTINE DELTRS
```

```
C         This subroutine calculates the Navier-Stokes
C         viscous terms in a manor similar to the artifical
C         viscousity and adds them to the DU's.
      INCLUDE 'MAIN.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
C
      DIMENSION QAVE(4),DR(4),DS(4),DQDXI(4),DQDET(4),
     1          DELR(4),DELS(4)
C
C         Calculate average properties for cell center
      DO 5 K=1,4
      KP2 = K+2
    5 QAVE(K) = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
      RAVE = QAVE(1)
      UAVE = QAVE(2)/QAVE(1)
      VAVE = QAVE(3)/QAVE(1)
      TAVE = GAMMA*GM1*(QAVE(4)
     1    -0.5*(QAVE(2)*QAVE(2)+QAVE(3)*QAVE(3))/QAVE(1))/QAVE(1)
      AMUE = (TAVE**(3./2.))*(1.+CSTAR)/(TAVE+CSTAR)
C
C         Calculate cell center gradients in coordinate directions
      DO 10 K=1,3
      KP2 = K+2
      DQDXI(K) = 0.5*(Q(KP2,I3)+Q(KP2,I4)-Q(KP2,I1)-Q(KP2,I2))
   10 DQDET(K) = 0.5*(Q(KP2,I2)+Q(KP2,I3)-Q(KP2,I1)-Q(KP2,I4))
C
      TI1 = GAMMA*GM1*(Q(6,I1)
     1    -0.5*(Q(4,I1)*Q(4,I1)+Q(5,I1)*Q(5,I1))/Q(3,I1))/Q(3,I1)
      TI2 = GAMMA*GM1*(Q(6,I2)
     1    -0.5*(Q(4,I2)*Q(4,I2)+Q(5,I2)*Q(5,I2))/Q(3,I2))/Q(3,I2)
      TI3 = GAMMA*GM1*(Q(6,I3)
     1    -0.5*(Q(4,I3)*Q(4,I3)+Q(5,I3)*Q(5,I3))/Q(3,I3))/Q(3,I3)
      TI4 = GAMMA*GM1*(Q(6,I4)
     1    -0.5*(Q(4,I4)*Q(4,I4)+Q(5,I4)*Q(5,I4))/Q(3,I4))/Q(3,I4)
      DTDXI = 0.5*(TI3+TI4-TI1-TI2)
      DTDET = 0.5*(TI2+TI3-TI1-TI4)
C
C         define constants
      C1 = 4./3.
      C2 = 2./3.
      C3 = -AMUE/(RAVE*DV*RE0)
      C4 = -AMUE/(GM1*PR*DV*RE0)
C
C         Calculate stress components
      DUDX = DYDET*DQDXI(2)-DYDXI*DQDET(2)
     1      -UAVE*(DYDET*DQDXI(1)-DYDXI*DQDET(1))
      DVDY =-DXDET*DQDXI(3)+DXDXI*DQDET(3)
     1      -VAVE*(-DXDET*DQDXI(1)+DXDXI*DQDET(1))
      TXX = C3*(C1*DUDX-C2*DVDY)
      TYY = C3*(C1*DVDY-C2*DUDX)
      TXY = C3*(-DXDET*DQDXI(2)+DXDXI*DQDET(2)
```

```
     1          +DYDET*DQDXI(3)-DYDXI*DQDET(3)
     2          +(UAVE*DXDET-VAVE*DYDET)*DQDXI(1)
     3          -(UAVE*DXDXI-VAVE*DYDXI)*DQDET(1))
C
C         Calculate DR(k) and DS(k)
      DR(2) = TXX
      DR(3) = TXY
      DR(4) = UAVE*TXX+VAVE*TXY
     1       +C4*(DYDET*DTDXI-DYDXI*DTDET)
      DS(2) = TXY
      DS(3) = TYY
      DS(4) = VAVE*TYY+UAVE*TXY
     1       +C4*(-DXDET*DTDXI+DXDXI*DTDET)
C
C         Calculate DELR(k) and DELS(k)
      DT2DV = 2.0*DT/DV
      DO 15 K=2,4
      DELR(K) = (DYDET*DR(K)-DXDET*DS(K))*DT2DV
      DELS(K) = (DXDXI*DS(K)-DYDXI*DR(K))*DT2DV
   15 CONTINUE
C
C        Distribute to cell corner nodes
      DO 20 K=2,4
      KP6 = K+6
      Q(KP6,I1) = Q(KP6,I1)+0.25*(-DELR(K)-DELS(K))*VIS1
      Q(KP6,I2) = Q(KP6,I2)+0.25*(-DELR(K)+DELS(K))*VIS2
      Q(KP6,I3) = Q(KP6,I3)+0.25*(DELR(K)+DELS(K))*VIS3
      Q(KP6,I4) = Q(KP6,I4)+0.25*(DELR(K)-DELS(K))*VIS4
   20 CONTINUE
C
CCC
CCC      WRITE(1,*)' I1,DELR,DELS(2)',I1,DELR(2),DELS(2)
CCC
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE FILE: FARFDBC2
C-----------------------------------------------------------------
      SUBROUTINE FARFDBC2
C
C         **   SPECIAL C-MESH FORMULATION    **
C         This subroutine calculates the far field boundary
C         conditions using a local characteristic analysis
C         tangent and normal to the boundary.  Both uniform
C         freestream or far field vortex boundary conditions
C         are possible.  The selection is made by setting the
C         following switch:
C           IFDTYPE = 0  for uniform freestreem conditions
C                     1  for vortex farfield conditions with
C                        the strength of the point vortex based
C                        on an integration of surface pressure
C                        to set the lift.
C
C         Note: for supersonic flows the uniform freestream
```

```
C               flow condition is automatically set since this
C               boundary condition is only correct for subsonic
C               flows.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'LUNITS.INC'
      DIMENSION UBAR(4)
C
C         Constants
      IFDTYPE = 1
      PHI   = 3.141592654
C
C         Calculate Lift Force Coefficients through an
C         integration of the surface pressures of airfoil.
      IF(IFDTYPE.EQ.1) THEN
C
        CHORD = 0.0
        CFN  = 0.0
        CFT  = 0.0
C
        DO 5 I=0,IPBBMX
        IF(I.NE.0) THEN
          J1 = ABS(IP(1,IPBB(2,I)))
          J2 = ABS(IP(4,IPBB(2,I)))
        ELSE
          J1 = ABS(IP(1,IPBB(1,1)))
          J2 = ABS(IP(4,IPBB(1,1)))
        END IF
C
        IF(I.EQ.0) THEN
          TX1 = Q(1,J1)
          TY1 = Q(2,J1)
        END IF
        TCHORD = (TX1-Q(1,J2))**2+(TY1-Q(2,J2))**2
        IF(TCHORD.GT.CHORD) CHORD = TCHORD
C
        DX = Q(1,J2)-Q(1,J1)
        DY = Q(2,J2)-Q(2,J1)
        DS = SQRT(DX*DX+DY*DY)
C
        P1 = GM1*(Q(6,J1)
     1      -0.5*(Q(4,J1)*Q(4,J1)+Q(5,J1)*Q(5,J1))/Q(3,J1))
        P2 = GM1*(Q(6,J2)
     1      -0.5*(Q(4,J2)*Q(4,J2)+Q(5,J2)*Q(5,J2))/Q(3,J2))
        TMP = P1+P2
        CFN = CFN+TMP*DX
        CFT = CFT+TMP*DY
    5   CONTINUE
C
        CHORD = SQRT(CHORD)
        QFS = ROFS*(UFS*UFS+VFS*VFS)*CHORD
        CFN = -CFN/QFS
```

```
        CFT = CFT/QFS
C
        ALPHAR = 3.14159*ALPHA/180.0
        CL = CFN*COS(ALPHAR)-CFT*SIN(ALPHAR)
C
C         SET AIRFOIL CENTER AT 1/4 CHORD
        XQC = TX1-0.75*CHORD
        YQC = TY1
      ELSE
        CL = 0.0
      END IF
C
C       Sweep around farfield boundary and correct DU's
C       using Characteristic analysis tangent and normal
C       to the boundary.
C
      DO 10 I=1,IPBUMX
      IF(IPBU(3,I).EQ.1) THEN
        J1 = ABS(IP(2,IPBU(1,I)))
        J2 = ABS(IP(3,IPBU(1,I)))
      ELSE IF(IPBU(3,I).EQ.2) THEN
        J1 = ABS(IP(3,IPBU(1,I)))
        J2 = ABS(IP(4,IPBU(1,I)))
      ELSE IF(IPBU(3,I).EQ.3) THEN
        J1 = ABS(IP(4,IPBU(1,I)))
        J2 = ABS(IP(1,IPBU(1,I)))
      ELSE IF(IPBU(3,I).EQ.4) THEN
        J1 = ABS(IP(1,IPBU(1,I)))
        J2 = ABS(IP(2,IPBU(1,I)))
      ELSE IF(IPBU(3,I).EQ.5) THEN
        J1 = ABS(IP(4,IPBU(1,I)))
        J2 = ABS(IP(1,IPBU(1,I)))
      ELSE IF(IPBU(3,I).EQ.6) THEN
        J1 = ABS(IP(2,IPBU(1,I)))
        J2 = ABS(IP(3,IPBU(1,I)))
      ELSE IF(IPBU(3,I).EQ.7) THEN
        J1 = ABS(IP(3,IPBU(2,I)))
        J2 = ABS(IP(2,IPBU(2,I)))
      ELSE
        WRITE(LU1,*)' ERROR IN FARFDBC2 IPBU(3,I) NOT 1-6'
      END IF
C
C         Calculate boundary normal vector
C         Note: Present analysis assumes eta lines run
C               Normal to the far field boundary.
      TMP1 = Q(1,J2)-Q(1,J1)
      TMP2 = Q(2,J2)-Q(2,J1)
      TMP3 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
      DX = TMP1/TMP3
      DY = TMP2/TMP3
C
C         Calculate local radius and direction
      TMP1 = XQC-Q(1,J1)
```

```
      TMP2 = YQC-Q(2,J1)
      RAD  = SQRT(TMP1**2+TMP2**2)
      DRX  = TMP1/RAD
      DRY  = TMP2/RAD
C
C         Calculate extrapolated quantities from
C         the predicted values of Q at the boundary.
      REX   = Q(3,J1)+Q(7,J1)
      UEX   = (Q(4,J1)+Q(8,J1))/REX
      VEX   = (Q(5,J1)+Q(9,J1))/REX
      EEX   = Q(6,J1)+Q(10,J1)
      QSQEX =UEX*UEX+VEX*VEX
      PEX   =GM1*(EEX-0.5*REX*QSQEX)
      IF(PEX.LE.0.0) WRITE(LU1,*)'** PEX<0 AT UP I=',I
      AEX   =SQRT(GAMMA*PEX/REX)
C
      QNEX  = UEX*DX+VEX*DY
      QTEX  =-UEX*DY+VEX*DX
C
C         Set barred or frozen quantities of linearization
C         based on the extrapolated conditions.
      RBAR = REX
      ABAR = AEX
C
C         Calculate the free stream conditions without
C         the vortex.
      QNFS = UFS*DX+VFS*DY
      QTFS =-UFS*DY+VFS*DX
      QFS  = SQRT(QNFS**2+QTFS**2)
C
C         Set far field conditions to either free stream
C         or calculate and set to vortex farfield conditions
C
C         Set vortex farfield condition
      IF(IFDTYPE.EQ.1.AND.AMFS.LE.1) THEN
        COSFD = (UFS*DRX+VFS*DRY)/QFS
        SINFD = (-UFS*DRY+VFS*DRX)/QFS
        BETA  = SQRT(1.0-AMFS*AMFS)
        TMP1  = 1.0/(COSFD**2+BETA*BETA*SINFD*SINFD)
        DQVORT = QFS*CHORD*CL*BETA*TMP1/(4.0*PHI*RAD)
        QNFD  = QNFS+DQVORT*(-DRY*DX+DRX*DY)
        QTFD  = QTFS+DQVORT*(DRY*DY+DRX*DX)
        QFD   = SQRT(QNFD**2+QTFD**2)
        PFD   = (APFS**GM1DG+GM1D2G*ROFS*(QFS**2-QFD**2)
     1           /(APFS**(1.0/GAMMA)))**(GAMMA/GM1)
        ROFD  = ROFS*((PFD/APFS)**(1.0/GAMMA))
C
C           Otherwise set farfield conditions to freestream
      ELSE
        QNFD  = QNFS
        QTFD  = QTFS
        PFD   = APFS
        ROFD  = ROFS
      END IF
```

```
C
C         Calculate corrected farfield flow conditions
C         based on whether it is supersonic or subsonic
C         and inflow or outflow
C
C           Subsonic inflow
      IF(QNEX.GE.0.0.AND.QNEX.LE.ABAR) THEN
        PNEW = 0.5*(PFD+PEX+RBAR*ABAR*(QNFD-QNEX))
        QTNEW = QTFD
        QNNEW = QNFD+(PFD-PNEW)/(RBAR*ABAR)
        RNEW  = ROFD+(PNEW-PFD)/(ABAR*ABAR)
C
C           Subsonic outflow
C           note: sets the downstream characteristic
      ELSE IF(QNEX.GE.-ABAR.AND.QNEX.LT.0.0) THEN
        PNEW  = 0.5*(PFD+PEX+RBAR*ABAR*(QNFD-QNEX))
CCC        PNEW  = PFD
        QTNEW = QTEX
        QNNEW = QNEX+(PNEW-PEX)/(RBAR*ABAR)
        RNEW  = REX+(PNEW-PEX)/(ABAR*ABAR)
C
C           Supersonic inflow
      ELSE IF(QNEX.GT.ABAR) THEN
        PNEW  = PFD
        QTNEW = QTFD
        QNNEW = QNFD
        RNEW  = ROFD
C
C           Supersonic outflow
      ELSE IF(QNEX.LT.-ABAR) THEN
        PNEW  = PEX
        QTNEW = QTEX
        QNNEW = QNEX
        RNEW  = REX
      END IF
C
      ENEW = PNEW/GM1+0.5*RNEW*(QNNEW*QNNEW+QTNEW*QTNEW)
C
C         Calculate corrected DQ's
      Q(7,J1)  = RNEW-Q(3,J1)
      Q(8,J1)  = RNEW*(QNNEW*DX-QTNEW*DY)-Q(4,J1)
      Q(9,J1)  = RNEW*(QNNEW*DY+QTNEW*DX)-Q(5,J1)
      Q(10,J1) = ENEW-Q(6,J1)
C
   10 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE FILE: INITIA
C-----------------------------------------------------------------
      SUBROUTINE INITIA
C
C         This subroutine calculates the freestream quantities
```

```
C         and if ISTART=1 initializes the flow field to uniform
C         flow based on ALPHA and AMFS using isentropic relations.
C         If uniform flow is set the solid wall points are corrected
C         for a zero flux through the boundary by holding the pressure
C         and energy constant and rotating the velocity vector to
C         the local wall tangent.   Note if ISTART=0 the flow is
C         left as read in the INPUT subroutine.
C
C           IF ISTART = 0   THEN UNIFORM FLOW
C                       1   THEN RESTART
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'LUNITS.INC'
      DIMENSION FSU(4)
C
C         Calculate surface tangent, radius of curvature and
C         extrapolation scalings
CCCC      CALL GEOWAL
      WRITE(LU1,*)' GEOWAL COMMENTED OUT'
C
C         CALCULATE FREE STREAM VECTOR U
      ALPHAR = ALPHA*3.14159/180.0
      TMP  = 1.0+GM1D2*AMFS*AMFS
      ROFS = TMP**(-1./GM1)
      APFS = (TMP**(-1./GM1DG))/GAMMA
      UFS  = AMFS*COS(ALPHAR)/SQRT(TMP)
      VFS  = AMFS*SIN(ALPHAR)/SQRT(TMP)
      AFS  = 1.0/SQRT(TMP)
C
CCC      EXITP  = APFS
CCCC
CCC      WCFS(1) = ROFS-APFS/(AFS*AFS)
CCC      WCFS(2) = -VFS
CCC      WCFS(3) = (UFS+APFS/(ROFS*AFS))/SQRT(2.0)
CCC      WCFS(4) = (-UFS+APFS/(ROFS*AFS))/SQRT(2.0)
C
      FSU(1) = ROFS
      FSU(2) = ROFS*UFS
      FSU(3) = ROFS*VFS
      FSU(4) = APFS/GM1+ROFS*(UFS*UFS+VFS*VFS)/2.0
C
C         RETURN IF RESTART
      IF(ISTART.EQ.1) RETURN
C
C         INITIALIZE FLOW FIELD TO FREE STREAM
      DO 1  I = 1,IQMAX
      DO 1  K = 1,4
    1 Q(K+2,I) = FSU(K)
C
C         CORRECT WALL PROPERTIES
      DO 2 I=1,IPBBMX
C
```

```
C         SET POINTERS & CALCULATE WALL TANGENT
      IF(IPBB(3,I).EQ.4) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
      ELSE IF(IPBB(3,I).EQ.5) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
C
        THETA1 = ATAN2((Q(2,J2)-Q(2,J1)),(Q(1,J2)-Q(1,J1)))
        THETA2 = ATAN2((Q(2,J2)-Q(2,J3)),(Q(1,J2)-Q(1,J3)))
        THETA  = 0.5*(THETA1+THETA2)
C
        DX = COS(THETA)
        DY = SIN(THETA)
      ELSE
        WRITE(LU1,*)' ERROR INITIA IPBB(3,I) NOT 4'
      END IF
C
      QFS = SQRT(UFS*UFS+VFS*VFS)
      SIGN = (UFS*DX+VFS*DY)
      SIGN = SIGN/ABS(SIGN)
      TU = SIGN*QFS*DX
      TV = SIGN*QFS*DY
      TR = APFS/(GM1DG*(HTOT-0.5*(TU*TU+TV*TV)))
      TE = TR*HTOT-APFS
C
C         IF EULER CALCULATION (INSSWT=0) MAKE FLOW TANGENT
      IF(INSSWT.EQ.0) THEN
CC        Q(3,J2)   = TR
CC        Q(4,J2) = TR*TU
CC        Q(5,J2) = TR*TV
CC        Q(6,J2) = TE
CCC
```

```
C
C         IF NAVIER-STOKES CALCULATION (INSSWT=1) SET ZERO FLOW
      ELSE IF(INSSWT.EQ.1) THEN
        Q(3,J2) = GAMMA*APFS/(AFS**2)
        Q(4,J2) = 0.0
        Q(5,J2) = 0.0
        Q(6,J2) = Q(3,J2)*HTOT-APFS
      END IF
    2 CONTINUE
C
C         OUTPUT OF INITIAL FLOW
      WRITE(LU1,1000)
      WRITE(LU1,1004) ROFS,UFS,VFS,APFS
      IF (IPRNT2.EQ.0) RETURN
      WRITE(LU1,*) ' INITIAL Q VALUES'
      DO 50 K=1,6
   50 WRITE(LU1,1001) (Q(K,I), I=1,IQMAX)
C
 1000 FORMAT(///,10X,'INITIAL FLOW FIELD  U1/U2/U3/U4',/)
 1001 FORMAT(1X,(10E12.4))
 1004 FORMAT(1X,'ROFS,UFS,VFS,APFS=',4E12.4,/)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE FILE: NISTEP5
C-----------------------------------------------------------------
      SUBROUTINE NISTEP5(N,LEV)
C
C         **** SPECIAL C-MESH FORMULATION ****
C         **** SHOULD STILL WORK WITH O-MESH***
C
C         This subroutine solves the Euler eqns.
C         using a cell oreinted version of Ni's Method
C         over grid level LEV.  This subroutine as written
C         performs either a fine mesh cell distribution or
C         a coarse mesh cell acceleration distribution depending
C         of the type of each cell.
C             In addition this particular version saves a
C         representative dt for each node in EDT(i) for use
C         in the error norm calculation.   This same time step
C         then acts as a indicator as to wether the node is to
C         be updated (i.e. if EDT(i)=0.0 then the node has not
C         been distributed to or interpolated to and therefore
C         should not be updated).
C             This subroutine contains a switch which will include
C         the Navier-Stokes terms on level 1 based on the following
C         switch:
C            INSSWT = 0  For Euler solver.
C                     1  For Navier-Stokes terms on level 1.
C                        Note: In this case no smoothing is applied
C                              on level 1.
C
      INCLUDE 'MAIN.INC'
```

```
      INCLUDE 'SOLV.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'MET.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'LUNITS.INC'
C
C         Inject changes from the next finer level based on
C         one of the following weighting formulae:
C             IFORM = 0    FOR SIMPLE INJECTION OF VALUE AT INC
C                     1    ALGEBRAIC WEIGHTING
C                     2    AREA WEIGHTING
C                     3    NOTHING DONE AT THIS TIME
C                     4    SPECIAL DISTRIBUTION INJECTION
C                     5    NOTHING DONE AT THIS TIME
C                     6    Ni's Distribution (modified form 4)
      IF(LEV.GT.1) CALL INJECT5(LEV,6)
C
C         Initialize DU and EDT before sweep
C           If LEV = 1 then all DU's and EDT's zeroed,
      IF(LEV.EQ.1) THEN
        DO 5 I=1,IQMAX
        EDT(I) = 0.0
        DO 5 K=7,10
    5   Q(K,I) = 0.0
C
C           Otherwize zero Du and EDT only at cell nodes.
      ELSE
        DO 7 I=LEVP(1,LEV),LEVP(2,LEV)
        DO 7 J=1,4
        JP = ABS(IP(J,I))
        EDT(JP) = 0.0
        DO 7 K=7,10
    7   Q(K,JP) = 0.0
C
C           In addition zero boundary du's so application
C           of boundary conditions on coarser levels will
C           only make changes at coarse nodes.
        DO 8 I=1,IPBBMX
        JP = ABS(IP(1,IPBB(2,I)))
        DO 8 K=7,10
    8   Q(K,JP) = 0.0
C
        DO 9 I=1,IPBUMX
        IF(IPBU(3,I).EQ.1) THEN
          JP = ABS(IP(1,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.2) THEN
          JP = ABS(IP(2,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.3) THEN
          JP = ABS(IP(3,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.4) THEN
          JP = ABS(IP(4,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.5) THEN
          JP = ABS(IP(1,IPBU(2,I)))
        ELSE IF(IPBU(3,I).EQ.6) THEN
```

```
          JP = ABS(IP(2,IPBU(1,I)))
        ELSE IF(IPBU(3,I).EQ.7) THEN
          JP = ABS(IP(3,IPBU(2,I)))
        END IF
        DO 9 K=7,10
    9   Q(K,JP) = 0.0
CCC
      END IF
C
C         Initialize embedded mesh interface nodes from
C         coarser mesh.  This  subroutine may also be used
C         to initalize interface DU's with embedded mesh
C         interface corrections.
      CALL INBC4(LEV)
C
C         If global time step is used calculate DT here
C         based on minimum DT for current level.
      IF(ITIM.EQ.0)  CALL GTIME(LEV)
C
C         Initialize error norms to zero.
      DO 10 K=1,5
   10 DELMAX(K) = 0.0
      DELUMAX = 0.0
C
C         Start of relaxation sweep for DU
C         over current level.
      DO 30 I = LEVP(1,LEV),LEVP(2,LEV)
C
C         Setup node pointers for cell.
      CALL CELPOINT(I)
C
C         Calculate cell metrics, volume, and other distances
      CALL METRC4
C
C         Calculate time step for local CFL calculations
C         based on current cell.
      IF(ITIM.EQ.2) CALL CTIME
C
C         Store cell DT in EDT(i) for residual calculations
C         Note: set in this way the final value of EDT is
C               the value of the last cell to be calculated
C               which contains this node.  It is note an
C               average.
      EDT(I1) = DT
      EDT(I2) = DT
      EDT(I3) = DT
      EDT(I4) = DT
C
C         Perform flux balance on cell for DELU(k)
C         then calculate distribution weightings
C         DELF and DELG for cell center.
C          Note: If INC = 0 this is a coarse cell
C                and injection is used.
      CALL DELTU
```

```
      CALL DELTFG
C
C         If level 1 is Navier-Stokes region calculate terms
      IF(INSSWT.EQ.1.AND.LEV.EQ.1) CALL DELTRS
C
C         Calculate artifical viscosity coefficient
C         if any of the cell nodes is to be smoothed.
      IF(IVIS.GT.0) THEN
        AVIS = AVISCF*DT*(DL+DM)/DV
      END IF
C
C         Distribute cell changes to nodes and if
C         the node is to be smoothed then add smoothing.
      DO 20 K=1,4
      KP6 = K+6
C
C              Distribution step
      Q(KP6,I1) = Q(KP6,I1)+(DELU(K)-DELF(K)-DELG(K))/4.0
      Q(KP6,I2) = Q(KP6,I2)+(DELU(K)-DELF(K)+DELG(K))/4.0
      Q(KP6,I3) = Q(KP6,I3)+(DELU(K)+DELF(K)+DELG(K))/4.0
      Q(KP6,I4) = Q(KP6,I4)+(DELU(K)+DELF(K)-DELG(K))/4.0
C
C              Smoothing step
      IF(INSSWT.EQ.1.AND.LEV.EQ.1) GO TO 20
      IF(IVIS.EQ.0) GO TO 20
      KP2 = K+2
C
      QBAR = 0.25*(Q(KP2,I1)+Q(KP2,I2)+Q(KP2,I3)+Q(KP2,I4))
      Q(KP6,I1) = Q(KP6,I1)+0.25*AVIS*(QBAR-Q(KP2,I1))*VIS1
      Q(KP6,I2) = Q(KP6,I2)+0.25*AVIS*(QBAR-Q(KP2,I2))*VIS2
      Q(KP6,I3) = Q(KP6,I3)+0.25*AVIS*(QBAR-Q(KP2,I3))*VIS3
      Q(KP6,I4) = Q(KP6,I4)+0.25*AVIS*(QBAR-Q(KP2,I4))*VIS4
C
   20 CONTINUE
C
C         Calculate Maximum cell RU residual
C         and its cell location.
      IF(DELUMAX.LT.DELU(2)/DT) THEN
        DELUMAX = DELU(2)/DT
        JMAX = I
      END IF
C
   30 CONTINUE
C
C         Zero embedded mesh interface points and
C         calculate interface corrections to be add
C         to interface points on the next coarser
C         level.
      CALL INFACBC2(LEV)
CCC      IF(LEV.GT.1) CALL INFACBC(LEV)
C
C         Double solid wall boundary DU's.
      CALL WALLDBL
C
```

```
C         Correct smoothing at all boundary points
C         (i.e. solid wall and farfield points at
C         this time.).
      CALL BDSMTH(LEV)
C
C         Interpolate DU's from current level to
C         the finest level in each mesh region.
C            IFORM = 1 For centered interpolation (i.e. algebraic)
C                    2 For interpolation based on physical lengths
      IF(LEV.NE.1) CALL INTERPT(LEV,1)
C
C         Apply boundary conditions to all
C         boundary points.
      IF(INSSWT.EQ.0) THEN
        CALL SDWALBC
CC          CALL EULERWAL(LEV)
      ELSE IF(INSSWT.EQ.1) THEN
        CALL NSSDWAL
      END IF
      CALL FARFDBC2
C
C         Update solution for all points
C         that have been changed and calculate
C         node error norms.
      NUMPTS = 0
      DO 60 I = 1,IQMAX
      IF(EDT(I).EQ.0.0) GO TO 60
      NUMPTS = NUMPTS+1
      DO 55 K = 1,4
      KP6 = K+6
      KP2 = K+2
      IF (DELMAX(K).LT.ABS(Q(KP6,I)/EDT(I))) THEN
         DELMAX(K) = ABS(Q(KP6,I)/EDT(I))
         IF (K.EQ.2) THEN
            IMAX = I
         END IF
      END IF
   55 Q(KP2,I) = Q(KP2,I)+Q(KP6,I)
      DELMAX(5) = DELMAX(5)+ABS(Q(8,I)/EDT(I))
   60 CONTINUE
C
      DELMAX(5) = DELMAX(5)/FLOAT(NUMPTS)
C
C         Write out error norms to plot file if
C         LEV is less than or equal to LSTOP.
      IF(LEV.LE.LSTOP) WRITE(LU2,1000) N,IMAX,DELMAX,JMAX,DELUMAX
 1000 FORMAT(2(2X,I5),5E12.5,2X,I5,E12.5)
C
      RETURN
      END
C-------------------------------------------------------------
C          SUBROUTINE FILE: NSSDWAL
C-------------------------------------------------------------
      SUBROUTINE NSSDWAL
```

```
C
C         ***** C MESH VERSION *****
C
C         THIS SUBROUTINE CALCULATES DU FOR WALL
C         BOUNDARY POINTS FOR THE NAVIER-STOKES EQN.
C         USING NORMAL EXTRAPOLATION OF PRESSURE AND TEMPERATURE
C         I.E. ADIABATIC WALL CONDITION.
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'LUNITS.INC'
C
C         BOTTOM WALL
      DO 10 I=1,IPBBMX
C
C         SET POINTERS & CALCULATE WALL TANGENT
      IF(IPBB(3,I).EQ.4) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
        J4 = ABS(IP(3,IPBB(1,I)))
C
        TMP1 = Q(1,J2)-Q(1,J1)
        TMP2 = Q(2,J2)-Q(2,J1)
        DS1 = SQRT(TMP1*TMP1+TMP2*TMP2)
        TMP1 = Q(1,J3)-Q(1,J2)
        TMP2 = Q(2,J3)-Q(2,J2)
        DS2 = SQRT(TMP1*TMP1+TMP2*TMP2)
C
        TMP1 = DS1+DS2
        TMP2 = DS2/(DS1*TMP1)
        TMP3 = (DS2-DS1)/(DS1*DS2)
        TMP4 = DS1/(DS2*TMP1)
C
        DXDS = -Q(1,J1)*TMP2+Q(1,J2)*TMP3+Q(1,J3)*TMP4
        DYDS = -Q(2,J1)*TMP2+Q(2,J2)*TMP3+Q(2,J3)*TMP4
        TMP1 = SQRT(DXDS*DXDS+DYDS*DYDS)
C
        DX = DXDS/TMP1
        DY = DYDS/TMP1
        IBCOND = 1
      ELSE IF(IPBB(3,I).EQ.5) THEN
        J1 = ABS(IP(1,IPBB(1,I)))
        J2 = ABS(IP(4,IPBB(1,I)))
        J3 = ABS(IP(4,IPBB(2,I)))
        J4 = ABS(IP(3,IPBB(1,I)))
C
        THETA1 = ATAN2((Q(2,J2)-Q(2,J1)),(Q(1,J2)-Q(1,J1)))
        THETA2 = ATAN2((Q(2,J2)-Q(2,J3)),(Q(1,J2)-Q(1,J3)))
        THETA  = 0.5*(THETA1+THETA2)
CC        THETA = THETA1
C
        DX = COS(THETA)
```

```
        DY = SIN(THETA)
        IBCOND = 1
      ELSE
        WRITE(LU1,*)' ERROR WALBC9C NOT VALID WALL TYPE I=',I
      END IF
      IF(IBCOND.EQ.0) GOTO 10
C
C         CALCULATION OF DWT,DWN,AO
      RTMP1 = Q(3,J4)
      UTMP1 = Q(4,J4)/RTMP1
      VTMP1 = Q(5,J4)/RTMP1
      ETMP1 = Q(6,J4)
      PTMP1 = GM1*(ETMP1-0.5*RTMP1*(UTMP1*UTMP1+VTMP1*VTMP1))
      TTMP1 = GAMMA*PTMP1/RTMP1
      IF(PTMP1.LT.0.0) THEN
         WRITE(LU1,*) '** PTMP1<0.0 IN SDWALBC AT BOTTOM I=',I
         STOP
      END IF
C
C         CALCULATION OF CORRECTED DELTA'S
      Q(7,J2)  = GAMMA*PTMP1/TTMP1-Q(3,J2)
      Q(8,J2)  = 0.0-Q(4,J2)
      Q(9,J2)  = 0.0-Q(5,J2)
      Q(10,J2) = PTMP1/GM1-Q(6,J2)
C
   10 CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: OUTPUT3
C-----------------------------------------------------------------
      SUBROUTINE OUTPUT3
C
C         **** SPECIAL C-MESH FORMULATION ****
C
C         THIS SUBROUTINE CREATES THE OUTPUT FILE
C         CALL REST.DAT WHICH IS READ BY EULER
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'GAM.INC'
      INCLUDE 'LUNITS.INC'
C
C         CALCULATION OF LIFT FORCE COEFFICIENTS
      CORD = 0.0
      CFN  = 0.0
      CFT  = 0.0
C
      DO 5 I=0,IPBBMX
      IF(I.GT.0) THEN
        J1 = ABS(IP(1,IPBB(2,I)))
        J2 = ABS(IP(4,IPBB(2,I)))
```

```
      ELSE
        J1 = ABS(IP(1,IPBB(1,1)))
        J2 = ABS(IP(4,IPBB(1,1)))
      END IF
C
      IF(I.EQ.0) THEN
        TX1 = Q(1,J1)
        TY1 = Q(2,J1)
      END IF
      TCORD = (TX1-Q(1,J2))**2+(TY1-Q(2,J2))**2
      IF(TCORD.GT.CORD) CORD = TCORD
C
      DX = Q(1,J2)-Q(1,J1)
      DY = Q(2,J2)-Q(2,J1)
      DS = SQRT(DX*DX+DY*DY)
C
      P1 = GM1*(Q(6,J1)
     1    -0.5*(Q(4,J1)*Q(4,J1)+Q(5,J1)*Q(5,J1))/Q(3,J1))
      P2 = GM1*(Q(6,J2)
     1    -0.5*(Q(4,J2)*Q(4,J2)+Q(5,J2)*Q(5,J2))/Q(3,J2))
      TMP = P1+P2
      CFN = CFN+TMP*DX
      CFT = CFT+TMP*DY
    5 CONTINUE
C
      CORD = SQRT(CORD)
      QFS = ROFS*(UFS*UFS+VFS*VFS)*CORD
      CFN = -CFN/QFS
      CFT = CFT/QFS
C
      ALPHAR = 3.14159*ALPHA/180.0
      CL = CFN*COS(ALPHAR)-CFT*SIN(ALPHAR)
      CD = CFN*SIN(ALPHAR)+CFT*COS(ALPHAR)
C
C         CALCULATE SPECTRIAL RADIUS
      SRAD = (DELMAX(5)/DELMAX1(5))**(1./(NFINSH-NSTART))
C
      OPEN(UNIT=LU4,TYPE='NEW',FORM='UNFORMATTED')
C
C         SET CONSTANTS
      ICONST(11) = NSTART
      ICONST(12) = NFINSH
      ICONST(13) = ITIM
      ICONST(14) = ISTART
      ICONST(15) = LSTOP
      NICONST = 50
C
      RCONST(1)  = AMFS
      RCONST(2)  = ALPHA
      RCONST(3)  = CFL
      RCONST(4)  = AVISCF
      RCONST(5)  = ROFS
      RCONST(6)  = UFS
      RCONST(7)  = VFS
```

```
      RCONST(8)  = APFS
      RCONST(9)  = CORD
      RCONST(10) = CFN
      RCONST(11) = CFT
      RCONST(12) = CL
      RCONST(13) = CD
      RCONST(14) = CM
      RCONST(15) = DELMAX1(1)
      RCONST(16) = DELMAX1(2)
      RCONST(16) = DELMAX1(3)
      RCONST(18) = DELMAX1(4)
      RCONST(19) = DELMAX1(5)
      RCONST(20) = DELMAX(1)
      RCONST(21) = DELMAX(2)
      RCONST(22) = DELMAX(3)
      RCONST(23) = DELMAX(4)
      RCONST(24) = DELMAX(5)
      RCONST(25) = SRAD
      NRCONST = 50
CCC
CCC      WRITE(LU1,*) '  IE,JE,LMAX,IC1,IC2,JC2,IF2,JF2'
CCC      WRITE(LU1,*) '  IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX'
CCC      WRITE(LU1,*) '  DELTA,AK,Y0'
CCC      WRITE(LU1,1000) IE,JE,LMAX,IC1,IC2,JC2,IF2,JF2
CCC      WRITE(LU1,1000) IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
CCC      WRITE(LU1,1001) DELTA,AK,Y0
CCC 1000 FORMAT(1X,20I5)
CCC 1001 FORMAT(1X,10E13.4)
CCC
C
      WRITE(LU1,1004)
      WRITE(LU1,1005)ROFS,UFS,VFS,CORD
      WRITE(LU1,1006)CFN,CFT,CL,CD
 1004 FORMAT(//,5X,'SECTION LIFT PROPERTIES',/)
 1005 FORMAT(5X,'ROFS     =',F10.7,5X,'UFS      =',F10.7,
     1       5X,'VFS      =',F10.7,5X,'CHORD    =',F10.7)
 1006 FORMAT(5X,'CFN      =',F10.7,5X,'CFT      =',F10.7,
     1       5X,'CL       =',F10.7,5X,'CD       =',F10.7)
CCC
C
C         WRITE OUT GRID POINTERS
C
      WRITE(LU4) GLABEL1,GLABEL2,RLABEL1,RLABEL2
      WRITE(LU4) NICONST,NRCONST
      WRITE(LU4) (ICONST(K), K=1,NICONST)
      WRITE(LU4) (RCONST(K), K=1,NRCONST)
      WRITE(LU4) LMAX,IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
      WRITE(LU4) ((IPBIMX(M,N), M=1,2), N=1,LMAX)
      WRITE(LU4) ((LEVP(M,N), M=1,2), N=1,LMAX)
C
      DO 10 LEV = 1,LMAX
      WRITE(LU4) ((IP(M,N), M=1,9), N=LEVP(1,LEV),LEVP(2,LEV))
   10 CONTINUE
C
```

```
      DO 15 LEV=1,LMAX
      IF(IPBIMX(2,LEV).NE.0)
     1  WRITE(LU4) ((IPBI(M,N), M=1,3),
     2              N=IPBIMX(1,LEV),IPBIMX(2,LEV))
   15 CONTINUE
C
      WRITE(LU4) ((IPBU(M,N), M=1,3), N=1,IPBUMX)
      WRITE(LU4) ((IPBD(M,N), M=1,3), N=1,IPBDMX)
      WRITE(LU4) ((IPBT(M,N), M=1,3), N=1,IPBTMX)
      WRITE(LU4) ((IPBB(M,N), M=1,3), N=1,IPBBMX)
C
      DO 8 K =1,6
    8 WRITE(LU4) (Q(K,I), I=1,IQMAX)
C
      RETURN
      END
C-----------------------------------------------------------------
C          SUBROUTINE: OUTRESTT
C-----------------------------------------------------------------
      SUBROUTINE OUTRESTT
C
C         **** SPECIAL C-MESH FORMULATION ****
C
C         THIS SUBROUTINE CREATES A TEMPORARY OUTPUT FILE
C         CALL TREST.DAT WHICH IS READ BY EULER
C
      INCLUDE 'MAIN.INC'
      INCLUDE 'POINT.INC'
      INCLUDE 'INPT.INC'
      INCLUDE 'GAM.INC'
C
C         CALCULATION OF LIFT FORCE COEFFICIENTS
      CORD = 0.0
      CFN  = 0.0
      CFT  = 0.0
C
      DO 5 I=0,IPBBMX
      IF(I.NE.0) THEN
        J1 = ABS(IP(1,IPBB(2,I)))
        J2 = ABS(IP(4,IPBB(2,I)))
      ELSE
        J1 = ABS(IP(1,IPBB(1,1)))
        J2 = ABS(IP(4,IPBB(1,1)))
      END IF
C
      IF(I.EQ.0) THEN
        TX1 = Q(1,J1)
        TY1 = Q(2,J1)
      END IF
      TCORD = (TX1-Q(1,J2))**2+(TY1-Q(2,J2))**2
      IF(TCORD.GT.CORD) CORD = TCORD
C
      DX = Q(1,J2)-Q(1,J1)
      DY = Q(2,J2)-Q(2,J1)
```

```
      DS = SQRT(DX*DX+DY*DY)
C
      P1 = GM1*(Q(6,J1)
     1    -0.5*(Q(4,J1)*Q(4,J1)+Q(5,J1)*Q(5,J1))/Q(3,J1))
      P2 = GM1*(Q(6,J2)
     1    -0.5*(Q(4,J2)*Q(4,J2)+Q(5,J2)*Q(5,J2))/Q(3,J2))
      TMP = P1+P2
      CFN = CFN+TMP*DX
      CFT = CFT+TMP*DY
    5 CONTINUE
C
      CORD = SQRT(CORD)
      QFS = ROFS*(UFS*UFS+VFS*VFS)*CORD
      CFN = -CFN/QFS
      CFT = CFT/QFS
C
      ALPHAR = 3.14159*ALPHA/180.0
      CL = CFN*COS(ALPHAR)-CFT*SIN(ALPHAR)
      CD = CFN*SIN(ALPHAR)+CFT*COS(ALPHAR)
C
C         CALCULATE SPECTRIAL RADIUS
      SRAD = (DELMAX(5)/DELMAX1(5))**(1./(NFINSH-NSTART))
C
      OPEN(UNIT=4,NAME='TREST.DAT',TYPE='OLD',FORM='UNFORMATTED')
C
C         SET CONSTANTS
      ICONST(11) = NSTART
      ICONST(12) = NFINSH
      ICONST(13) = ITIM
      ICONST(14) = ISTART
      ICONST(15) = LSTOP
      NICONST = 50
C
      RCONST(1)  = AMFS
      RCONST(2)  = ALPHA
      RCONST(3)  = CFL
      RCONST(4)  = AVISCF
      RCONST(5)  = ROFS
      RCONST(6)  = UFS
      RCONST(7)  = VFS
      RCONST(8)  = APFS
      RCONST(9)  = CORD
      RCONST(10) = CFN
      RCONST(11) = CFT
      RCONST(12) = CL
      RCONST(13) = CD
      RCONST(14) = CM
      RCONST(15) = DELMAX1(1)
      RCONST(16) = DELMAX1(2)
      RCONST(16) = DELMAX1(3)
      RCONST(18) = DELMAX1(4)
      RCONST(19) = DELMAX1(5)
      RCONST(20) = DELMAX(1)
      RCONST(21) = DELMAX(2)
```

```
      RCONST(22) = DELMAX(3)
      RCONST(23) = DELMAX(4)
      RCONST(24) = DELMAX(5)
      RCONST(25) = SRAD
      NRCONST = 50
CCC
CCC      WRITE(1,*) '  IE,JE,LMAX,IC1,IC2,JC2,IF2,JF2'
CCC      WRITE(1,*) '  IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX'
CCC      WRITE(1,*) '  DELTA,AK,Y0'
CCC      WRITE(1,1000) IE,JE,LMAX,IC1,IC2,JC2,IF2,JF2
CCC      WRITE(1,1000) IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
CCC      WRITE(1,1001) DELTA,AK,Y0
CCC 1000 FORMAT(1X,20I5)
CCC 1001 FORMAT(1X,10E13.4)
CCC
C
CC      WRITE(1,1004)
CC      WRITE(1,1005)ROFS,UFS,VFS,CORD
CC      WRITE(1,1006)CFN,CFT,CL,CD
 1004 FORMAT(//,5X,'SECTION LIFT PROPERTIES',/)
 1005 FORMAT(5X,'ROFS     =',F10.7,5X,'UFS      =',F10.7,
     1       5X,'VFS      =',F10.7,5X,'CHORD    =',F10.7)
 1006 FORMAT(5X,'CFN      =',F10.7,5X,'CFT      =',F10.7,
     1       5X,'CL       =',F10.7,5X,'CD       =',F10.7)
CCC
C
C         WRITE OUT GRID POINTERS
C
      WRITE(4) GLABEL1,GLABEL2,RLABEL1,RLABEL2
      WRITE(4) NICONST,NRCONST
      WRITE(4) (ICONST(K), K=1,NICONST)
      WRITE(4) (RCONST(K), K=1,NRCONST)
      WRITE(4) LMAX,IQMAX,IPBUMX,IPBDMX,IPBTMX,IPBBMX
      WRITE(4) ((IPBIMX(M,N), M=1,2), N=1,LMAX)
      WRITE(4) ((LEVP(M,N), M=1,2), N=1,LMAX)
C
      DO 10 LEV = 1,LMAX
      WRITE(4) ((IP(M,N), M=1,9), N=LEVP(1,LEV),LEVP(2,LEV))
   10 CONTINUE
C
      DO 15 LEV=1,LMAX
      IF(IPBIMX(2,LEV).NE.0)
     1  WRITE(4) ((IPBI(M,N), M=1,3), N=IPBIMX(1,LEV),IPBIMX(2,LEV))
   15 CONTINUE
C
      WRITE(4) ((IPBU(M,N), M=1,3), N=1,IPBUMX)
      WRITE(4) ((IPBD(M,N), M=1,3), N=1,IPBDMX)
      WRITE(4) ((IPBT(M,N), M=1,3), N=1,IPBTMX)
      WRITE(4) ((IPBB(M,N), M=1,3), N=1,IPBBMX)
C
      DO 8 K =1,6
    8 WRITE(4) (Q(K,I), I=1,IQMAX)
C
      CLOSE(UNIT=4)
```

```
C
      RETURN
      END
C-----------------------------------------------------------------
C          DATA FILE: AIRFOIL.INP
C-----------------------------------------------------------------
RUN 219:
EULER ON C-MESH, NI INJECTION(6),CAR S/W DOUBLE T/E,
.75          ENTER FREE STREAM MACH NO., AMES
.12         ENTER ANGLE OF ATTACK
.95         ENTER CFL NO.
2           ENTER TYPE OF TIME STEP 0 = SINGLE TIME 2 = EACH CELL
.05         ENTER ARTIFICIAL VISCOSITY COEF. 0.<AVISCF<0.1
1  1000       ENTER NUMBER OF ITERATIONS: NSTART,NMAX
1E-5        ENTER CONVERGENCE CUT OFF DELSTP
1           ENTER LEVEL TO CHECK CONVERGENCE ON, LSTOP
0           DO YOU WANT THE INITIAL FLOW PRINTED??  0=NO   1=YES
1           ENTER TYPE OF INITIAL SOLUTION 0 = UNIFORM FLOW 1 = RESTART
[CFD.USAB.EULERCELL.EULERCMSH]REST.TMP
0           ENTER NAVIER-STOKES SWITCH INSSWT= 1:YES, 0:NO
2.342E7 .72 288.0    ENTER RE0,TREF FOR NAVIER-STOKES SUBDOMAIN
C-----------------------------------------------------------------
C*****************************************************************
C-----------------------------------------------------------------
```