GRID ADAPTATION FOR COMPLEX TWO-DIMENSIONAL TRANSONIC FLOWS

by

John Francis Dannenhoffer, III
B.S., Rensselaer Polytechnic Institute (1976)
M.E., Rensselaer Polytechnic Institute (1978)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Doctor of Science
in
Computational Fluid Dynamics
Department of Aeronautics and Astronautics
at the
Massachusetts Institute of Technology
August 25, 1987

Signature of author _____
Department of Aeronautics and Astronautics
August 25, 1987

Certified by _____
Professor Judson R. Baron
Thesis Supervisor, Department of Aeronautics and Astronautics

Certified by _____
Professor Earll M. Murman
Department of Aeronautics and Astronautics

Certified by _____
Professor Eugene E. Covert
Department of Aeronautics and Astronautics

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

GRID ADAPTATION FOR COMPLEX TWO-DIMENSIONAL TRANSONIC FLOWS

by

John Francis Dannenhoffer, III

Submitted to the Department of Aeronautics and Astronautics
on August 25, 1987
in partial fulfillment of the requirements for the degree of
Doctor of Science in Computational Fluid Dynamics

Numerical simulations of compressible flows are performed on grids whose resolution are chosen as a trade-off between fineness for solution accuracy and coarseness for computational efficiency. Fortunately for practical problems, the computational errors are not uniformly distributed over the domains, but instead have inaccuracies which are concentrated at isolated flow features. The technique developed here takes advantage of that maldistribution by combining a coarse global grid for efficiency with locally fine grids for accuracy. Any number of irregularly-shaped embedded regions, which are coupled to a fixed global grid by a multiple-grid method, are employed to achieve any user-specified level of accuracy. A wide variety of flow-field topologies are shown to be treated easily with the general data structure required for the irregularly-shaped embedded domains. Robust adaptation is achieved through the use of an expert system which controls the number of and location of embedded regions as well as error recovery for especially difficult cases. The adaptation technique is applied to the computation of transonic and supersonic isolated airfoils, airfoil/flap combinations, and biplanes. The present adaptively-refined solutions have the same accuracy as globally-refined solutions and are generally about ten times more efficient, where the factor depends on the case and accuracy required.

Thesis Supervisor: Dr. Judson R. Baron

Title: Professor of Aeronautics and Astronautics

Acknowledgements

Thomas Edison said that "Genius is one percent inspiration and ninety-nine percent perspiration". For all my friends, collegues, and family who helped me and put up with me through both the perspiration and inspiration, I am eternally grateful (even though genius doesn't apply).

Many thanks to my advisor, Professor Judson Baron, for putting up with me, for truely caring about the research, and for always being there to ask sometimes unusual but always thought-provoking questions.

To Professor Earll Murman, I am grateful for his enthusiasm and insight in separating the truely important from the not-so-important; for providing a different perspective in which to view my work, my thanks to Professor Eugene Covert.

I would also like to thank the official readers of my thesis, Professor William Usab and Dr. Ron-Ho Ni, for sharing their expertise and experiences in related work and for helping me tune this thesis with their helpful comments.

To Professor Michael Giles, I extend special appreciation for sharing his friendship and for helping me think through tough technical aspects when the rest of us mortals were too frustrated to go on.

Thanks also to: Dana Lindquist and Rich Shapiro, for their tireless effort is reading this thesis, catching typo's and sentences which sounded like they were written at 4 A.M.; Professor Ken Powell and Dr. Tom Roberts for their 4 A.M. discussions, their humor (and sometimes lack of humor);

Professor William Usab for always challenging me to prove that I was right; Bob Haimes for stimulating technical conversations and for prodding me to finish; and Bob Bruen for keeping the computer alive and well.

To my very close friends at United Technologies, Dr. Ron-Ho Ni, Dr. Dave Ives, Dr. Thomas Barber, and Dr. Roger Davis, thank you for continued friendship, shared technical discussions, and a much needed connection to reality.

Also, I would like to thank Harris Weingold and Herb Buckout for believing in me and fighting the 'system' in setting up my leave of absence from Pratt & Whitney.

This work was supported by AFOSR Grant 82-0136, Dr. James D. Wilson technical monitor. Thanks also to Doug Dwoyer for seeing to it that computer time was available for part of this work on the VPS-32 at NASA Langley.

Last, but certainly not least, I would like to thank my wonderful wife Joan and daughter Joanne for always being there, for putting up with lonely weeks in Connecticut, and for keeping the home-fires burning. They, more than anyone, helped me through the thesis-writing ordeal by sharing limitless loving and caring. I love you both. ♡

Contents

List of Figures

List of Tables

Nomenclature

a	speed of sound
A	area of control volume, streamtube area
$\hat{\mathbf{A}}$	Jacobian in intrinsic coordinates
$\hat{\mathbf{B}}$	source term in intrinsic coordiantes
c	airfoil chord (reference) length, element of C
$\mathbf{C}$	characteristic variable vector
c_p	coefficient of specific heat at constant pressure
c_v	coefficient of specific heat at constant volume
C_L	lift coefficient
C_D	drag coefficient
C_p	pressure coefficient
e	internal energy per unit mass
E	total energy per unit volume
$f(a,b)$	smoothing function
$\vec{F}$	force vector
f, g	rotated flux vectors
$\mathbf{F}, \mathbf{G}$	flux vectors in Cartesian coordinates
$\mathcal{F}$	contra-varient flux through a face

h	enthalpy per unit mass
$\mathbf{I}$	identity matrix
L	reference length, left eigenvector, lift
$\mathbf{L}$	left eigenvector matrix
m	mass per unit volume
$\dot{m}$	mass flow rate
M	Mach number
$\vec{M}$	momentum vector per unit volume
$\mathcal{M}$	matrix used in boundary condition calculations
n	independent variable in surface splines
$\vec{n}$	unit normal outward vector
p	pressure
q	magnitude of velocity
$\dot{q}$	rate of heat added to the fluid
$\tilde{q}$	streamwise momentum
R	gas constant
$\mathcal{R}$	refinement parameter
$\mathcal{R}_d$	division threshold
$\mathcal{R}_f$	fusion threshold
s, n	intrinsic coordinates
s	entropy
t	time
T	temperature

T_x^n	shift operator
$\mathbf{U}$	state vector in Cartesian coordinates
$\hat{\mathbf{U}}$	state vector in intrinsic coordinates
U_∞	free stream velocity
$\tilde{u}$	x-component of $\vec{M}$
$\tilde{v}$	y-component of $\vec{M}$
$\vec{V}$	velocity vector
W	rate of work done to the fluid
$\vec{x}$	position vector
x, y	Cartesian components of $\vec{x}$
α	angle change at a corner, angle of attack, measure of grid non-uniformity
β	compressibility factor, measure of grid non-uniformity
γ	ratio of specific heats
Γ	vortex strength
$\Delta \ldots$	change in ...
$\delta \ldots$	correction of ...
δ_x	difference operator
ϵ_L	fractional lift error
η	total pressure loss
κ	smoothing forcing function
θ	relative orientations of x, y and s, n

λ	eigenvalue, time step parameter
$\mathbf{\Lambda}$	eigenvalue matrix
μ	smoothing coefficient
μ_x	averaging operator
ν	specific volume, $1/\rho$
ϕ	measure of grid non-uniformity, smoothing switch variable
ψ	a value to be specified at a boundary
ρ	density
ρ_∞	free stream density
σ_d	inward-distribution coefficient
σ_i	injection coefficient
Σ	source strength
τ	volume of control volume
ξ, η	coordinates in computational space
$\|\ldots\|$	norm of ...
$\tilde{\nabla}\ldots$	undivided first difference of ...
$\tilde{\nabla}^2\ldots$	undivided second difference of ...

$()_{bc}$	a result of bounadry condition application
$()_{c}$	corresponging to the center of a cell
$()_{f}$	for a uniform free-stream
$()_{fvs}$	for a free-stream-plus-vortex-plus-source
$()_{ff}$	far-field
$()_{fs}$	free-stream
$()_{o}$	stagnation conditions, e.g. p_o and h_o
$()_{old}$	the value after the last iteration
$()_{pres}$	a prescribed value
$()_{s}$, $()_{e}$, $()_{n}$, $()_{w}$	corresponding to one of the sides of a cell
$()_{sw}$, $()_{se}$, $()_{ne}$, $()_{nw}$	corresponding to one of the corners of a cell
$()_{te}$	at the trailing edge
$()_{wall}$	corresponding to the wall

Chapter 1

Introduction

Over the last decade, the field of Computational Fluid Dynamics (CFD) has evolved into a major tool of the aerospace industry. In many cases, its daily use in the design and analysis of all types of aerospace vehicles has been prompted by the rising costs of building and testing real hardware and the decreasing costs of computers. In addition, there are certain vehicles which are virtually impossible to test, making computer simulations the only tool available to the designer; the National AeroSpace Plane (NASP) is a current example.

In order for designers to trust computer simulations, they must provide accurate results in a timely manner. This can only be achieved if the simulation is based upon both appropriate analytical models of the fluid physics (derived from fundamental principals) as well as accurate and efficient numerical procedures for obtaining approximate solutions to the physical model.

The objective of this thesis is to develop a computer-based procedure for accurately and efficiently approximating one such analytical fluid flow model. The particular fluid model employed throughout is the Euler flow equations (for compressible, inviscid, rotational flows) because of its applicability to a vast number of current aeronautical designs.

1.1 The Problem

In general, the steady state solution of a system of hyperbolic, partial-differential equations can be computed by a discrete approximation to the governing equations. The discreteness of the numerical approximation results in truncation errors which are typically related to both the local computational grid size and the local solution behavior. For problems of practical interest, the truncation errors tend to be unevenly distributed over the computational domain; over a large portion of the domain, the solution is smooth relative to the local mesh spacing, resulting in relatively small errors, whereas near flow *features* (such as shock waves, boundary layers, and wakes) the spatial variation of the solution is of the scale of the mesh spacing, causing significantly larger errors.

In order to ensure a given level of accuracy, one must control the largest error which occurs in the entire computational domain. Various techniques have been devised to accomplish this: modify the descriptive equations used near flow features, refine the computational grid globally, or refine the computational grid only in the vicinity of the offending features. Each of these are explored in the following paragraphs.

In the first of these, known as *equation adaptation*, the mathematical model of the physics in the vicinity of the features is altered so as to allow a more accurate solution, either by re-scaling so that the flow becomes smoother in the modified description (such as in boundary layer transformations[20]), or by accounting for mathematical singularities (such as in shock fitting[63]). This requires that the flow features first be detected and then classified by comparing them to known feature types. Once their classifications are known, the descriptive equations can be modified accordingly. The new description must be carefully formulated so that the global and local descriptions blend smoothly.

Many researchers have successfully employed equation adaptation by

solving only a limited class of problems in which the feature types were known *a priori*. For example, Moretti[63] has altered the description of inviscid flow fields at shocks through their shock fitting techniques. In both cases, it was assumed *a priori* that a shock was present. In another type of equation adaptation, inviscid/viscous interaction, Carter[19] and others assume the presence of a boundary layer and exploit that by lumping all viscous effects into a boundary/shear layer which is then coupled to an outer inviscid flow. In all such cases, the major difficulty with equation adaptation (that is, classification of detected features) has been circumvented by limiting the class of problems which they can treat. For general problems, these techniques are apt to fail.

At the other extreme, that is *global grid refinement*, the errors near flow features are decreased by decreasing the mesh spacing everywhere. This approach has the advantage that it is simple to implement and hence enjoys widespread popularity. However, it has the serious disadvantage that in order to control errors in a very small part of the domain, the amount of computer resources (memory and time) required is significantly increased.

The third approach, *grid adaptation*, falls between the two alternatives above in both complexity and effectiveness. Here, a common set of descriptive equations are used throughout, but the grid spacing is adjusted so that the integration algorithm properly captures the local physics near the flow features. This technique does not require that features be classified, but simply located, making grid adaptation much simpler than equation adaptation. Additionally, since the resolution is only enhanced locally at the features, the computations with grid adaptation tend to consume significantly fewer computer resources than global refinement.

In this thesis, grid adaptation is used to achieve accurate results for a modest expenditure of resources. Within the basic framework of adaptive grid algorithms, there are currently two major approaches, *grid point redis-*

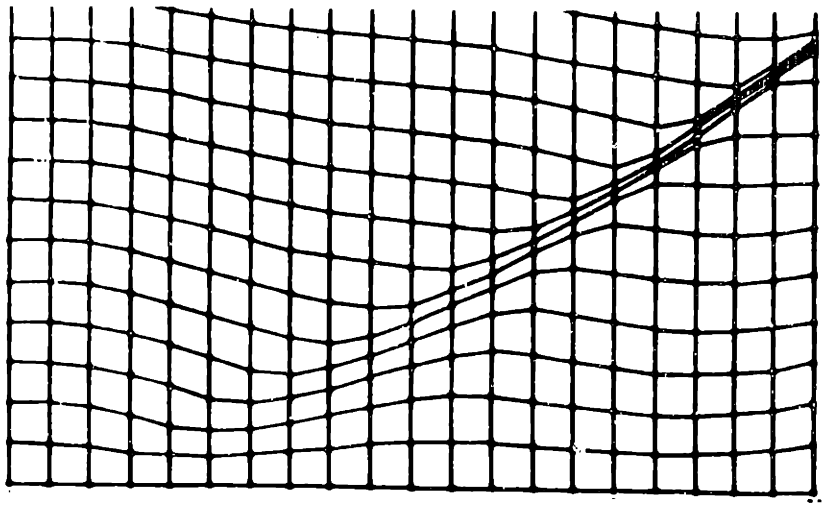


Figure 1.1: Grid generated by redistribution method

tribution, and ***mesh embedding*** or enrichment, which are described in the following paragraphs.

Grid point redistribution is an adaptive grid technique in which the nodes in a computational grid of a fixed topology and size are moved (either directly or indirectly) such that they congregate near flow features. Many different grid point redistribution schemes have appeared recently in the literature[4,5,7,18,21,27,33,34,35,36,38,43,44,46,47,48,61,64,72,78,86]; most are discussed in the survey articles by Thompson[80]. A typical example is shown in figure 1.1.

Early redistribution schemes were aimed at determining the 'optimal' grid point distribution on which the solutions of ordinary differential equations could be computed[9]. Here optimal refers to a grid for which the norm of the solution error is minimized. This could be achieved exactly by arranging the grid spacing such that the error was uniform over the domain.

The extension to partial differential equations (two or more dimensions)

introduced topological difficulties, resulting in the inability to achieve uniform errors over the entire domain. For example, grid line concentration at an isolated flow feature has to be dispersed gradually into the smooth regions away from the feature. The most classic example of this occurs in the airfoil problem, where the increased resolution needed at a normal shock standing on the airfoil surface somehow has to be extended away from the airfoil, into the nearly uniform free stream. Nevertheless, the redistribution technique has been extended with the goal of making the computational errors as nearly uniform as possible.

The most popular method of directly controlling grid point movement, and hence position, is through a spring analogy[64]. Each node is considered to be connected to its neighbors with tension springs, the spring constant of each being related to some measure of the local solution behavior (such as the current flow gradient). The entire set of grid points are then allowed to relax to their new positions and hence a redistributed grid is generated. Unfortunately, this technique allows the grid lines to possibly cross and/or more likely be excessively skewed. To circumvent this, Nakahashi[64] and others have added torsion springs to control the skew.

Others have avoided this problem by indirectly controlling the grid point motion. Here, some grid-smoothness equation, which contains forcing functions to approximately achieve the desired local grid spacing, are solved. Formulating the grid generation problem as the solution to a set of coupled Poisson equations lends itself directly to being used as grid-smoothness equations such as done by Mastin and Thompson[61]; others have posed the grid generation as the solution to a variational problem, where integrals of the desired grid concentration, smoothness, and skew are simultaneously minimized[75].

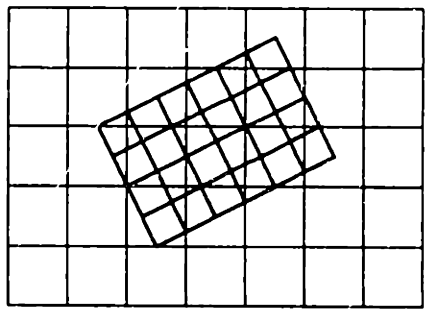
However the grid points are moved, the result is a grid with varying spacing and skew (figure 1.1), both of which have been shown to cause significant

errors in some computed solutions[60]. Also, some have reported a coupling which can occur between the evolving solution and the grid motion[32]; care must be exercised to avoid such oscillations. Despite these drawbacks, the redistribution method is very popular since the logic needed to implement the algorithm is relatively straightforward; independent of the existence or absence of flow features, the same operations can be performed for all cases.

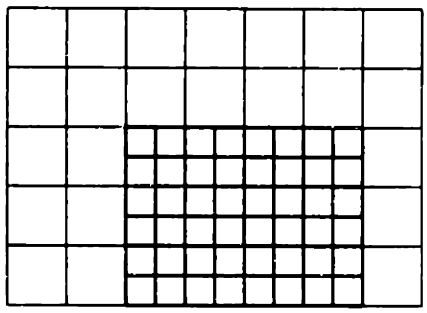
Grid embedding is a technique in which the solution is obtained on a composite grid, composed of a fixed global grid (in both computational and physical space) and any number of embedded regions. These embedded regions may be recursive (embedded regions may contain further embedded regions) and may be aligned with the global grid[85], be rotated[11], or may be topologically dissimilar, as shown in figure 1.2. The embedded regions are created by surrounding all nodes which have a higher than acceptable error. Many of these mesh embedding techniques are discussed in the survey article by Berger[10].

This procedure maintains the global grid accuracy outside embedded regions and simultaneously increases the accuracy in the embedded regions. In addition, since the global grid is fixed, the skew of the composite grid is the same as that of the initial global grid. Even though the scheme involves adding grid points (thereby increasing the total computational effort and required storage), they are added only near features, resulting in more efficiency than if comparable resolution were obtained by adding nodes globally.

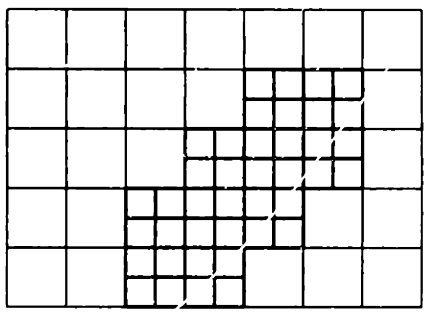
Since the fine regions are embedded only locally, artificial internal boundaries with an abrupt grid spacing change result; special care must be taken there to account for this abrupt change and to assure both stability and conservation[12]. For embedded regions which are aligned with the global grid, the coupling between the grids poses a rather minor problem for aligned patches, since nodes are shared by the two grids. On the other hand, rotated or topologically dissimilar embedded regions require iteration between



a: regularly-shaped, non-aligned



b: regularly-shaped, aligned



c: irregularly-shaped, aligned

Figure 1.2: Embedded mesh topologies

solutions on the two grids, with Dirichlet-like boundary conditions on each derived from the solution on the other.

The creation of the embedded patches can be accomplished rather easily for irregularly-shaped patches, simply by dividing cells adjacent to high error nodes. On the other hand, the use of rotated embedded regions frequently requires that the nodes with the high errors be clustered before an embedded patch can be created[11].

The chief disadvantage of embedded mesh adaptation is that the adaptation is a discrete process; either there is an embedded region or there is not. Hence small changes in the solution can result in rather severe differences in the structure of the computational grid. Additionally, this technique is more difficult to retrofit into an existing program than is a redistribution technique.

The above discussions of the two grid adaptation approaches leads one to the essential difference between them, which can be summarized as:

- in redistribution, the level of effort (the size of the grid) is specified and the "best" solution possible for that expense is obtained, and

- in grid embedding, the level of accuracy is specified and the "least expensive" grid (fewest grid points) possible to achieve that accuracy is employed.

The choice of which grid adaptation technique to pursue is therefore highly dependent on one's requirements. In cases where economic considerations determine the level of accuracy which is tolerable, redistribution is a preferable technique. In other cases, where a specified level of solution quality is more important than economics (within reason), the embedded grid approach is superior. In most cases in industry and academe the latter prevails and hence the embedded grid approach is pursued in this thesis.

1.2 Overview of Strategy Employed

The basic approach of the adapted, embedded solution procedure developed herein is to integrate the governing equations to steady state on successively refined grids until solutions on successive grids are "close" to each other. The grids are generated by employing a fixed global grid and automatically placing irregularly-shaped embedded grids where necessary. The governing equation integrator couples the solutions on the various grids through a generalization of Ni's multiple-grid method[67].

The major steps in the adaptation scheme can be summarized as:

1. Set up a fixed global grid.
2. Integrate the governing equations to convergence (steady state) on the current grid.
3. If the latest solution is "close" to the previously computed solution, then STOP.
4. Locate the pertinent flow features.
5. Create embedded meshes around the flow features and destroy the embedded meshes elsewhere.
6. Loop back to step 2.

This strategy has to be modified slightly for especially difficult cases. In order to expedite such modifications, an expert system is used to store the evolving adaptation strategy.

1.3 Overview of Thesis

This thesis describes the various components of the adaptation process described above. Included are derivations of the basic concepts, descriptions

of the algorithms employed, and analytical proofs and demonstrations of the properties of the various components. The resulting computer program, MITOSIS, is described fully in the appendices.

The name MITOSIS for the current computer program is derived from two sources: it is an acronym, standing for MIT Optimal Scheme for Inviscid Systems; and MITOSIS is the biological process of cell division, which is exactly the process used here to achieve high accuracy.

Chapter 2 describes the governing equations which are used throughout, namely the two-dimensional Euler equations. Following a brief derivation of them, appropriate physical boundary conditions for flows about airfoil-like bodies in a two-dimensional free-stream are derived. This is followed by an examination of the mathematical properties of the Euler equations which are exploited by the integration scheme.

Chapter 3 describes the basic integration scheme used, that is Ni's form of a one-step Lax-Wendroff-type scheme. After an extensive discussion of various techniques for implementing the boundary conditions numerically, the properties of the integration scheme (accuracy, consistency, stability, conservation, and wave propagation) are both derived analytically and numerical demonstrations of each are given. The discussion of the wave propagation characteristics of the basic scheme then leads to the development of the multiple-grid accelerator, whose properties are also explored. The chapter concludes with the development of a spatially-varying smoothing operator and a new smoothing coefficient partial differential equation.

In Chapter 4, the basic scheme with the multiple-grid accelerator is extended to embedded meshes. The extension is first developed conceptually, and then specific implementation issues in one and two dimensions are explored. The properties of the embedded mesh scheme are next examined, both analytically and through various computed demonstrations. The chapter concludes with a complete summary of the integration scheme, including

the basic grid operations, the multiple-grid accelerator, and the coupling mechanism.

Because the embedded regions can be irregularly shaped, one needs to use a more complex data structure than the usual i, j indexing used in most computational programs. Chapter 5 begins with a discussion of data structures and describes various alternatives. This then leads to a description of the data structure used in the present program as well as the various grid and data structure handling routines necessary for the efficient implementation of the data structure chosen.

The feature detection algorithm is the subject of Chapter 6. It begins with a basic description of the various type of features which are encountered. This then leads to a discussion of refinement parameters — how to choose and compute them. This is followed by the development of an algorithm for automatically choosing threshold values which separate features from the background physics. The chapter concludes with a desciption of the scheme used to adapt the grid based upon the refinement parameters and thresholds.

Chapter 7 offers a slight digression, its subject being expert systems. The chapter begins by describing expert systems, first in basic terms and then by examining their various components. To illustrate the power of an expert system, a local compressible flow problem is solved using both a forward- and backward-chaining expert system. From this, one can readily identify the strengths and weaknesses of expert systems, at least as compared with conventional systems. This then leads to the development of a new hybrid system approach (EXPROC) for structuring large, complex programs. The remainder of Chapter 7 is devoted to establishing the 'hooks' upon which the grid adaptation process can be integrated into the hybrid system approach.

Chapter 8 describes the evolution of the strategy (knowledge base) which controls the adaptation process. The chapter begins with the development

of the basic strategy for the initial test case (that is, an RAE-2822 airfoil at $M_\infty = 0.75$ and $\alpha = 3°$), and a comparison of the efficiency and accuracy of the adaptively-refined solution as compared with a globally refined solution. Because the computed solution does not agree with solutions computed by others, the possible causes of these discrepencies are investigated.

This basic strategy is then applied to the first five test cases taken from the AGARD Working Group 7 workshop[3] (all for a NACA-0012 airfoil but with differing far-field conditions which yield significantly different flow-field topologies). Through these test cases, deficiencies in the basic strategy are uncovered and subsequently eliminated by a systematic growth of the knowledge base. Finally, since the data structure employed allows arbitrarily-shaped embedded regions, this concept is extended to arbitrarily-shaped global domains; computations of the flows over a flapped airfoil and a biplane are used to illustrate this feature.

The major conclusions are presented in the final Chapter (9). This chapter also contains a discussion of possible extensions as well as limitations of the method developed.

Four appendices complete the thesis; the first three contain material pertaining to the MITOSIS program (a user's guide, the program listing, and a sample input and output). The final appendix is a user's guide for the EXPROC knowledge base language.

Chapter 2

Governing Equations and Boundary Conditions

This chapter describes the governing equations and boundary conditions to be solved. It begins with a brief derivation of the Euler equations in both integral and differential form, along with the assumptions needed to derive them. The equations are then non-dimensionalized and auxiliary parameters which will be used throughout the remainder of the thesis are given. Next, the governing equations are cast in Cartesian form in two dimensions, and the Jacobians of the resulting vector equations are presented. The equations are then cast in intrinsic coordinates and the eigenvalues and eigenvectors of the resulting system are determined, which in turn leads to the determination of the characteristic variables. Finally, physical boundary conditions needed to close the problem analytically are discussed.

As noted in the Introduction, the primary purpose of this thesis is to develop an adaptation method which allows the accurate and efficient calculation of any hyperbolic system of equations. To this end, the material covered in this chapter serves merely to define the particular set of governing equations on which the adaptation strategies will be tested.

2.1 Derivation of the Euler Equations

The Euler equations are a set of coupled equations which state the conservation of mass (continuity), Newton's second law applied to the fluid (momentum), and the conservation of energy. These laws will be derived here in integral form and then converted to the appropriate differential form. Additionally, a state equation which is needed to close the system is presented.

Consider a fluid flowing through the control volume C shown in figure 2.1. This control volume is taken to be stationary and of fixed size. Also shown in the figure is the outward normal $\vec{n}$ from the surface of the control volume ∂C. The fluid flow through the control volume is characterized by its density ρ, velocity $\vec{V}$, pressure p and temperature T.

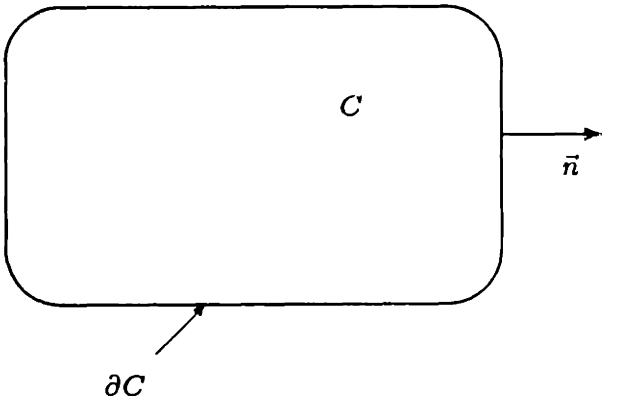


Figure 2.1: Typical control volume.

2.1.1 Continuity equation

The continuity equation is based on the law of the conservation of mass, which simply states

$$\Delta m_{\text{unsteady}} - \Delta m_{\text{influx}} = 0 \tag{2.1}$$

where $\Delta m_{\text{unsteady}}$ is the rate of mass accumulation in the control volume and Δm_{influx} is the net mass flow into the control volume.

This in general holds independently for each species in the flow. For the purposes of this thesis, the following simplifying assumption will be made:

Assumption 2.1 *The fluid is homogeneous. This implies that the following are excluded: mixture of gases, chemical reaction, change of phase, changes in molecular weight and specific heats causes by combustion, etc.*

The total mass enclosed by the control volume at any time is simply the integral of the density over the whole control volume. Its rate of change due to unsteady effects in the control volume is then given by

$$\Delta m_{\text{unsteady}} = \frac{d}{dt}\iint_C \rho \, d\tau = \iint_C \frac{\partial \rho}{\partial t} \, d\tau \tag{2.2}$$

where τ is the volume of the control volume C. The order of differentiation and integration can be interchanged because the control volume is assumed to be fixed.

The mass flux entering the control volume per unit time is given by

$$\Delta m_{\text{influx}} = -\oint_{\partial C} \rho(\vec{V} \cdot \vec{n}) \, dA \tag{2.3}$$

where $\vec{V}$ is the local velocity and dA is the surface area along the closed contour ∂C. Substituting equations (2.2) and (2.3) into (2.1) yields the continuity equation in integral form

$$\iint_C \frac{\partial \rho}{\partial t} \, d\tau + \oint_{\partial C} \rho(\vec{V} \cdot \vec{n}) \, dA = 0 \tag{2.4}$$

The divergence theorem can now be applied to the second term, which yields after rearranging

$$\iint_C \left[\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V})\right] d\tau = 0 \tag{2.5}$$

Since the control volume is arbitrary, equation (2.5) can be satisfied only if the integrand vanishes, yielding the continuity equation in differential form

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) = 0 \tag{2.6}$$

2.1.2 Momentum equation

This vector equation is derived by applying Newton's second law to a flowing fluid. It states that

$$\Delta \vec{M}_{\text{unsteady}} - \Delta \vec{M}_{\text{influx}} = \vec{F}_{\text{surface}} + \vec{F}_{\text{body}} \tag{2.7}$$

where $\Delta \vec{M}_{\text{unsteady}}$ is the unsteady rate of momentum accumulation in the control volume, $\Delta \vec{M}_{\text{influx}}$ is the net momentum influx through ∂C, and $\vec{F}_{\text{surface}}$ and $\vec{F}_{\text{body}}$ are the surface and body forces respectively which act on the fluid in C. The local momentum per unit volume, $\vec{M}$, is a vector quantity which is related to the density and velocity by

$$\vec{M} = \rho \vec{V} \tag{2.8}$$

Thus, equation (2.7) is actually a set of equations which apply independently in each coordinate direction.

The rate of change of momentum due to the unsteady nature of the flow gives (in a similar manner as with the continuity equation)

$$\Delta \vec{M}_{\text{unsteady}} = \frac{d}{dt} \iint_C \vec{M} d\tau = \iint_C \frac{\partial \vec{M}}{\partial t} d\tau \tag{2.9}$$

while the rate of change of momentum due to the net rate of inflow of momentum through ∂C is

$$\Delta \vec{M}_{\text{influx}} = - \oint_{\partial C} \vec{M} (\vec{V} \cdot \vec{n}) \, dA \tag{2.10}$$

Surface forces are those which act on the fluid only through the surface of the control volume ∂C. Now make the following assumption:

Assumption 2.2 *The flow is inviscid. This implies that shear stresses are absent, despite shearing deformations in the fluid. The velocity of an inviscid fluid at a solid boundary does not necessarily vanish as it must for a viscous fluid.*

This results in surface forces which act only perpendicular to the control surface. Thus the only surface force exerted on the fluid is the pressure which is given as

$$\vec{F}_{\text{surface}} = -\oint_{\partial C} p\vec{n}\, dA \tag{2.11}$$

Also make the following assumption:

Assumption 2.3 *There are no body forces present, or*

$$\vec{F}_{\text{body}} = 0 \tag{2.12}$$

This implies that the following forces are absent (among possible others): gravity, electrostatic, or magnetic.

Substituting equations (2.9) through (2.12) into (2.7) yields the momentum equation in integral form

$$\iint_C \frac{\partial \vec{M}}{\partial t}\, d\tau + \oint_{\partial C} \vec{M}(\vec{V}\cdot\vec{n})\, dA = -\oint_{\partial C} p\vec{n}\, dA \tag{2.13}$$

As before, this equation can be converted to differential form, yielding

$$\frac{\partial \vec{M}}{\partial t} + \vec{M}(\nabla\cdot\vec{V}) + (\vec{V}\cdot\nabla)\vec{M} + \nabla p = 0 \tag{2.14}$$

Recall that equations (2.13) and (2.14) are vector equations, resulting in one scalar equation for each coordinate direction.

2.1.3 Energy equation

The energy equation comes from the conservation of energy, which states

$$\Delta E_{\text{unsteady}} - \Delta E_{\text{influx}} = \dot{q} + W \tag{2.15}$$

where $\Delta E_{\text{unsteady}}$ is the unsteady rate of energy accumulation and ΔE_{influx} is the net rate of energy influx. In both these terms, ΔE refers to the total energy per unit volume which contains both the internal energy and the kinetic energy. (Potential energy is not included since there is none by Assumption 2.3). The rate of heat added to the fluid is given by $\dot{q}$ whereas W denotes the rate of work done on the fluid.

Again, as for the continuity and momentum equations, the rate of accumulation is given by

$$\Delta E_{\text{unsteady}} = \frac{d}{dt}\iint_C E\, d\tau = \iint_C \frac{\partial E}{\partial t}\, d\tau \tag{2.16}$$

and the net energy influx by

$$\Delta E_{\text{influx}} = -\oint_{\partial C} E(\vec{V}\cdot\vec{n})\, dA \tag{2.17}$$

Now make the following assumption:

Assumption 2.4 *The flow is adiabatic. This implies that there is no heat transferred to or from the surroundings, or*

$$\dot{q} = 0 \tag{2.18}$$

This assumption is valid except in the cases of "cold" or "hot" walls, as in a combustor or turbine or in cases involving radiation, etc.

The rate of work done on the fluid is given by a force times distance per unit time (velocity), or

$$W = -\oint_{\partial C} p\vec{V}\cdot\vec{n}\, dA \tag{2.19}$$

Again, the inviscid assumption (2.2) has been used here.

Substituting equations (2.16) through (2.19) into (2.15) yields the energy equation in integral form

$$\iiint_C \frac{\partial E}{\partial t}\, d\tau + \oint_{\partial C} E\vec{V} \cdot \vec{n}\, dA = -\oint_{\partial C} p\vec{V} \cdot \vec{n}\, dA \tag{2.20}$$

which can be converted to the differential form

$$\frac{\partial E}{\partial t} + \nabla \cdot \left[(E+p)\vec{V}\right] = 0 \tag{2.21}$$

As stated earlier, E is the total energy per unit volume, which consists of both internal and kinetic energies, which can be written as

$$E = \rho e + \frac{1}{2}\rho \vec{V} \cdot \vec{V} \tag{2.22}$$

where e is the internal energy per unit mass. Now the enthalpy per unit mass of the fluid is defined by

$$h = e + \frac{p}{\rho} \tag{2.23}$$

and the total (or stagnation) enthalpy per unit mass as

$$h_o = h + \frac{1}{2}\vec{V} \cdot \vec{V} = e + \frac{p}{\rho} + \frac{1}{2}\vec{V} \cdot \vec{V} \tag{2.24}$$

Combining equations (2.22) to (2.24) yields

$$E = \rho h_o - p \tag{2.25}$$

which can be substituted into (2.21) to yield an alternate form of the energy equation in differential form

$$\frac{\partial E}{\partial t} + \nabla \cdot (\rho h_o \vec{V}) = 0 \tag{2.26}$$

2.1.4 State equation

Above, three equations (continuity, momentum, and energy) have been derived in the four dependent variables ρ, $\vec{M}$, E, and p. (Note that $\vec{V}$ appeared in the equations, but it can easily be eliminated using the definition of $\vec{M}$.) A state equation, which will close this system, can be obtained from the following assumption:

Assumption 2.5 *The fluid is a thermally perfect gas. This implies that the gas obeys the thermal equation of state*

$$p = \rho RT \tag{2.27}$$

where R is the gas constant. This assumption is valid for moderate pressures and temperatures. At low temperatures and high pressures, the intermolecular forces become important; these are the so-called van der Waals forces which account for the possibility of liquifying a gas. At high temperatures and low preesures, dissociation and internal mode excitations may play a role.

For a homogeneous fluid (Assumption 2.1), the internal energy is solely a function of temperature and specific volume, or

$$e = e(T, \nu) \tag{2.28}$$

which yields

$$de = \frac{\partial e}{\partial T} dT + \frac{\partial e}{\partial \nu} d\nu \tag{2.29}$$

It may be shown that for any pure substance satisfying (2.27), it is also true that

$$\frac{\partial e}{\partial \nu} = 0 \tag{2.30}$$

Now define the specific heat at constant volume as

$$c_v = \frac{\partial e}{\partial T} \tag{2.31}$$

so that equation (2.29) becomes

$$de = c_v dT \tag{2.32}$$

Since e is a function of T only, and it can be seen by (2.27) that p/ρ is also a function of T only, it follows from (2.23) that h is a function of T only, or

$$dh = \frac{\partial h}{\partial T} dT \tag{2.33}$$

Defining the specific heat at constant pressure and expanding using (2.23) and (2.31) yields

$$c_p = \frac{\partial h}{\partial T} = \frac{\partial}{\partial T}\left(e + \frac{p}{\rho}\right) = \frac{\partial e}{\partial T} + \frac{\partial}{\partial T}(RT) = c_v + R \tag{2.34}$$

Substituting (2.34) into (2.33) yields

$$dh = c_p dT \tag{2.35}$$

Further, define the ratio of specific heats as

$$\gamma = c_p / c_v \tag{2.36}$$

Combining (2.34) and (2.36) yields the useful results

$$c_p = \frac{\gamma}{\gamma - 1} R, \quad c_v = \frac{R}{\gamma - 1}, \quad \gamma = 1 + \frac{R}{c_v} \tag{2.37}$$

Now make the following assumption:

Assumption 2.6 *The fluid is a calorically perfect gas. This implies that the specific heats at constant pressure and volume (c_p and c_v) are constants, independent of temperature. This assumption is valid at moderate pressures and temperatures.*

Thus equation (2.35) can be integrated to yield

$$h = c_p T \tag{2.38}$$

Substitute these latter results and (2.24) into (2.27) to yield

$$p = \frac{\gamma - 1}{\gamma} \rho \left[h_o - \frac{1}{2} \vec{V} \cdot \vec{V} \right] \tag{2.39}$$

or

$$p = (\gamma - 1) \left[E - \frac{1}{2} \rho \vec{V} \cdot \vec{V} \right] \tag{2.40}$$

2.1.5 Summary

The governing equations are now collected and rewritten in a consistent set of dependent variables, namely ρ, $\vec{M}$, E, and p. The first three variables represent the conserved quantities (mass, momentum, and energy per unit volume), and are thus known as the conservation variables. The equations are said to be in the strong conservation law form since the coefficients of the derivative terms are constants.

The governing equations in integral form are given by

continuity:

$$\iint_C \frac{\partial \rho}{\partial t} d\tau + \oint_{\partial C} \vec{M} \cdot \vec{n} \, dA = 0 \tag{2.41}$$

momentum:

$$\iint_C \frac{\partial \vec{M}}{\partial t} d\tau + \oint_{\partial C} \vec{M} \left(\frac{\vec{M}}{\rho} \cdot \vec{n} \right) dA = - \oint_{\partial C} p \vec{n} \, dA \tag{2.42}$$

energy:

$$\iint_C \frac{\partial E}{\partial t} d\tau + \oint_{\partial C} E \left(\frac{\vec{M}}{\rho} \cdot \vec{n} \right) dA = - \oint_{\partial C} p \left(\frac{\vec{M}}{\rho} \cdot \vec{n} \right) dA \tag{2.43}$$

The governing equations in differential form are given by

continuity:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{M} = 0 \tag{2.44}$$

momentum:

$$\frac{\partial \vec{M}}{\partial t} + \vec{M} \left(\nabla \cdot \frac{\vec{M}}{\rho} \right) + \left(\frac{\vec{M}}{\rho} \cdot \nabla \right) \vec{M} + \nabla p = 0 \tag{2.45}$$

energy:

$$\frac{\partial E}{\partial t} + \nabla \cdot \left[(E+p)\frac{\vec{M}}{\rho} \right] = 0 \tag{2.46}$$

To close the system of equations, the pressure is related to the conserved quantities by the state equation:

$$p = (\gamma - 1)\left[E - \frac{1}{2}\frac{\vec{M}\cdot\vec{M}}{\rho} \right] \tag{2.47}$$

2.2 Non-Dimensionalization and Auxiliary Parameters

In the previous section, the governing equations were derived in terms of dimensional quantities. In this section, a suitable non-dimensionalization of the governing equations will be given. Also, physical parameters which will be used elsewhere in the thesis, namely the velocity, speed of sound, Mach number, and stagnation pressure, will be derived in terms of the dependent variables.

2.2.1 Non-dimensionalization

The purpose of non-dimensionalization is to cast the equations such that the effect of characteristic parameters such as Reynolds number or Prandtl number can easily be ascertained. A secondary purpose of non-dimensionalization is to scale the variables such that all have approximately the same numerical size which helps minimize round-off errors in calculations.

For purposes of this thesis, the following quantities will be chosen as the basis on non-dimensionalization:

L reference length (e.g., the chord length)

U_∞ the free-stream velocity

ρ_∞ the free-stream density

These quantities were chosen so that the dependent variables are all of order-of-magnitude unity.

If the dimensional quantities are given as above (without primes) and the non-dimensional quantities are given with primes, the following results

$$\vec{x}' = \frac{\vec{x}}{L}, \quad t' = \frac{t}{L/U_\infty}, \quad \rho' = \frac{\rho}{\rho_\infty} \tag{2.48a}$$

$$\vec{M}' = \frac{\vec{M}}{\rho_\infty U_\infty}, \quad E' = \frac{E}{\rho_\infty U_\infty^2}, \quad p' = \frac{p}{\rho_\infty U_\infty^2} \tag{2.48b}$$

Substituting these results into the continuity equation (2.44) yields

$$\frac{\partial \rho'}{\partial t'} + \nabla' \cdot \vec{M}' = 0 \tag{2.49}$$

Note that the "del" operator, ∇', also has been non-dimensionalized and that a common factor of $\rho_\infty U_\infty / L$ has been eliminated. The form of (2.49) is identical to (2.44) except for the primes, indicating that there are no characteristic parameters associated with this equation. As a matter of fact, if this same process is carried out with the remaining governing equations, one finds that none of these contain any characteristic parameters either. Therefore for the remainder of this thesis, all variables will be treated implicitly as non-dimensional with the prime notation being dropped.

2.2.2 Auxiliary parameters

In addition to the dependent variables defined previously, there are other physical quantities which will appear throughout the thesis. It is the purpose of this section to define these terms and to develop suitable forms for these quantities in terms of the dependent variables.

The first parameter is the magnitude of the velocity, q. Although the velocity was used in the derivation of the governing equations, it is considered an auxiliary parameter here because it is not one of the dependent variables.

The velocity can be written directly from the definition of momentum, or

$$q = \left[\frac{\vec{M} \cdot \vec{M}}{\rho^2}\right]^{1/2} \tag{2.50}$$

As expected, the velocity is non-dimensionalized by U_∞.

The second auxiliary parameter is the speed of sound, which is given by the relation

$$a = \sqrt{\gamma RT} = \sqrt{\gamma p/\rho} \tag{2.51}$$

This can then be written in terms of the dependent variables by using (2.47) to yield

$$a = \left[\gamma(\gamma - 1)\left(\frac{E}{\rho} - \frac{1}{2}\frac{\vec{M} \cdot \vec{M}}{\rho^2}\right)\right]^{1/2} \tag{2.52}$$

The speed of sound is also non-dimensionalized by U_∞.

The third auxiliary parameter is the Mach number, which is defined as the ratio of the velocity magnitude to the speed of sound. Written in terms of (2.50) and (2.52), this becomes

$$M = \frac{q}{a} = \left[\frac{\vec{M} \cdot \vec{M}}{\gamma(\gamma - 1)\left[\rho E - \frac{1}{2}(\vec{M} \cdot \vec{M})\right]}\right]^{1/2} \tag{2.53}$$

The Mach number is a non-dimensional parameter.

The final parameter defined in this section is the stagnation (or total) pressure which is related to the static pressure on an isentropic basis by

$$\frac{p_o}{p} = \left[1 + \frac{\gamma - 1}{2}M^2\right]^{\frac{\gamma}{\gamma-1}} \tag{2.54}$$

Written in terms of the dependent variables this becomes

$$\frac{p_o}{p} = \left[1 + \frac{\vec{M} \cdot \vec{M}}{\gamma[2\rho E - \vec{M} \cdot \vec{M}]}\right]^{\frac{\gamma}{\gamma-1}} \tag{2.55}$$

The total pressure is non-dimensionalized the same as the static pressure, namely by $\rho_\infty U_\infty^2$.

2.3 Cartesian Form

The governing equations (2.41) through (2.46) will now be cast in Cartesian variables in two dimensions. In order to do so, the vector quantities $\vec{x}$ and $\vec{M}$ will be written in terms of their components as

$$\vec{x} = (x, y) \tag{2.56}$$

and

$$\vec{M} = (\tilde{u}, \tilde{v}) \tag{2.57}$$

where $\tilde{u}$ is the x-component of momentum, etc. Here the tilde notation is used so that the components of the momentum vector can readily be discerned from the components of a velocity vector.

Direct substitution of (2.56) and (2.57) into the integral form of the continuity equation (2.41) yields

$$\iint_C \frac{\partial \rho}{\partial t}\, dx\, dy + \oint_{\partial C} (\tilde{u}\, dy - \tilde{v}\, dx) = 0 \tag{2.58}$$

In a similar manner, the first component of the momentum equation in integral form (2.42) yields

$$\iint_C \frac{\partial \tilde{u}}{\partial t}\, dx\, dy + \oint_{\partial C} \tilde{u} \left(\frac{\tilde{u}}{\rho} dy - \frac{\tilde{v}}{\rho} dx \right) = - \oint_{\partial C} p\, dy \tag{2.59}$$

and the second component yields

$$\iint_C \frac{\partial \tilde{v}}{\partial t}\, dx\, dy + \oint_{\partial C} \tilde{v} \left(\frac{\tilde{u}}{\rho} dy - \frac{\tilde{v}}{\rho} dx \right) = + \oint_{\partial C} p\, dx \tag{2.60}$$

Similarly, substituting into the energy equation (2.43) yields

$$\iint_C \frac{\partial E}{\partial t}\, dx\, dy + \oint_{\partial C} E \left(\frac{\tilde{u}}{\rho} dy - \frac{\tilde{v}}{\rho} dx \right) = - \oint_{\partial C} p \left(\frac{\tilde{u}}{\rho} dy - \frac{\tilde{v}}{\rho} dx \right) \tag{2.61}$$

Equations (2.58) to (2.61) can all be written in a compact form, known as the vector form, which is given by

$$\iint_C \frac{\partial \mathbf{U}}{\partial t}\, dx\, dy + \oint_{\partial C} (\mathbf{F}\, dy - \mathbf{G}\, dx) = 0 \tag{2.62}$$

where the vectors are given by

$$\mathbf{U} = \begin{pmatrix} \rho \\ \tilde{u} \\ \tilde{v} \\ E \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \tilde{u} \\ \frac{\tilde{u}^2}{\rho} + p \\ \frac{\tilde{u}\tilde{v}}{\rho} \\ (E+p)\frac{\tilde{u}}{\rho} \end{pmatrix}, \quad \mathbf{G} = \begin{pmatrix} \tilde{v} \\ \frac{\tilde{u}\tilde{v}}{\rho} \\ \frac{\tilde{v}^2}{\rho} + p \\ (E+p)\frac{\tilde{v}}{\rho} \end{pmatrix} \tag{2.63}$$

Here $\mathbf{U}$ is called the state vector and $\mathbf{F}$ and $\mathbf{G}$ are the flux vectors. Similar substitutions into the differential forms of the governing equations (2.44) to (2.46) yields the vector equation

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = 0 \tag{2.64}$$

where the vectors $\mathbf{U}$, $\mathbf{F}$, and $\mathbf{G}$ have the definitions as given in (2.63).

To complete the definition of the governing equations in Cartesian form, the state equation (2.47) can be written as

$$p = (\gamma - 1)\left[E - \frac{1}{2}\frac{\tilde{u}^2 + \tilde{v}^2}{\rho}\right] \tag{2.65}$$

The vector forms of the governing equations [(2.62) and (2.64)] are the forms that will be used most frequently throughout the remainder of the thesis.

The Jacobians of the flux vectors, which also will appear frequently throughout the thesis, are given by the matrices whose elements are

$$\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right)_{i,j} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{U}_j} \quad \text{and} \quad \left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\right)_{i,j} = \frac{\partial \mathbf{G}_i}{\partial \mathbf{U}_j} \tag{2.66}$$

Using the definition given in (2.63), the Jacobians for the two-dimensional Euler equation in Cartesian coordinates are

$$\frac{\partial \mathbf{F}}{\partial \mathbf{U}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{\tilde{u}^2}{\rho^2} + \frac{\partial p}{\partial \rho} & 2\frac{\tilde{u}}{\rho} + \frac{\partial p}{\partial \tilde{u}} & \frac{\partial p}{\partial \tilde{v}} & \frac{\partial p}{\partial E} \\ -\frac{\tilde{u}\tilde{v}}{\rho^2} & \frac{\tilde{v}}{\rho} & \frac{\tilde{u}}{\rho} & 0 \\ -\frac{\tilde{u}(E+p)}{\rho^2} + \frac{\tilde{u}}{\rho}\frac{\partial p}{\partial \rho} & \frac{E+p}{\rho} + \frac{\tilde{u}}{\rho}\frac{\partial p}{\partial \tilde{u}} & \frac{\tilde{u}}{\rho}\frac{\partial p}{\partial \tilde{v}} & \frac{\tilde{u}}{\rho}\left(1 + \frac{\partial p}{\partial E}\right) \end{pmatrix} \tag{2.67}$$

and

$$\frac{\partial \mathbf{G}}{\partial \mathbf{U}} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -\frac{\tilde{u}\tilde{v}}{\rho^2} & \frac{\tilde{v}}{\rho} & \frac{\tilde{u}}{\rho} & 0 \\ -\frac{\tilde{v}^2}{\rho^2} + \frac{\partial p}{\partial \rho} & \frac{\partial p}{\partial \tilde{u}} & 2\frac{\tilde{v}}{\rho} + \frac{\partial p}{\partial \tilde{v}} & \frac{\partial p}{\partial E} \\ -\frac{\tilde{v}(E+p)}{\rho^2} + \frac{\tilde{v}}{\rho}\frac{\partial p}{\partial \rho} & \frac{\tilde{v}}{\rho}\frac{\partial p}{\partial \tilde{u}} & \frac{E+p}{\rho} + \frac{\tilde{v}}{\rho}\frac{\partial p}{\partial \tilde{v}} & \frac{\tilde{v}}{\rho}\left(1 + \frac{\partial p}{\partial E}\right) \end{pmatrix} \tag{2.68}$$

where

$$\begin{aligned} \frac{\partial p}{\partial \rho} &= \left(\frac{\gamma-1}{2}\right)\frac{\tilde{u}^2+\tilde{v}^2}{\rho^2}, & \frac{\partial p}{\partial \tilde{u}} &= -(\gamma-1)\frac{\tilde{u}}{\rho} \\ \frac{\partial p}{\partial \tilde{v}} &= -(\gamma-1), \frac{\tilde{v}}{\rho} & \frac{\partial p}{\partial E} &= (\gamma-1) \end{aligned} \tag{2.69}$$

2.4 Intrinsic Coordinates

The Euler equations, which were developed in Cartesian coordinates in the previous section, are now cast in intrinsic coordinates as shown in figure 2.2. The Cartesian system, which is denoted by (x, y), has a fixed orientation with respect to the geometry of the problem. The intrinsic coordinate system (s, n) on the other hand is one which is attached to the flow, such that the s-direction is aligned with the local flow direction. The local orientation of the two coordinate system is described by the angle θ which may vary throughout the flow field.

From trigonometric considerations, the two coordinate systems are related by

$$\frac{\partial}{\partial s} = +\cos\theta\,\frac{\partial}{\partial x} + \sin\theta\,\frac{\partial}{\partial y} \tag{2.70a}$$

and

$$\frac{\partial}{\partial n} = -\sin\theta\,\frac{\partial}{\partial x} + \cos\theta\,\frac{\partial}{\partial y} \tag{2.70b}$$

which can be inverted to yield

$$\frac{\partial}{\partial x} = +\cos\theta\,\frac{\partial}{\partial s} - \sin\theta\,\frac{\partial}{\partial n} \tag{2.71a}$$

and

$$\frac{\partial}{\partial y} = +\sin\theta\,\frac{\partial}{\partial s} + \cos\theta\,\frac{\partial}{\partial n} \tag{2.71b}$$

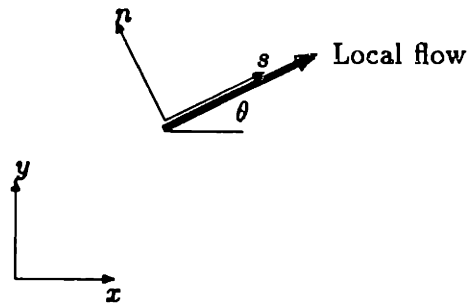


Figure 2.2: Coordinate Systems

If the streamwise momentum component is defined as $\tilde{q}$, the momentum components in the two systems are related by

$$\tilde{q} = \sqrt{\tilde{u}^2 + \tilde{v}^2} \tag{2.72a}$$

and

$$\tan\theta = \frac{\tilde{v}}{\tilde{u}} \tag{2.72b}$$

and conversely

$$\tilde{u} = \tilde{q}\cos\theta \tag{2.73a}$$

and

$$\tilde{v} = \tilde{q}\sin\theta \tag{2.73b}$$

Substituting these results into the continuity equation (the first component of (2.64)) yields

$$\frac{\partial\rho}{\partial t} + \left[\cos\theta\frac{\partial}{\partial s} - \sin\theta\frac{\partial}{\partial n}\right](\tilde{q}\cos\theta) + \left[\sin\theta\frac{\partial}{\partial s} + \cos\theta\frac{\partial}{\partial n}\right](\tilde{q}\sin\theta) = 0 \tag{2.74}$$

which can be rearranged to give

$$\frac{\partial\rho}{\partial t} + \frac{\partial\tilde{q}}{\partial s} + \tilde{q}\frac{\partial\theta}{\partial n} = 0 \tag{2.75}$$

Similar substitutions into the x-momentum equation yields

$$\cos\theta\frac{\partial\tilde{q}}{\partial t}-\tilde{q}\sin\theta\frac{\partial\theta}{\partial t}+\cos\theta\frac{\partial}{\partial s}\left(\frac{\tilde{q}^2}{\rho}+p\right)-\sin\theta\frac{\tilde{q}^2}{\rho}\frac{\partial\theta}{\partial s}+\cos\theta\frac{\tilde{q}^2}{\rho}\frac{\partial\theta}{\partial n}-\sin\theta\frac{\partial p}{\partial n}=0 \tag{2.76}$$

and the y-momentum equation yields

$$\sin\theta\frac{\partial\tilde{q}}{\partial t}+\tilde{q}\cos\theta\frac{\partial\theta}{\partial t}+\sin\theta\frac{\partial}{\partial s}\left(\frac{\tilde{q}^2}{\rho}+p\right)+\cos\theta\frac{\tilde{q}^2}{\rho}\frac{\partial\theta}{\partial s}+\sin\theta\frac{\tilde{q}^2}{\rho}\frac{\partial\theta}{\partial n}+\cos\theta\frac{\partial p}{\partial n}=0 \tag{2.77}$$

Now, the streamwise momentum equation results from taking the linear combination $[(2.76)\times\cos\theta\ +\ (2.77)\times\sin\theta]$, yielding

$$\frac{\partial\tilde{q}}{\partial t}+\frac{\partial}{\partial s}\left(\frac{\tilde{q}^2}{\rho}+p\right)+\frac{\tilde{q}^2}{\rho}\frac{\partial\theta}{\partial n}=0 \tag{2.78}$$

and the normal momentum equation is formed by taking the linear combination $[-(2.76)\times\sin\theta\ +(2.77)\times\cos\theta]$, yielding

$$\frac{\partial\theta}{\partial t}+\frac{\tilde{q}}{\rho}\frac{\partial\theta}{\partial s}+\frac{1}{\tilde{q}}\frac{\partial p}{\partial n}=0 \tag{2.79}$$

The energy equation is derived in a manner similar to the continuity equation, or

$$\frac{\partial E}{\partial t}+\frac{\partial}{\partial s}\left[(E+p)\frac{\tilde{q}}{\rho}\right]+(E+p)\frac{\tilde{q}}{\rho}\frac{\partial\theta}{\partial n}=0 \tag{2.80}$$

It is interesting now to note the duality of the Euler equations in intrinsic coordinates (2.75), (2.78), (2.79), and (2.80) with the quasi-one-dimensional Euler equations, given by

$$\frac{\partial\rho}{\partial t}+\frac{\partial\tilde{q}}{\partial s}+\tilde{q}\frac{1}{A}\frac{dA}{ds}=0 \tag{2.81a}$$

$$\frac{\partial\tilde{q}}{\partial t}+\frac{\partial}{\partial s}\left(\frac{\tilde{q}^2}{\rho}+p\right)+\frac{\tilde{q}^2}{\rho}\frac{1}{A}\frac{dA}{ds}=0 \tag{2.81b}$$

and

$$\frac{\partial E}{\partial t}+\frac{\partial}{\partial s}\left[(E+p)\frac{\tilde{q}}{\rho}\right]+(E+p)\frac{\tilde{q}}{\rho}\frac{1}{A}\frac{dA}{ds}=0 \tag{2.81c}$$

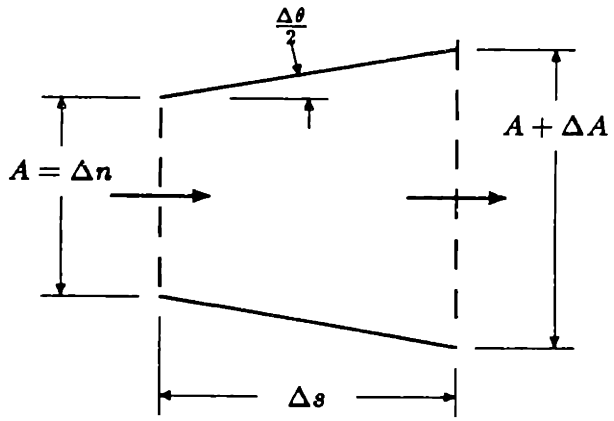


Figure 2.3: Elementary segment of streamtube.

The similarities between these two sets of equations should be expected if one considers a portion of a streamtube as shown in figure 2.3. The length of this elementary segment is Δs, the width at the inlet as A, and the width at the exit as $A + \Delta A$, resulting in a total included angle of $\Delta\theta = \frac{\Delta A}{\Delta s}$. The change in the coordinate normal to the flow direction from one streamtube to the next is simply Δn (which is equal to the area A). For a small angle, one can relate the included angle to the area change by

$$\frac{\partial\theta}{\partial n} \approx \frac{\Delta\theta}{\Delta n} = \frac{1}{\Delta n}\frac{\Delta A}{\Delta s} \approx \frac{1}{A}\frac{dA}{ds} \tag{2.82}$$

Thus, anywhere the term $\partial\theta/\partial n$ appears in the intrinsic-coordinate equations, the term $\frac{1}{A}\frac{dA}{ds}$ appears in the quasi-one-dimensional equations. Also for the equations in intrinsic coordinates, an additional equation, the normal momentum equation (2.79), results which describes the curvature of the two-dimensional streamtube.

2.5 Eigenanalysis and Characteristic Variables

Very often, the development of a numerical procedure for the solution of a set of coupled, hyperbolic differential equations requires knowledge of the eigenvalues and eigenvectors of the governing equation set. The existence of a complete set of real eigenvalues and a complete set of linearly independent eigenvectors is guaranteed if the governing equation are hyperbolic, as are the unsteady Euler equations. Furthermore, it is often desirable to diagonalize the set of governing equations, especially for the purpose of applying numerical boundary conditions.

In this section, the governing equations are expressed in terms of intrinsic coordinates. This set was chosen since, as will be seen, a principle direction must be chosen in order to perform the diagonalization. The only preferred direction one can choose, which is completely independent of any arbitrary coordinate system choice, is the streamline, or intrinsic coordinate direction.

First the eigenvalues and eigenvectors will be derived. Using these, the governing equations will then be diagonalized, resulting in the definition of the characteristic variables and equations.

2.5.1 Eigenvalues and eigenvectors

Begin with the intrinsic coordinate form of the Euler equations in the following vector notation

$$\frac{\partial \hat{\mathbf{U}}}{\partial t} + \hat{\mathbf{A}} \frac{\partial \hat{\mathbf{U}}}{\partial s} + \hat{\mathbf{B}} = 0 \tag{2.83}$$

where

$$\hat{\mathbf{U}} = \begin{pmatrix} \rho \\ \tilde{q} \\ \theta \\ E \end{pmatrix} \tag{2.84a}$$

$$\hat{\mathbf{A}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{\tilde{q}^2}{\rho^2} + \frac{\partial p}{\partial \rho} & 2\frac{\tilde{q}}{\rho} + \frac{\partial p}{\partial \tilde{q}} & 0 & \frac{\partial p}{\partial E} \\ 0 & 0 & \frac{\tilde{q}}{\rho} & 0 \\ -\left(\frac{E+p}{\rho}\right)\frac{\tilde{q}}{\rho} + \frac{\tilde{q}}{\rho}\frac{\partial p}{\partial \rho} & \frac{E+p}{\rho} + \frac{\tilde{q}}{\rho}\frac{\partial p}{\partial \tilde{q}} & 0 & \frac{\tilde{q}}{\rho}\left(1 + \frac{\partial p}{\partial E}\right) \end{pmatrix} \tag{2.84b}$$

and

$$\hat{\mathbf{B}} = \begin{pmatrix} \tilde{q}\frac{\partial \theta}{\partial n} \\ \frac{\tilde{q}^2}{\rho}\frac{\partial \theta}{\partial n} \\ \frac{1}{\tilde{q}}\frac{\partial p}{\partial n} \\ (E+p)\frac{\tilde{q}}{\rho}\frac{\partial \theta}{\partial n} \end{pmatrix} \tag{2.84c}$$

The solution of

$$\det[\hat{\mathbf{A}} - \lambda \mathbf{I}] = 0 \tag{2.85}$$

is used to determine the eigenvalues λ of $\hat{\mathbf{A}}$, where $\mathbf{I}$ is the identity matrix, and det indicates the determinant. Written out, equation (2.85) becomes

$$\det \begin{pmatrix} -\lambda & 1 & 0 & 0 \\ \frac{\gamma-3}{2}\frac{\tilde{q}^2}{\rho^2} & -(\gamma-3)\frac{\tilde{q}}{\rho} - \lambda & 0 & \gamma - 1 \\ 0 & 0 & \frac{\tilde{q}}{\rho} - \lambda & 0 \\ -\gamma\frac{E\tilde{q}}{\rho^2} + (\gamma-1)\frac{\tilde{q}^3}{\rho^3} & \gamma\frac{E}{\rho} - \frac{3}{2}(\gamma-1)\frac{\tilde{q}^2}{\rho^2} & 0 & \gamma\frac{\tilde{q}}{\rho} - \lambda \end{pmatrix} = 0 \tag{2.86}$$

and thus the eigenvalues of $\hat{\mathbf{A}}$ are

$$\lambda_1 = \frac{\tilde{q}}{\rho} - a, \quad \lambda_2 = \frac{\tilde{q}}{\rho}, \quad \lambda_3 = \frac{\tilde{q}}{\rho}, \quad \lambda_4 = \frac{\tilde{q}}{\rho} + a \tag{2.87}$$

where a is the speed of sound, derived previously.

Define $\mathbf{A}$ as the eigenvalue matrix, that is the matrix whose diagonal

elements are the eigenvalues of $\hat{\mathbf{A}}$ and whose other entries are 0, or

$$\mathbf{\Lambda} = \begin{pmatrix} \frac{\tilde{q}}{\rho} - a & 0 & 0 & 0 \\ 0 & \frac{\tilde{q}}{\rho} & 0 & 0 \\ 0 & 0 & \frac{\tilde{q}}{\rho} & 0 \\ 0 & 0 & 0 & \frac{\tilde{q}}{\rho} + a \end{pmatrix} \tag{2.88}$$

If the left eigenvectors of matrix $\hat{\mathbf{A}}$ are defined as the non-trivial solutions of the equation

$$L_k(\hat{\mathbf{A}} - \lambda_k \mathbf{I}) = 0 \tag{2.89}$$

then the eigenvector matrix $\mathbf{L}$, whose rows are the left eigenvectors, can be written as

$$\mathbf{L} = \begin{pmatrix} \frac{1}{2}\frac{\tilde{q}^2}{\rho^2} + \frac{\tilde{q}}{\rho}\frac{a}{\gamma-1} & -\frac{\tilde{q}}{\rho} - \frac{a}{\gamma-1} & 0 & 1 \\ \frac{\gamma+1}{2}\frac{\tilde{q}^2}{\rho^2} - \gamma\frac{E}{\rho} & -\frac{\tilde{q}}{\rho} & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \frac{1}{2}\frac{\tilde{q}^2}{\rho^2} - \frac{\tilde{q}}{\rho}\frac{a}{\gamma-1} & -\frac{\tilde{q}}{\rho} + \frac{a}{\gamma-1} & 0 & 1 \end{pmatrix} \tag{2.90}$$

Any matrix whose rows differ from the above matrix by only a multiplicative factor is also a suitable eigenvector matrix.

2.5.2 Characteristic variables

In this section, the changes in the characteristic variables will be determined from the eigenanalysis of the previous section. Begin with the diagonalization property of the eigenvectors, namely

$$\mathbf{L}\hat{\mathbf{A}} = \mathbf{\Lambda}\mathbf{L} \tag{2.91}$$

Premultiplying (2.83) by $\mathbf{L}$ yields

$$\mathbf{L}\frac{\partial \hat{\mathbf{U}}}{\partial t} + \mathbf{L}\hat{\mathbf{A}}\frac{\partial \hat{\mathbf{U}}}{\partial s} + \mathbf{L}\hat{\mathbf{B}} = 0 \tag{2.92}$$

or after using (2.91)

$$\mathbf{L}\frac{\partial \hat{\mathbf{U}}}{\partial t} + \mathbf{\Lambda}\mathbf{L}\frac{\partial \hat{\mathbf{U}}}{\partial s} + \mathbf{L}\hat{\mathbf{B}} = 0 \tag{2.93}$$

If one interprets $\mathbf{L}$ as the Jacobian of some vector $\mathbf{C}$ with respect to $\hat{\mathbf{U}}$, that is

$$d\mathbf{C} = \frac{\partial \mathbf{C}}{\partial \hat{\mathbf{U}}}\, d\hat{\mathbf{U}} = \mathbf{L}\, d\hat{\mathbf{U}} \tag{2.94}$$

then equation (2.93) can be written in the form

$$\frac{\partial \mathbf{C}}{\partial t} + \mathbf{\Lambda}\frac{\partial \mathbf{C}}{\partial s} + \mathbf{L}\hat{\mathbf{B}} = 0 \tag{2.95}$$

The vector $\mathbf{C}$ contains the characteristic variables. Notice that equation (2.95) is a diagonal equation, that is each row of the equation is uncoupled from the other if one assumes that $\mathbf{L}$ is a constant matrix. In the following, $\mathbf{L}$ will be taken to be frozen.

Based upon (2.94), one can calculate the change in the characteristic variables as

$$d\mathbf{C} = \begin{pmatrix} \left[\frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} + \frac{\tilde{q}a}{\rho}\right] d\rho + \left[-(\gamma-1)\frac{\tilde{q}}{\rho} - a\right] d\tilde{q} + [\gamma-1]\, dE \\ \left[\frac{\gamma+1}{2}\frac{\tilde{q}^2}{\rho^2} - \gamma\frac{e}{\rho}\right] d\rho + \left[-\frac{\tilde{q}}{\rho}\right] d\tilde{q} + [1]\, dE \\ [1]\, d\theta \\ \left[\frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} - \frac{\tilde{q}a}{\rho}\right] d\rho + \left[-(\gamma-1)\frac{\tilde{q}}{\rho} + a\right] d\tilde{q} + [\gamma-1]\, dE \end{pmatrix} \tag{2.96}$$

Most often, the change in characteristic variables is given in terms of primitive variables (density ρ, velocity q, and pressure p) instead of in terms of the conservation variables as above. To convert the above to primitive variable form, start by using the definition of the pressure

$$p = (\gamma-1)\left[E - \frac{1}{2}\frac{\tilde{q}^2}{\rho}\right] \tag{2.97}$$

from which the change in pressure can be seen to be

$$dp = (\gamma-1)\left[dE + \frac{1}{2}\frac{\tilde{q}^2}{\rho^2}\, d\rho - \frac{\tilde{q}}{\rho}\, d\tilde{q}\right] \tag{2.98}$$

The velocity magnitude is defined by

$$q = \frac{\tilde{q}}{\rho} \tag{2.99}$$

and its change by

$$dq = -\frac{\tilde{q}}{\rho^2}\, d\rho + \frac{1}{\rho}\, d\tilde{q} \tag{2.100}$$

Substituting (2.98) and (2.100) into (2.96) and using the definition of the speed of sound yields an alternative and more popular expression for the change in characteristic variables

$$d\mathbf{C} = \begin{pmatrix} -dq + \frac{dp}{\rho a} \\ d\rho - \frac{dp}{a^2} \\ d\theta \\ dq + \frac{dp}{\rho a} \end{pmatrix} \tag{2.101}$$

2.6 Boundary Conditions

The problems of interest in this thesis all consist of two-dimensional bodies moving at a constant velocity in an unbounded medium. To close the problem analytically, boundary conditions are needed both at the body surfaces and in the far-field. In this section, a description of each of these conditions is given.

2.6.1 Solid surface

For inviscid flow (Assumption 2.2), the appropriate boundary conditions on a solid surface is that there be no flow through the surface, or

$$\vec{M} \cdot \vec{n} = 0 \tag{2.102}$$

where $\vec{n}$ is the unit normal outward from the solid surface. This is analytically equivalent to the requirement that the flow direction be tangent to the solid surface, making it a streamline.

This condition holds both for straight and curved surfaces. However, at slope discontinuities or corners, there is no unique normal direction and this condition cannot be rigorously applied. An example of such a point is at a sharp trailing edge which is the subject of the next section.

2.6.2 Trailing edge

It is well known that the flow about closed bodies which are moving at a constant subsonic velocity in an unbounded medium cannot be determined uniquely by a purely inviscid analysis. The inviscid Assumption (2.2) allows an infinity of solutions for this problem due to the unspecified circulation about the body. But once the circulation is specified, the flow is uniquely specified and the locations of the stagnation points on the body can be determined. To determine which of these solutions is physically realizable one must take into account viscous effects, even for vanishingly small viscosities. A similar analysis to the one which follows has been published by Melnik[62] for high Reynolds-number airfoils.

The inviscid flow in the immediate vicinity of a sharp corner (for example, a sharp trailing edge) can be predicted for incompressible flow by conformally mapping a straight wall into one with a corner. The resulting velocity distribution along the wall is given by

$$q \propto r^{\left(\frac{\pi-\alpha}{\alpha}\right)} \tag{2.103}$$

where q is the velocity along the wall, r is the distance from the corner, and α is the change in flow direction at the corner. This expression can be extended to subcritical compressible flows by applying a compressibility correction such as Prandtl-Glauert.

Consider now an airfoil which has a subsonic flow in the vicinity of its sharp trailing edge. Suppose that the circulation about the body is such that the rearward stagnation point is not at the trailing edge, thus requiring the flow to pass around the sharp trailing edge. The change in flow

angle at the trailing edge would be $\alpha > \pi$, resulting in an infinite trailing edge velocity by (2.103). The physical viscosity, no matter how small, precludes such a situation. On the basis of this, and as a result of experimental observations, Kutta (and independently Joukowski) hypothesized that the circulation about the airfoil must adjust itself such that a rearward stagnation point be located at the trailing edge, resulting in a flow which smoothly separates from the trailing edge.

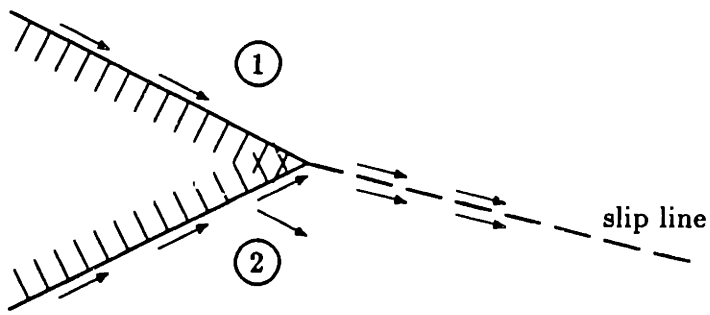


Figure 2.4: Airfoil trailing edge.

The trailing edge behavior can be predicted by viewing the local flow as the merging of two streams as shown in figure 2.4. The general solution as sketched in the figure is that a slip line emanates from the trailing edge, separating the upper and lower flows which do not mix since the flow is inviscid. The two conditions which must hold along the slip line are that there be no pressure jump across the slip line and that the flow be everywhere tangent to the slip line. For the purposes of this analysis, the trailing edge region will be treated as incompressible, allowing the use of Bernoulli's equation

$$p_o = p + \frac{1}{2}\rho q^2 \tag{2.104}$$

or

$$q = \sqrt{\frac{p_o - p}{\rho/2}} \tag{2.105}$$

where p_o is the stagnation pressure. This analysis can be extended to compressible, subsonic flows by replacing the density, ρ, in the above expression with some average density.

There are two cases which will be considered here. In Case I, the stagnation pressures of the upper and lower flows will be equal, such as in isentropic flows. This case, which has been the subject of most classical treatments, is the one which is consistent with the potential approximation. Case II is for trailing edges which have different upper and lower stagnation pressures. Such a case might arise for an airfoil when a strong shock on one of its surfaces causes a decrease in the stagnation pressure from its free-stream value.

In both cases, just upstream of the trailing edge the two flows are tangent to their respective walls and hence have unequal flow directions. Just downstream of the trailing edge however, the two streams must have the same inclination along the slip line, requiring that either one or both streams change direction instantaneously at the trailing edge. The only way to instantaneously change direction in subsonic flow without having an infinite velocity gradient (which is not physically realizable) is for the flow to stagnate.

First consider Case I in which the two streams have the same stagnation pressure, $(p_o)_1 = (p_o)_2$. Assume for the moment that the upper flow changes direction at the trailing edge, and hence stagnates. Equation (2.104) can then be applied, resulting in a trailing edge pressure $p_{te} = (p_o)_1$. Then since $p_{te} = (p_o)_2$, equation (2.105) yields $(q_{te})_2 = 0$ and hence the lower flow stagnates too. From this it follows that the direction that the slip line leaves the trailing edge is between the tangents to the upper and lower surfaces. Its exact inclination can be determined by considering the pressure at the

slip line just downstream of the trailing edge. As seen from (2.103), the pressure is simply a function of the distance from the corner and the flow angle change at the corner. For the pressures to be equal along the slip line, the angle change for the two streams must be the same, or in other words the flow leaves the trailing edge at the bisector of the trailing edge angle. This result could have been expected based purely on the symmetry of the problem.

Now consider the more difficult case, Case II, where the two streams have different stagnation pressures such that $(p_o)_1 > (p_o)_2$. Again either one or both flows must change direction instantaneously at the trailing edge. Assume that the upper flow (1) changes direction and hence stagnates. By (2.104), it can be seen that $p_{te} = (p_o)_1$. Substituting into (2.105) for the velocity of the lower flow at the trailing edge yields $q_2 = \sqrt{\frac{(p_o)_2 - (p_o)_1}{\rho/2}}$. But since $(p_o)_1 > (p_o)_2$, there is no real solution, contradicting the assumption that the upper flow changes direction.

Alternatively assume that the lower flow (2) stagnates at the trailing edge. By equation (2.104), it can be seen that $p_{te} = (p_o)_2$ and by (2.105) that $q_1 = \sqrt{\frac{(p_o)_1 - (p_o)_2}{\rho/2}} > 0$. Therefore the upper flow (1) does not stagnate and hence the slip line must leave the trailing edge parallel to the upper surface. Immediately upon leaving the trailing edge, the flow turns toward the trailing edge bisector; thus a small distance downstream, the flows in Case I and II follow nearly the same trajectory.

Now treat the case of a trailing edge embedded in locally supersonic flow. Again the general solution is that of two flows merging and being separated by a slip line emanating from the trailing edge point as shown in figure 2.4. The flows have different inclinations just upstream of the trailing edge and one or both must change direction instantaneously at the trailing edge so that they have equal slopes downstream. But in contrast to subsonic flows where this direction change can only occur at a stagnation point (which has

upstream influence), the supersonic flow can change direction through either an oblique shock or a Prandtl-Meyer expansion fan, neither of which create an upstream influence. Hence the pressure distribution on the body, and as a result the forces on the body, are not affected either by the trailing edge point or by the flow on the opposing surface.

To determine the downstream pressure and the angle that the slip line leaves the trailing edge, one must consider both the Mach number and stagnation pressure for both upper and lower flows upstream of the trailing edge. Downstream of a corner, the pressure is a constant with its value given by

$$p = \begin{cases} f_{sh}(M_{up}, p_{o_{up}}, \delta) & \delta > 0 \text{ (oblique shock)} \\ f_{ex}(M_{up}, p_{o_{up}}, \delta) & \delta < 0 \text{ (expansion fan)} \end{cases} \tag{2.106}$$

where δ is the angle change ($\delta > 0$ is concave). This equation can be applied independently to both the upper (1) and lower (2) flows. Two conditions, namely that the sum of the angle changes is equal to the trailing edge angle and that the pressures on both sides of the slip line be equal, can be used then to determine uniquely the slip line inclination and the downstream pressure.

In summary for subsonic trailing edge flows, the airfoil circulation is set so that a stagnation point exists at the trailing edge, either on one or both sides of a slip line emanating from the trailing edge point. Which one of these cases exists and the angle that the slip line leaves the trailing edge is determined solely by the relative stagnation pressures of the upper and lower surface flows. Thus the pressure increase on the airfoil surface(s) just upstream of the trailing edge, and hence the forces on the body, is influenced by a communication between the two flows around the trailing edge. This is in contrast to supersonic trailing edge flows where the pressure is only affected by the flow in its upstream Mach cone. Hence, there is no influence of either the trailing edge or the other surface on a surface's pressure distribution and consequently the forces on the body.

2.6.3 Far-field boundary

The true far-field boundary condition for two-dimensional bodies moving at a constant velocity in an unbounded medium is quite simple, namely that the flow is uniform at an infinite distance from the body. Unfortunately, this boundary condition cannot in general be used directly since numerical simulations are always limited to finite domains. In this section, an approximate far-field model will be developed for a boundary placed arbitrarily at some finite distance from the body, somewhat following the analysis by Ludford[58] and Cole[24]. Then the effect of this approximation on the body solution will be examined in order to determine the adequacy of the approximation.

Initially consider a body moving with a subsonic free-stream Mach number, $M_\infty < 1$. At any point in the flow field, the flow quantities can be determined by combining the influence of the free-stream flow and the disturbances which emanate outward from the body. Define the far-field as the region where the perturbations of the flow quantities from their free-stream values are small, or

$$\begin{aligned} \tfrac{u-U_\infty}{U_\infty} &\ll 1 \qquad \tfrac{v}{U_\infty} \ll 1 \\ \tfrac{p-p_\infty}{p_\infty} &\ll 1 \qquad \tfrac{\rho-\rho_\infty}{\rho_\infty} \ll 1 \end{aligned} \tag{2.107}$$

where the ∞ subscript denotes free-stream conditions. Here the Cartesian coordinates (x, y) has been set up centered on the body with the x-axis aligned with the free-stream velocity, U_∞. This small disturbance requirement automatically restricts the far-field to the region outside local supersonic regions (and shocks) attached to the body.

Two assumptions which are needed to specify the far-field behavior are given as:

Assumption 2.7 *The far-field flow is homenthalpic, that is the stagnation enthalpy, h_o, is constant. This assumption is valid for an adiabatic flow (Assumption 2.4) with a uniform upstream stagnation enthalpy.*

Assumption 2.8 *The far-field flow is homentropic, that is the entropy is constant. For flows with uniform upstream entropy, this assumption holds, except possibly in a wake downstream of the body. The source of the wake's entropy increase can be either boundary layer induced (which is not the case here by the inviscid Assumption 2.2) or shock induced (which is quite possible for arbitrary bodies moving at transonic speeds). It has been shown by Garabedian[37] that the isentropic assumption can still be used for weak shocks if one instead models the effect of the shocks as a mass source rather than an entropy source. This is the approach that will be taken here.*

Given the above assumptions, the far-field disturbances are governed by the linearized small-disturbance (Prandtl-Glauert) equation

$$(1 - M_\infty^2)\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \tag{2.108}$$

where the potential function ϕ corresponds to the perturbations emanating from the body. The corresponding Cartesian velocities (u and v) are given by the sum of the free-stream and perturbation velocities, which in terms of the perturbation potential becomes

$$\frac{u}{U_\infty} = 1 + \frac{\partial \phi}{\partial x}, \qquad \frac{v}{U_\infty} = \frac{\partial \phi}{\partial y} \tag{2.109}$$

To determine the density, ρ, and the pressure, p, in the far-field, the state equation (2.39) is to be used to relate the pressure and density, or

$$\frac{p}{\rho} = \frac{\gamma - 1}{\gamma}\left[h_o - \frac{1}{2}u^2 - \frac{1}{2}v^2\right] \tag{2.110}$$

where h_o is constant by the homenthalpic assumption. The other condition which will be used in the far-field is that the flow is isentropic (as discussed above), which in terms of the density and pressure yields

$$\frac{p}{\rho^\gamma} = \text{constant} \tag{2.111}$$

Equation (2.110) and (2.111) can be combined to yield

$$\left(\frac{\rho}{\rho_\infty}\right)^{\gamma-1} = (\gamma-1)M_\infty^2\left[\frac{h_o}{U_\infty^2} - \frac{1}{2}\left(\frac{u}{U_\infty}\right)^2 - \frac{1}{2}\left(\frac{v}{U_\infty}\right)^2\right] \tag{2.112}$$

where use has been made of equation (2.51) and the definition of the free-stream Mach number $M_\infty = \frac{U_\infty}{a_\infty}$.

Substituting (2.109) into (2.112), and recalling that the perturbations are small leads to the expression

$$\frac{\rho}{\rho_\infty} = 1 - M_\infty^2\frac{\partial\phi}{\partial x} \tag{2.113}$$

which can be substituted into (2.111) to further yield

$$\frac{p}{\rho_\infty U_\infty^2} = \frac{1}{\gamma M_\infty^2} - \frac{\partial\phi}{\partial x} \tag{2.114}$$

Thus all four far-field quantities (ρ, u, v, and p) can be expressed in terms of the perturbation velocities $\frac{\partial\phi}{\partial x}$ and $\frac{\partial\phi}{\partial y}$.

The perturbation potential is computed using equation (2.108), which can be transformed into Laplace's equation

$$\frac{\partial^2\phi}{\partial\check{x}^2} + \frac{\partial^2\phi}{\partial\check{y}^2} = 0 \tag{2.115}$$

by applying the affine transformation

$$\check{x} = x, \quad \check{y} = \beta y, \quad \text{where } \beta^2 = 1 - M_\infty^2 \tag{2.116}$$

Since the analytic solution is obtained by applying the boundary conditions infinitely far from the body, it is convenient to transform (2.115) into the polar coordinate system (r,θ), defined by

$$r = \sqrt{\check{x}^2 + \check{y}^2}, \quad \theta = \arctan\frac{\check{y}}{\check{x}} \tag{2.117}$$

In terms of (r,θ), equation (2.115) becomes

$$\frac{\partial^2\phi}{\partial r^2} + \frac{1}{r}\frac{\partial\phi}{\partial r} + \frac{1}{r^2}\frac{\partial^2\phi}{\partial\theta^2} = 0 \tag{2.118}$$

whose general solution, which can be found by separation of variables, is given by[45, pg 434]

$$phi = a_0 + b_0 \ln r + c_0\theta + d_0\theta \ln r + \sum_{n=1}^{\infty} \left[(a_n r^n + b_n r^{-n}) \cos n\theta + (c_n r^n + d_n r^{-n}) \sin n\theta\right] \quad (2.119)$$

Here it is assumed that the expansions in θ are trigonometric (rather than exponential) and that they have a common period of 2π.

The boundary conditions which are to be applied to the perturbation potential ϕ are that the disturbances due to the body vanish at infinitely large distances from the body, or

$$\left.\begin{array}{l} \frac{\partial \phi}{\partial r} \to 0 \\ \frac{1}{r}\frac{\partial \phi}{\partial \theta} \to 0 \end{array}\right\} \text{ as } r \to \infty \quad (2.120)$$

Taking the derivatives of (2.119) yields

$$\frac{\partial \phi}{\partial r} = b_0 \frac{1}{r} + d_0 \frac{\theta}{r} + \sum_{n=1}^{\infty} \left[(+n a_n r^{n-1} - n b_n r^{-n-1}) \cos n\theta + (n c_n r^{n-1} + n d_n r^{-n-1}) \sin n\theta\right] \quad (2.121a)$$

and

$$\frac{1}{r}\frac{\partial \phi}{\partial \theta} = c_0 \frac{1}{r} + d_0 \frac{\ln r}{r} + \sum_{n=1}^{\infty} \left[(-n a_n r^{n-1} - n b_n r^{-n-1}) \sin n\theta + (n c_n r^{n-1} + n d_n r^{-n-1}) \cos n\theta\right] \quad (2.121b)$$

The boundary conditions (2.120), when combined with (2.121), yields

$$a_n = c_n = 0 \quad n = 1, \ldots, \infty \quad (2.122)$$

and since the perturbation velocity $\frac{\partial \phi}{\partial r}$ must be single-valued, (2.121) further yields

$$d_0 = 0 \quad (2.123)$$

Substituting (2.122) and (2.123) into (2.119) yields the general solution for the perturbation potential associated with an arbitrary body

$$\phi = a_0 + b_0 \ln r + c_0\theta + \sum_{n=1}^{\infty}\left[\frac{1}{r^n}(b_n \cos n\theta + d_n \sin n\theta)\right] \tag{2.124}$$

The leading terms in (2.124) can be identified as the so-called elementary solutions: a_0 represents an arbitrary constant which is of no consequence in determining the flow-field velocities and can therefore be set to 0; $b_0 \ln r$ represents a mass source; $c_0\theta$ represents a potential vortex; $b_1\frac{\cos\theta}{r}$ represents a source doublet; and $d_1\frac{\sin\theta}{r}$ represents a vortex doublet. For the analysis which follows, only the leading order terms (source and vortex) will be considered, with their coefficients written in the more conventional manner $b_0 = \frac{\Sigma}{2\pi}$ and $c_0 = -\frac{\Gamma}{2\pi}$.

Written in terms of the physical Cartesian coordinates, the perturbation potential is given by

$$\phi = \frac{\Sigma}{4\pi}\ln(x^2 + \beta^2 y^2) - \frac{\Gamma}{2\pi}\arctan\left(\beta\frac{y}{x}\right) \tag{2.125}$$

with its derivatives

$$\frac{\partial\phi}{\partial x} = \frac{\Sigma}{2\pi}\frac{x}{x^2 + \beta^2 y^2} + \frac{\Gamma}{2\pi}\frac{\beta y}{x^2 + \beta^2 y^2} \tag{2.126a}$$

and

$$\frac{\partial\phi}{\partial y} = \frac{\Sigma}{2\pi}\frac{\beta^2 y}{x^2 + \beta^2 y^2} - \frac{\Gamma}{2\pi}\frac{\beta x}{x^2 + \beta^2 y^2} \tag{2.126b}$$

Note that the only unknown quantities at this point are the source and vortex strengths whose values depend on the specific body shape and inclination (with respect to the free-stream). Various approaches for calculating these strengths have been given in the literature, some of which will be discussed here.

Wubs *et al*[90] treated the incompressible potential flow around an airfoil by a zonal technique where the inner solution was calculated by a finite difference solution of Laplace's equation and the outer region was an analytic

series representation in the form of (2.124). The source strength was set to zero since physically there is no mass created in or around the airfoil. Physical reasoning was also used to link the vortex strength to a Kutta condition at the airfoil trailing edge. The higher order terms, $b_1, d_1, b_2, d_2, \ldots$ are determined such that the potential and its normal derivative match (in a least-squares sense) at the far-field boundary. They found that for the same accuracy, the domain size could be reduced if more terms were used in the analytic far-field expansion. But for far-field boundaries at about 2.5 chords from the airfoil, the solution holding only the vortex term (ignoring the doublet and higher order terms $b_1, d_1, b_2, \ldots$) was in error by less than 1.3 percent. Therefore, for transonic calculations the far-field boundary is generally placed at least 5 chords from the body in order to ensure that any local supersonic regions are complete enclosed, it follows that the higher order terms can be ignored.

Usab[84] treated the transonic airfoil problem, with the inner region computed by a finite-volume solution of the Euler equations and the outer analytic part using the small disturbance form of the potential equation. The outer analytic expansion consisted only of a uniform flow and a compressible vortex, whose strength, Γ, was linked to the lift coefficient which was computed through a pressure intergration on the airfoil surface. This link, which can be rigorously established for shockless (isentropic) flows, was extended to transonic problems with shock induced entropy wakes by concluding that far from the airfoil, there is no net vorticity in the wake even though the wake is rotational. The only limitation of the extension is that the far-field vortex strength must be computed from the airfoil lift and not its circulation since the circulation changes between the airfoil surface and the far-field due to the vorticity produced at the shock. Thomas and Salas[79] repeated Usab's analysis from a slightly different point of view, but ended up with the same far-field expansions.

Here, a different approach is taken to determine the appropriate free constants in the perturbation potential equation, (2.125). The disturbances caused by the body are modelled by those of an equivalent body composed of a vortex and source. The doublet and higher order effects are neglected as suggested by Wubs' results. The momentum equation is applied to a closed circuit around the body (in the far-field) to determine the relationships, if any, between the forces on the body and the source and vortex strengths of the equivalent body. Once these strengths are known, the far-field flow quantities can be written directly in terms of the forces on the body (determined by integrating the pressure), without regard for the body shape or the distribution of pressure on the body.

Consider an elliptical control volume around the body being analyzed, with a radius large enough so as to surround the body (or bodies) and any local supersonic regions. The ellipse can be defined parametrically by

$$x = R\cos\theta, \qquad y = \frac{R}{\beta}\sin\theta, \qquad 0 \le \theta \le 2\pi \tag{2.127}$$

which is simply a circle in the transformed polar coordinate system (r, θ) defined above.

The momentum equations (2.45) can be applied to this control volume, to yield

$$F_x = L = -\oint (\rho u^2 + p)\, dy - (\rho uv)\, dx \tag{2.128a}$$

and

$$F_y = D = -\oint (\rho uv)\, dy - (\rho v^2 + p)\, dx \tag{2.128b}$$

where the body forces F_x and F_y must be included. Since the free-stream is aligned with the x-axis, these forces are simply the drag D and lift L respectively on the body.

Substituting (2.109), (2.113) and (2.114) into (2.128) yields

$$F_x = D = -\rho_\infty U_\infty^2 \oint \left\{ \left(1 - M_\infty^2 \frac{\partial \phi}{\partial x}\right) \left(1 + \frac{\partial \phi}{\partial x}\right)^2 + \left(\frac{1}{\gamma M_\infty^2} - \frac{\partial \phi}{\partial x}\right) \right\} dy$$

$$- \left\{ \left(1 - M_\infty^2 \frac{\partial \phi}{\partial x} \right) \left(1 + \frac{\partial \phi}{\partial x} \right) \left(\frac{\partial \phi}{\partial y} \right) \right\} dx \tag{2.129a}$$

and

$$F_y = L = -\rho_\infty U_\infty^2 \oint \left\{ \left(1 - M_\infty^2 \frac{\partial \phi}{\partial x} \right) \left(1 + \frac{\partial \phi}{\partial x} \right) \left(\frac{\partial \phi}{\partial y} \right) \right\} dy$$
$$- \left\{ \left(1 - M_\infty^2 \frac{\partial \phi}{\partial x} \right) \left(\frac{\partial \phi}{\partial y} \right)^2 + \left(\frac{1}{\gamma M_\infty^2} - \frac{\partial \phi}{\partial x} \right) \right\} dx \tag{2.129b}$$

Again since the perturbations are small, these reduce to

$$F_x = D = -\rho_\infty U_\infty^2 \oint \left\{ \left(1 + \frac{1}{\gamma M_\infty^2} \right) + \beta^2 \frac{\partial \phi}{\partial x} \right\} dy - \left\{ \frac{\partial \phi}{\partial y} \right\} dx \tag{2.130a}$$

and

$$F_y = L = -\rho_\infty U_\infty^2 \oint \left\{ \frac{\partial \phi}{\partial y} \right\} dy - \left\{ \frac{1}{\gamma M_\infty^2} - \frac{\partial \phi}{\partial x} \right\} dx \tag{2.130b}$$

Substituting (2.126) into (2.130) and carrying out the integration gives

$$F_x = D = -\rho_\infty U_\infty^2 \beta \Sigma \tag{2.131a}$$

and

$$F_y = L = +\rho_\infty U_\infty^2 \Gamma \tag{2.131b}$$

Now define the lift and drag coefficients as

$$C_L = \frac{L}{\frac{1}{2}\rho_\infty U_\infty^2 c}, \qquad C_D = \frac{D}{\frac{1}{2}\rho_\infty U_\infty^2 c} \tag{2.132}$$

where c is a reference length, usually taken to be the airfoil chord length. Substituting (2.131) into (2.132) finally yields the source and vortex strengths

$$\Sigma = -\frac{cC_D}{2\beta} \tag{2.133a}$$

and

$$\Gamma = +\frac{cC_L}{2} \tag{2.133b}$$

The far-field conditions in terms of the lift and drag coefficients on the body are finally given by

$$\frac{\rho}{\rho_\infty} = 1 + M_\infty^2 \left\{ \frac{C_D}{4\pi\beta(r/c)} \cos\theta - \frac{C_L}{4\pi(r/c)} \sin\theta \right\} \tag{2.134a}$$

$$\frac{u}{U_\infty} = 1 - \frac{C_D}{4\pi\beta(r/c)} \cos\theta + \frac{C_L}{4\pi(r/c)} \sin\theta \tag{2.134b}$$

$$\frac{v}{U_\infty} = -\frac{C_D}{4\pi(r/c)} \sin\theta - \frac{\beta C_L}{4\pi(r/c)} \cos\theta \tag{2.134c}$$

and

$$\frac{p}{\rho_\infty U_\infty^2} = \frac{1}{\gamma M_\infty^2} + \frac{C_D}{4\pi\beta(r/c)} \cos\theta - \frac{C_L}{4\pi(r/c)} \sin\theta \tag{2.134d}$$

where (for reference)

$$\left(\frac{r}{c}\right)^2 = \left(\frac{x}{c}\right)^2 + \beta^2 \left(\frac{y}{c}\right)^2, \quad \theta = \arctan\left(\beta \frac{y}{x}\right) \tag{2.135}$$

It can be noted that if one ignores the source terms (those containing a C_D), the above expressions are the same as those derived by Usab.

The origin of the source term in the above expressions is not actual mass being emitted from the airfoil, but rather as a result of the shock modelling. By describing the flow field behavior by a single scalar, the velocity potential (ϕ), it is not possible to ensure that both mass and momentum are conserved at the shock. Enforcing the over-all conservation of momentum (by applying the momentum equation in the far-field), implicitly models the shock as a series of mass sources. In other words, the entropy blockage is modelled as a mass blockage through these sources. Drela[31] also uses a source term in the far-field expansion for airfoils to account for the blockage caused by shock waves. He determines the source strength Σ directly from the difference between the actual mass flux in the wake and the mass flux of an equivalent irrotational, isentropic flow.

It is interesting to note that for shock-free flows, the inviscid drag is identically zero and hence the source terms in (2.134) vanish, yielding the same far-field expression as derived by Usab.

Now that the far-field has been described in terms of a uniform flow plus a source and vortex, it remains to quantify the importance of these latter terms in the far-field representation. This can be done by comparing predictions employing the source and vortex terms in the far-field with those which result from a uniform far-field approximation.

Assume that the accuracy of an airfoil calculation can be characterized by the accuracy of its lift prediction. An error term, ϵ_L, can be defined as the fractional lift difference between the two predictions, or

$$\epsilon_L = \frac{L_{\text{fvs}} - L_{\text{f}}}{L_{\text{f}}} \tag{2.136}$$

where L_{fvs} denotes the lift predicted using a free-stream-plus-vortex-plus-source representation for the far-field and L_{f} denotes the lift predicted using a free-stream representation.

The lift generated by an airfoil is a first-order function of its shape, its angle of attack, and the dynamic pressure of the upstream flow. In recognition of this fact, it is traditional to write the lift as

$$L = C_L \frac{1}{2} \rho q^2 c \tag{2.137}$$

where the airfoil shape and angle of attack determine the lift coefficient, C_L, with the dynamic pressure given by $\frac{1}{2}\rho q^2$. Substituting (2.137) into (2.136) yields

$$\epsilon_L = \frac{(C_L)_{\text{fvs}}}{(C_L)_{\text{f}}} \frac{\left(\frac{1}{2}\rho q^2\right)_{\text{fvs}}}{\left(\frac{1}{2}\rho q^2\right)_{\text{f}}} - 1 \tag{2.138}$$

For most airfoils operating below stall and the drag rise Mach number, the lift coefficient is linearly related to the angle of attack, α, or

$$(C_L)_{\text{fvs}} - (C_L)_{\text{f}} = \frac{dC_L}{d\alpha} (\alpha_{\text{fvs}} - \alpha_{\text{f}}) \tag{2.139}$$

Dividing (2.139) by $(C_L)_{\text{f}}$, and using the fact that for most airfoils, the lift-curve slope $\frac{dC_L}{d\alpha}$ is very close to the flat plate value of 2π yields

$$\frac{(C_L)_{\text{fvs}}}{(C_L)_{\text{f}}} = 1 + \frac{2\pi (\alpha_{\text{fvs}} - \alpha_{\text{f}})}{(C_L)_{\text{f}}} \tag{2.140}$$

Now assume that the airfoil acts primarily based upon the air along the upstream stagnation streamline, and further assume that the stagnation streamline lies along the x-axis. Therefore, one can determine the airfoil's angle of attack by evaluating (2.134) along the line $\theta = \pi$, yielding

$$\alpha_{fvs} - \alpha_f = \arctan\left(\frac{v}{u}\right)_{(\theta=\pi)} - \arctan\frac{0}{U_\infty} \tag{2.141}$$

which for small angles yields

$$\alpha_{fvs} - \alpha_f = \frac{\beta C_L}{4\pi(r/c)} \tag{2.142}$$

which can be substituted into (2.140) to yield

$$\frac{(C_L)_{fvs}}{(C_L)_f} = 1 + \frac{\beta}{2(r/c)} \tag{2.143}$$

In a similar way, the ratio of dynamic pressures is given by

$$\frac{\left(\frac{1}{2}\rho q^2\right)_{fvs}}{\left(\frac{1}{2}\rho q^2\right)_f} = \left(\frac{\rho}{\rho_\infty}\right)_{(\theta=\pi)} \left(\frac{u}{U_\infty}\right)^2_{(\theta=\pi)} + \left(\frac{\rho}{\rho_\infty}\right)_{(\theta=\pi)} \left(\frac{v}{U_\infty}\right)^2_{(\theta=\pi)} \tag{2.144}$$

which after using (2.134) and dropping higher order terms yields

$$\frac{\left(\frac{1}{2}\rho q^2\right)_{fvs}}{\left(\frac{1}{2}\rho q^2\right)_f} = 1\frac{(1+\beta^2)C_D}{4\pi(r/c)} \tag{2.145}$$

Substituting equations (2.143) and (2.145) into (2.138) and dropping the higher order terms yields the final expression

$$\epsilon_L = \underbrace{\frac{\beta}{2(r/c)}}_{\text{vortex}} + \underbrace{\frac{(1+\beta^2)C_D}{4\pi\beta(r/c)}}_{\text{source}} \tag{2.146}$$

Here it can be seen that the source predominates only if $C_D > \frac{2\pi\beta^2}{1+\beta^2}$ which never occurs for a reasonably well designed airfoil. Therefore, the source term will be dropped for the remainder of this thesis.

It is also interesting to note the relative size of the errors introduced by dropping the vortex term as opposed to the higher order doublets. Recall that for an incompressible airfoil, Wubs found that for a far-field radius of $(r/c) = 2.5$, the error introduced by ignoring the doublet terms was about 1.3 percent. Equation (2.146) shows that the effect of ignoring the vortex for the same far-field radius is to incur about a 20 percent error.

Hence in the far-field, the vortex term has been demonstrated to be the dominant term in the far-field expansion, being at least an order of magnitude more important than either the source or doublet terms.

Chapter 3

Basic Numerical Integration Scheme

This chapter presents the scheme that is used to numerically integrate the governing equation derived in the previous chapter. This explicit time marching technique, originally developed by Ni[67], is a finite volume form of a one-step, Lax-Wendroff-type scheme. Since only the steady state solution is desired, various acceleration techniques are applied to the basic scheme which alter the time evolution of the solution, but which hasten convergence to the steady state.

This chapter begins with the development of the basic integration scheme, followed by a description of the numerical implementation of the boundary conditions. Once the scheme has been defined, its relevant properties of conservation, accuracy, consistency, and stability are determined. The scheme is then applied to a model one-dimensional equation, which leads to the development of a multiple-grid accelerator to enhance its convergence properties. The multiple-grid accelerated scheme is demonstrated again on the model equation and then on the quasi-one-dimensional Euler equations. The development and analysis of the smoothing which is needed to capture shocks is discussed next, with the convergence checking algorithm described in the final section.

The integration scheme is developed for a general system of first-order, hyperbolic partial differential equations, such as the Euler equations. Therefore, with the exception of some of the material pertaining to boundary conditions, the scheme can easily be applied to most other sets of governing equations of the above form, such as the wave equation or the Navier-Stokes equations.

3.1 Development of Integration Scheme

The unsteady governing equations to be solved comprise a set of first-order, non-linear, hyperbolic partial-differential equations written in conservation law form. In two dimensions, they are represented by the integral equation

$$\iint_C \frac{\partial \mathbf{U}}{\partial t}\, dx\, dy + \oint_{\partial C} (\mathbf{F}\, dy - \mathbf{G}\, dx) = 0 \tag{3.1}$$

and the corresponding differential equation

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = 0 \tag{3.2}$$

where the control volume C is fixed. Here $\mathbf{U}$ is the vector of dependent variables, t is the time, x and y are the Cartesian coordinates, and $\mathbf{F}$ and $\mathbf{G}$ (which are functions of $\mathbf{U}$) are the flux vectors in the x and y directions respectively. Equation 2.63 gives the definitions of $\mathbf{U}$, $\mathbf{F}$, and $\mathbf{G}$ for the unsteady Euler equations.

Since the equations have been formulated in conservation-law form, that is the coefficients of the derivative terms in (3.2) are constant, they are well suited to a finite volume discretization. In this spatial discretization technique, the flow field is divided into a set of non-overlapping volumes which span the entire domain. The conservation law described by the integral form of the governing equation is approximated on each volume, resulting in the statement that the change of the conserved quantity in the volume is equal to the net flux across the cell boundaries.

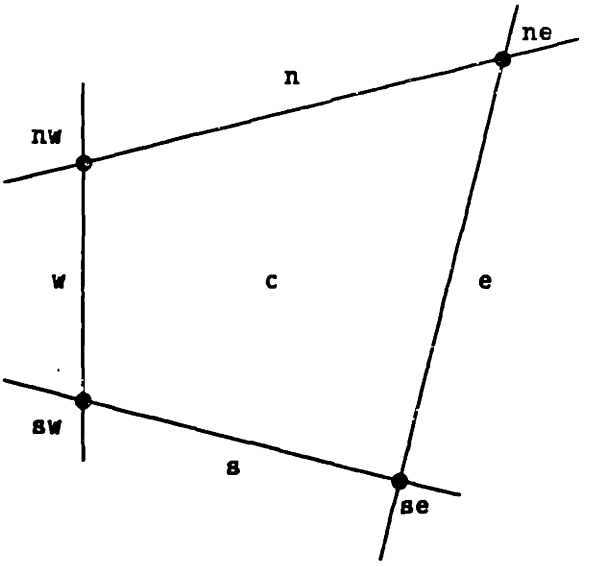


Figure 3.1: Elementary control volume

Consider a generalized computational cell (control volume) given in figure 3.1, where the cell sides and corners are named based upon compass directions and c denotes the cell center. A finite volume form of (3.1) can be written for this general cell (which is fixed in time) as

$$\frac{\partial \mathbf{U}_c}{\partial t} A_c + \mathcal{F}_s + \mathcal{F}_e + \mathcal{F}_n + \mathcal{F}_w = 0 \tag{3.3}$$

where for example

$$\mathcal{F}_s = \int_{sw}^{se} (\mathbf{F}\, dy - \mathbf{G}\, dx) \tag{3.4}$$

and A_c, the cell area, can be computed as half the cross product of the cell's diagonal vectors, or

$$A_c = \frac{1}{2} \{(x_{ne} - x_{sw})(y_{nw} - y_{se}) - (x_{nw} - x_{se})(y_{ne} - y_{sw})\} \tag{3.5}$$

In order to compute the integral in (3.4), one has to specify where the discrete values of the dependent variables are stored; two possible alternatives are node-based grids on which the dependent variables are stored at

the corner nodes of the computational cells, and cell-based grids on which those dependent variables are stored at the cell centers. A node-based grid was chosen for this work as denoted by the solid circles at the intersections of the grid lines in figure 3.1.

The flux integral required in (3.4) can be directly approximated by the trapezoidal integration

$$\mathcal{F}_s = \frac{\mathbf{F}_{sw}+\mathbf{F}_{se}}{2}(y_{se}-y_{sw}) - \frac{\mathbf{G}_{sw}+\mathbf{G}_{se}}{2}(x_{se}-x_{sw}) \tag{3.6}$$

Since the dependent variables are stored at the cell corners, the flux calculation requires knowledge of $\mathbf{F}$ and $\mathbf{G}$ only along the cell edge, resulting in an approximation which is second-order accurate, even for non-rectangular and highly stretched cells.

Substituting (3.6) and the corresponding fluxes for the other faces into (3.3) yields

$$\begin{aligned}\frac{\partial \mathbf{U}_c}{\partial t}A_c &+ \frac{\mathbf{F}_{sw}+\mathbf{F}_{se}}{2}(y_{se}-y_{sw}) - \frac{\mathbf{G}_{sw}+\mathbf{G}_{se}}{2}(x_{se}-x_{sw}) \\ &+ \frac{\mathbf{F}_{se}+\mathbf{F}_{ne}}{2}(y_{ne}-y_{se}) - \frac{\mathbf{G}_{se}+\mathbf{G}_{ne}}{2}(x_{ne}-x_{se}) \\ &+ \frac{\mathbf{F}_{ne}+\mathbf{F}_{nw}}{2}(y_{nw}-y_{ne}) - \frac{\mathbf{G}_{ne}+\mathbf{G}_{nw}}{2}(x_{nw}-x_{ne}) \\ &+ \frac{\mathbf{F}_{nw}+\mathbf{F}_{sw}}{2}(y_{sw}-y_{nw}) - \frac{\mathbf{G}_{nw}+\mathbf{G}_{sw}}{2}(x_{sw}-x_{nw}) = 0\end{aligned} \tag{3.7}$$

which can be rewritten in the simpler form

$$\begin{aligned}\frac{\partial \mathbf{U}_c}{\partial t}A_c &+ \frac{\mathbf{F}_{sw}-\mathbf{F}_{ne}}{2}(y_{se}-y_{nw}) - \frac{\mathbf{G}_{sw}-\mathbf{G}_{ne}}{2}(x_{se}-x_{nw}) \\ &+ \frac{\mathbf{F}_{se}-\mathbf{F}_{nw}}{2}(y_{ne}-y_{sw}) - \frac{\mathbf{G}_{se}-\mathbf{G}_{nw}}{2}(x_{ne}-x_{sw}) = 0\end{aligned} \tag{3.8}$$

Unfortunately the node-based grid arrangement suffers from the feature that (3.8) predicts changes at the cell center which then somehow must be assigned or distributed to the corner nodes. A straightforward choice here is to assign $(\Delta t/4)(\partial \mathbf{U}_c/\partial t)$ to each corner node.

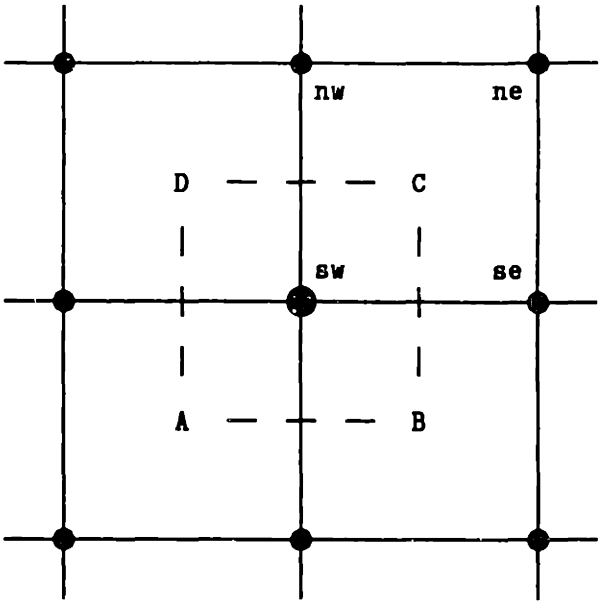


Figure 3.2: Control volumes adjoining the common node (sw)

To examine the change $\Delta t(\partial \mathbf{U}/\partial t)$ at any node, one must consider the contribution from all adjoining cells. Figure 3.2 shows the four cells (labelled A, B, C, and D) which are adjacent to the heavy node at the center of the figure. (Note that the labels sw, se, ne, and nw refer to the orientation of the respective nodes as viewed by cell C.)

The change at the node marked sw is the sum of the contributions from the four cells, or

$$\left\{\frac{\partial \mathbf{U}}{\partial t}\right\}_{\mathtt{sw}} = \frac{1}{4}\left\{\frac{\partial \mathbf{U}}{\partial t}\right\}_{\mathtt{A}} + \frac{1}{4}\left\{\frac{\partial \mathbf{U}}{\partial t}\right\}_{\mathtt{B}} + \frac{1}{4}\left\{\frac{\partial \mathbf{U}}{\partial t}\right\}_{\mathtt{C}} + \frac{1}{4}\left\{\frac{\partial \mathbf{U}}{\partial t}\right\}_{\mathtt{D}} \tag{3.9}$$

where the subscripts on the right side of (3.9) denote the cell in which $\partial \mathbf{U}/\partial t$ is calculated.

The transfer requirement for the change $\partial \mathbf{U}/\partial t$ (from the cells centers to the corners) can be eliminated if one uses a cell-based configuration instead. In this case trapezoidal integration cannot be used and the fluxes are

calculated by

$$\mathcal{F}_s = \mathbf{F}_s(y_{se} - y_{sw}) - \mathbf{G}_s(x_{se} - x_{sw}) \tag{3.10}$$

where the average flux vectors on each face ($\mathbf{F}_s$ and $\mathbf{G}_s$) have to be approximated, typically by averaging the values from the centers of the two cells adjoining the face. The flux evaluation is second order accurate for rectangular, uniformly spaced grid lines but degrades to first order accuracy for cells when the mesh has sufficient skew or stretching. Since uniform grids are seldom found in practice, this property was deemed unacceptable here and led to the choice of a node-based scheme.

The temporal derivatives must also be approximated in a discrete form. The most obvious procedure uses the first term in a Taylor series of $\mathbf{U}$ (with respect to t), yielding a simple forward time difference

$$\delta\mathbf{U} = \mathbf{U}^{n+1} - \mathbf{U}^n = \Delta t \left.\frac{\partial \mathbf{U}}{\partial t}\right|^n + \mathcal{O}(\Delta t)^2 \tag{3.11}$$

where $\partial\mathbf{U}/\partial t$ is calculated as in (3.9) and n is the time level. Unfortunately, this yields an unconditionally unstable scheme as are all forward-time, centered-space approximations of (3.2).

One way of circumventing this problem is to use a multi-stage Runge-Kutta-type integration instead of (3.11) or a backward time difference. Both of these centrally differenced schemes are purely dispersive (in space), requiring the explicit addition of a dissipative term such as fourth order smoothing, even in smooth regions of the flow field.

An alternative method of stabilizing the time integration is to introduce upwind biasing as is done implicitly in a Lax-Wendroff-type scheme[55,52,54,53]. Here (3.11) is replaced by the first two terms in the Taylor series expansion of $\mathbf{U}$, or

$$\delta\mathbf{U} \equiv \mathbf{U}^{n+1} - \mathbf{U}^n = \Delta t \left.\frac{\partial \mathbf{U}}{\partial t}\right|^n + \frac{(\Delta t)^2}{2} \left.\frac{\partial^2 \mathbf{U}}{\partial t^2}\right|^n + \mathcal{O}(\Delta t)^3 \tag{3.12}$$

The second time derivative is calculated by differentiating the governing equation (3.2)

$$\frac{\partial^2 \mathbf{U}}{\partial t^2} = \frac{\partial}{\partial t}\left(\frac{\partial \mathbf{U}}{\partial t}\right) = \frac{\partial}{\partial t}\left(-\frac{\partial \mathbf{F}}{\partial x} - \frac{\partial \mathbf{G}}{\partial y}\right) = -\frac{\partial}{\partial x}\left(\frac{\partial \mathbf{F}}{\partial t}\right) - \frac{\partial}{\partial y}\left(\frac{\partial \mathbf{G}}{\partial t}\right) \quad (3.13)$$

and since $\mathbf{F}$ and $\mathbf{G}$ are functions only of $\mathbf{U}$, the chain rule can be used, giving

$$\frac{\partial^2 \mathbf{U}}{\partial t^2} = -\frac{\partial}{\partial x}\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right) - \frac{\partial}{\partial y}\left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right) \quad (3.14)$$

where $\partial \mathbf{F}/\partial \mathbf{U}$ and $\partial \mathbf{G}/\partial \mathbf{U}$ are the Jacobians.

To include this second difference term, consider a secondary control volume as shown by the dashed lines in figure 3.2. Here the corners of the secondary cell are assumed to be at the centroids of the cells.

Begin by applying the divergence theorem to (3.14), obtaining

$$\iint \frac{\partial^2 \mathbf{U}}{\partial t^2}\, dx\, dy + \oint \left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\, dy - \frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\, dx\right) = 0 \quad (3.15)$$

This can be spatially integrated using a finite volume approximation as done above for (3.1), yielding (after simplification)

$$\begin{aligned}
\frac{\partial^2 \mathbf{U}_{sv}}{\partial t^2} A_{sv} &+ \frac{\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_A - \left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_C}{2}(y_B - y_D) \\
&- \frac{\left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_A - \left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_C}{2}(x_B - x_D) \\
&+ \frac{\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_B - \left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_D}{2}(y_C - y_A) \\
&- \frac{\left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_B - \left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}\right)_D}{2}(x_C - x_A) = 0
\end{aligned} \quad (3.16)$$

In order to use (3.16) on a cell-by-cell basis, the areas and the projections of the diagonals of the secondary volume are assumed to be approximately equal to their corresponding quantities on the current cell. For example, for cell C assume that

$$A_{sv} \approx A_C \quad (3.17a)$$

$$y_B - y_D \approx y_{se} - y_{nw} \tag{3.17b}$$

$$x_B - x_D \approx x_{se} - x_{nw} \tag{3.17c}$$

These first order accurate approximations are always multiplied by a term of $O(\Delta x)$, and thus the overall accuracy of the scheme is maintained.

Substituting (3.17) into (3.16) and rearranging, yields

$$\begin{aligned}\left\{\frac{\partial^2 \mathbf{U}}{\partial t^2}\right\}_{sw} = &\left\{-\frac{1}{2A}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(y_{se}-y_{nw}) + \frac{1}{2A}\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(x_{se}-x_{nw})\right\}_A \\ &+\left\{-\frac{1}{2A}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(y_{ne}-y_{sw}) + \frac{1}{2A}\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(x_{ne}-x_{sw})\right\}_B \\ &+\left\{+\frac{1}{2A}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(y_{se}-y_{nw}) - \frac{1}{2A}\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(x_{se}-x_{nw})\right\}_C \\ &+\left\{+\frac{1}{2A}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(y_{ne}-y_{sw}) - \frac{1}{2A}\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(x_{ne}-x_{sw})\right\}_D\end{aligned} \tag{3.18}$$

Substituting (3.9) and (3.18) into (3.12) then yields the contribution of cell **C** to node **sw** as

$$\begin{aligned}&(\delta \mathbf{U}_{sw})_C = \\ &\left\{\frac{\Delta t}{4}\left[\frac{\partial \mathbf{U}}{\partial t}\right] + \frac{(\Delta t)^2}{4A}\left[\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(y_{se}-y_{nw}) - \frac{\partial \mathbf{G}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial t}(x_{se}-x_{nw})\right]\right\}_C\end{aligned} \tag{3.19}$$

Ni defines the following terms:

$$\Delta \mathbf{U}_c = \left(\frac{\partial \mathbf{U}}{\partial t}\right)_c \Delta t_c \tag{3.20}$$

$$\begin{aligned}\Delta f_c = \frac{\Delta t_c}{A_c}\Bigg[&+\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right)_c \Delta \mathbf{U}_c \left(\frac{y_{nw}+y_{ne}-y_{sw}-y_{se}}{2}\right) \\ &-\left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\right)_c \Delta \mathbf{U}_c \left(\frac{x_{nw}+x_{ne}-x_{sw}-x_{se}}{2}\right)\Bigg]\end{aligned} \tag{3.21}$$

and

$$\begin{aligned}\Delta g_c = \frac{\Delta t_c}{A_c}\Bigg[&-\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right)_c \Delta \mathbf{U}_c \left(\frac{y_{se}+y_{ne}-y_{nw}-y_{sw}}{2}\right) \\ &+\left(\frac{\partial \mathbf{G}}{\partial \mathbf{U}}\right)_c \Delta \mathbf{U}_c \left(\frac{x_{se}+x_{ne}-x_{nw}-x_{sw}}{2}\right)\Bigg]\end{aligned} \tag{3.22}$$

After much algebra, Ni obtains

$$(\delta \mathbf{U}_{sw})_c = \frac{1}{4}[\Delta \mathbf{U}_c - \Delta f_c - \Delta g_c] \tag{3.23}$$

In summary, the node-based finite volume Lax-Wendroff-type integration scheme developed by Ni which is used here consists of the following steps:

FLUX Compute the "change" of the dependent variables in the center of each cell using

$$\Delta \mathbf{U}_c = \frac{\Delta t_c}{A_c}\left\{ -\frac{\mathbf{F}_{sw} - \mathbf{F}_{ne}}{2}(y_{se} - y_{nw}) + \frac{\mathbf{G}_{sw} - \mathbf{G}_{ne}}{2}(x_{se} - x_{nw}) - \frac{\mathbf{F}_{se} - \mathbf{F}_{nw}}{2}(y_{ne} - y_{sw}) + \frac{\mathbf{G}_{se} - \mathbf{G}_{nw}}{2}(x_{ne} - x_{sw})\right\} \tag{3.24}$$

which is simply a discrete representation of the conservation laws in integral form;

DIST Compute the "correction" at each node by the applying the "distribution" formulae

$$(\delta \mathbf{U}_{sw})_c = \frac{1}{4}[\Delta \mathbf{U}_c - \Delta f_c - \Delta g_c] \tag{3.25a}$$

$$(\delta \mathbf{U}_{se})_c = \frac{1}{4}[\Delta \mathbf{U}_c + \Delta f_c - \Delta g_c] \tag{3.25b}$$

$$(\delta \mathbf{U}_{ne})_c = \frac{1}{4}[\Delta \mathbf{U}_c + \Delta f_c + \Delta g_c] \tag{3.25c}$$

$$(\delta \mathbf{U}_{nw})_c = \frac{1}{4}[\Delta \mathbf{U}_c - \Delta f_c + \Delta g_c] \tag{3.25d}$$

where Δf_c and Δg_c are given by (3.21) and (3.22), respectively.

UPDT Update the dependent variables at each node i by

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \delta \mathbf{U}_i \tag{3.26}$$

where $\delta \mathbf{U}_i$ is the sum of the corrections from the adjacent cells, as distributed above.

This scheme is very similar to Hall's scheme[41]. In both cases, since the scheme is explicit, the CFL condition imposes a restriction on the size of Δt. That limit is determined in section 3.3.2, along with an analysis of the other properties of this scheme.

3.2 Boundary Conditions

Boundary conditions must also be discretized in order to form a closed set of equations. First a general boundary condition formulation will be given, and then specific implementations of the general form for the Euler equations applied to airfoil-type bodies. The general boundary condition formulation discussed below is intended primarily for open boundaries, that is boundaries introduced to solve the problem but which are not true boundaries of the physical problem.

3.2.1 General formulation

Hyperbolic governing equations such as (3.2) have by definition complete sets of real eigenvalues and linearly independent eigenvectors. The eigenvalues describe the paths (characteristic directions) along which the characteristic variables (given by the eigenvectors) propagate, remaining constant as long as the governing equations contain no source terms. The sign of each eigenvalue determines whether that characteristic propagates from the left or from the right and hence for boundary nodes whether the corresponding physics originates from the interior or exterior of the domain. This difference underlies the general boundary condition formulation: values for "entering" characteristics can be either directly or indirectly imposed; for "exiting" characteristics, the values cannot be imposed but must be consistent with the discrete solution in the interior of the domain. Similar approaches have been previously discussed by Chakravarthy[22] and Boerstoel[13].

To demonstrate this general boundary condition formulation, consider

the one-dimensional wave equation

$$\frac{\partial^2 v}{\partial t^2} - a^2 \frac{\partial^2 v}{\partial \xi^2} = 0 \tag{3.27}$$

which can be written in the form (3.2) if

$$\mathbf{U} = \begin{pmatrix} v \\ w \end{pmatrix} \quad \mathbf{F} = \begin{pmatrix} aw \\ av \end{pmatrix} \tag{3.28}$$

The Jacobian matrix for this equation is

$$\frac{\partial \mathbf{F}}{\partial \mathbf{U}} = \begin{pmatrix} 0 & a \\ a & 0 \end{pmatrix} \tag{3.29}$$

which gives the eigenvalue and left-eigenvector matrices

$$\mathbf{\Lambda} = \begin{pmatrix} a & 0 \\ 0 & -a \end{pmatrix} \quad \mathbf{L} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.30}$$

The relationships between the corrections in the characteristic and dependent variables are then given by

$$\begin{pmatrix} \delta c_1 \\ \delta c_2 \end{pmatrix} = \delta \mathbf{C} = \mathbf{L}\, \delta \mathbf{U} = \begin{pmatrix} \delta v + \delta w \\ \delta v - \delta w \end{pmatrix} \tag{3.31}$$

and conversely

$$\begin{pmatrix} \delta v \\ \delta w \end{pmatrix} = \delta \mathbf{U} = \mathbf{L}^{-1}\, \delta \mathbf{C} = \begin{pmatrix} \frac{\delta c_1 + \delta c_2}{2} \\ \frac{\delta c_1 - \delta c_2}{2} \end{pmatrix} \tag{3.32}$$

The propagation of the characteristic variables is shown in figure 3.3, where the abscissa represents the spatial coordinate ξ and the ordinate represents t, the time. All values are assumed known at time t_0 and it remains to determine the values Δt later, say at D. From D, one can trace back the path of each characteristic back to time t_0. For example, using the value of

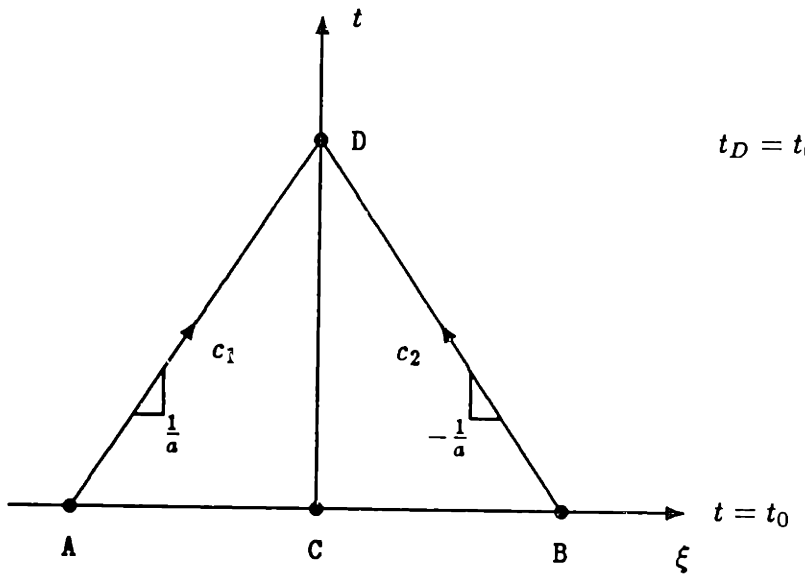


Figure 3.3: Characteristic paths for one-dimensional wave equation

the first eigenvalue, the slope of the path for the first characteristic variable can be determined as

$$\left(\frac{dt}{d\xi}\right)_1 = \frac{1}{\lambda_1} = \frac{1}{a} \tag{3.33}$$

resulting in the intersection at A. Similarly, the second characteristic, c_2, propagates to the left from B to D along the path whose slope is

$$\left(\frac{dt}{d\xi}\right)_2 = \frac{1}{\lambda_2} = -\frac{1}{a} \tag{3.34}$$

Consider now the situation where point C (and hence D) lies on a boundary such that A is outside and B is within the domain. Here, because the first characteristic variable c_1 propagates to the boundary from outside the domain, the correction δc_1 has to be specified by a boundary condition. This can be accomplished in a variety of ways, three of which are discussed here.

In the simplest formulation, called a *radiation condition*, it is assumed that there are no incoming waves. Thus by definition the correction in the incoming characteristic is given by

$$\delta c_{\text{in}} = 0 \tag{3.35}$$

which for the model equation currently being discussed becomes

$$\delta c_1 = L_1 \, \delta \mathbf{U} = \delta v + \delta w = 0 \tag{3.36}$$

where L_1 is the first row of the eigenvector matrix $\mathbf{L}$. This boundary condition is generally acceptable only when the effects of incoming waves are negligible.

In the second boundary condition formulation, the value for the incoming characteristic variable at the boundary is prescribed *a priori*, either as a constant or as a function of time. Hence at every iteration the correction in the incoming characteristic is given by

$$\delta c_{\text{in}} = (c_{\text{in}})_{\text{pres}} - (c_{\text{in}})_{\text{old}} \tag{3.37}$$

where $(c_{\text{in}})_{\text{pres}}$ is the prescribed value and $(c_{\text{in}})_{\text{old}}$ is the current (old) value. For the one-dimensional wave equation example, this takes the form

$$\delta c_1 = L_1 \, \delta \mathbf{U} = \delta v + \delta w = (c_1)_{\text{pres}} - (c_1)_{\text{old}} \tag{3.38}$$

The only condition on the use of this boundary condition is that the value of the incoming characteristic variable be known *a priori*. Unfortunately physical boundary conditions are generally prescribed in terms of physical quantities and thus this boundary condition is used infrequently.

In the third boundary condition formulation, called a *prescription condition*, for each incoming characteristic there is some parameter $\psi(\mathbf{U})$ whose value is prescribed at the boundary, either as a constant or as some function of time. Hence the correction in the characteristic variable is determined as that which is sufficient to cause ψ to achieve its prescribed value after the boundary condition application. This can be accomplished by ensuring that

$$\delta \psi = \psi_{\text{pres}} - \psi_{\text{old}} \tag{3.39}$$

where ψ_{pres} is the prescribed value after the current iteration and ψ_{old} is the (old) value at the beginning of the iteration. Since ψ is a function of $\mathbf{U}$ and

hence C, it follows that

$$\delta\psi = \frac{\partial\psi}{\partial c_1}\delta c_1 + \frac{\partial\psi}{\partial c_2}\delta c_2 + \cdots \tag{3.40}$$

For the one-dimensional wave equation, (3.40) can be rearranged to give the correction in the incoming characteristic, δc_1, in terms of the outgoing characteristic δc_2, or

$$\delta c_1 = \frac{\delta\psi - \frac{\partial\psi}{\partial c_2}\delta c_2}{\frac{\partial\psi}{\partial c_1}} = \frac{(\psi_{\text{pres}} - \psi_{\text{old}}) - \frac{\partial\psi}{\partial c_2}\delta c_2}{\frac{\partial\psi}{\partial c_1}} \tag{3.41}$$

The denominator of (3.41) will only vanish if ψ is a linear combination of the outgoing characteristics, in this case c_2; this then puts a restriction on the choice of ψ.

A slightly simpler yet entirely equivalent form of the prescription condition can be derived if one writes

$$\delta\psi = \frac{\partial\psi}{\partial u_1}\delta u_1 + \frac{\partial\psi}{\partial u_2}\delta u_2 + \cdots \tag{3.42}$$

A direct combination of (3.39) and (3.42) for the one-dimensional wave equation yields

$$\frac{\partial\psi}{\partial v}\delta v + \frac{\partial\psi}{\partial w}\delta w = \psi_{\text{pres}} - \psi_{\text{old}} \tag{3.43}$$

Notice that the characteristic variables do not enter (3.43) directly but could be found if desired.

In addition to determining the correction at the boundary due to the incoming characteristics, the corrections in the outgoing characteristics must also be determined. Here it is assumed that the corrections in the dependent variables (and hence the characteristic variables) which are predicted by the interior scheme are correct. In other words, the corrections in the outgoing characteristic variables at the boundary can be computed from

$$\delta c_{\text{out}} = L_{\text{out}}\,\delta \mathbf{U}_{\text{pred}} \tag{3.44}$$

where L_{out} is the appropriate row from the left-eigenvector matrix $\mathbf{L}$ and $\delta\mathbf{U}_{\text{pred}}$ is the correction in the dependent variables predicted by the interior scheme, that is the correction which results from the distribution formulae (3.25). Although this assumption cannot rigorously be shown to be true, it is based upon the observation that if it were not true, at least qualitatively, then the integration scheme would not be able to properly predict the flow-field in the interior of the domain.

For the one-dimensional wave equation, this takes the form

$$\delta c_2 = L_2\,\delta\mathbf{U}_{\text{pred}} = \delta v_{\text{pred}} - \delta w_{\text{pred}} \tag{3.45}$$

In conclusion, the boundary condition formulation then is simply a combination of the incoming and outgoing characteristics at the boundary. For the radiation condition, this results in m equations of the form

$$L_i\,\delta\mathbf{U}_{\text{bc}} = 0 \qquad i = 1,\ldots,m \tag{3.46}$$

where m is the number of incoming characteristics and the subscript $()_{\text{bc}}$ denotes the output of the boundary condition processing. For the $n-m$ outgoing characteristics, (where n is the total number of dependent variables), there are equations of the form

$$L_i\,\delta\mathbf{U}_{\text{bc}} = L_i\,\delta\mathbf{U}_{\text{pred}} \qquad i = m+1,\ldots,n \tag{3.47}$$

These sets of equations can be written together as one matrix equation, yielding the radiation boundary condition equation

$$\mathbf{L}\,\delta\mathbf{U}_{\text{bc}} = \begin{pmatrix} 0 \\ \vdots \\ L_{m+1}\,\delta\mathbf{U}_{\text{pred}} \\ \vdots \end{pmatrix} \tag{3.48}$$

For the one-dimensional wave equation, this then takes the form

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} \delta v \\ \delta w \end{pmatrix}_{\text{bc}} = \begin{pmatrix} 0 \\ \delta v_{\text{pred}} - \delta w_{\text{pred}} \end{pmatrix} \tag{3.49}$$

Similarly for the prescription condition, one obtains m equations of the form

$$\begin{pmatrix} \frac{\partial \psi_i}{\partial u_1} & \cdots & \frac{\partial \psi_i}{\partial u_n} \end{pmatrix} \delta \mathbf{U}_{\text{bc}} = (\psi_i)_{\text{pres}} - (\psi_i)_{\text{old}} \qquad i = 1, \ldots, m \tag{3.50}$$

where there are m different parameters ψ (and hence the subscript i), one corresponding to each of the m incoming characteristics. Additionally there are $n - m$ equations for the outgoing characteristics of the form

$$L_i \, \delta \mathbf{U}_{\text{bc}} = L_i \, \delta \mathbf{U}_{\text{pred}} \qquad i = m+1, \ldots, n \tag{3.51}$$

Again these sets of equations can be written together as one matrix equation, yielding the prescription boundary condition equation

$$\begin{pmatrix} \frac{\partial \psi_1}{\partial u_1} \cdots \frac{\partial \psi_1}{\partial u_n} \\ \vdots \\ L_{m+1} \\ \vdots \end{pmatrix} \delta \mathbf{U}_{\text{bc}} = \begin{pmatrix} (\psi_1)_{\text{pres}} - (\psi_1)_{\text{old}} \\ \vdots \\ L_{m+1} \, \delta \mathbf{U}_{\text{pred}} \\ \vdots \end{pmatrix} \tag{3.52}$$

For the one-dimensional wave equation, this then takes the form

$$\begin{pmatrix} \frac{\partial \psi}{\partial v} & \frac{\partial \psi}{\partial w} \\ 1 & -1 \end{pmatrix} \begin{pmatrix} \delta v \\ \delta w \end{pmatrix}_{\text{bc}} = \begin{pmatrix} \psi_{\text{pres}} - \psi_{\text{old}} \\ \delta v_{\text{pred}} - \delta w_{\text{pred}} \end{pmatrix} \tag{3.53}$$

It should be noted that (3.49) and (3.53) are only valid for the one-dimensional wave equation (3.27) at the left end of the domain. A similar analysis as above, but with c_1 being the outgoing and c_2 being the incoming characteristics, is needed for the right end of the domain.

The boundary conditions are implemented in the scheme given in the previous section by adding the following step between **DIST** and **UPDT**:

BCON for boundary nodes, recompute the corrections in the dependent variables $\delta \mathbf{U}_{bc}$ using boundary condition equations such as the radiation condition (3.48) or the prescription condition (3.52).

For multi-dimensional problems, a straightforward extension of the above analysis yields characteristic variables which propagate along the surface of cones rather than along lines. A discrete representation of these cones is significantly more complicated than the above[59]. To circumvent these difficulties, a local quasi-one-dimensional approximation in the vicinity of boundaries is often made[84,79], wherein some dominant direction is chosen (somewhat arbitrarily) such that variations in the chosen direction are assumed to be much larger than those normal to it. This then allows the multi-dimensional governing equation to be rewritten as a quasi-one-dimensional equation and the above characteristic analysis can be applied directly.

Another difficulty with the characteristic boundary analysis occurs when the flux vectors $\mathbf{F}$ and $\mathbf{G}$ are non-linear functions of the dependent variables $\mathbf{U}$. In this case the governing equation has to be linearized before the characteristic variables can be computed, resulting in relationships between the physical and characteristic variables which are weak functions of the condition used in the linearization. Normally this dependence on the linearization conditions is not serious if either the radiation or prescription conditions are used as described above.

Additionally, for linear problems the relationships between the characteristic and dependent variables are exact. As a consequence summing two characteristic variables directly yields exactly the same result as converting the characteristic variables to their corresponding dependent variables, summing them, and then converting the result back to the characteristic form. Unfortunately non-linear equations do not in general possess this quality. As a result, in order to ensure that the value of variables which have to maintain some prescribed value do not drift, terms of the form $\mathcal{A}_{\text{pres}} - \mathcal{A}_{\text{old}}$

are required on the right hand side of (3.48) and (3.52), even when the prescribed values are constant.

In summary, the characteristic boundary condition formulation described in this section is applicable to computations on any set of hyperbolic governing equations in the form (3.1) and (3.2). For one-dimensional linear equations the radiation and prescription conditions given by (3.48) and (3.52) can be applied directly. For multi-dimensional equations these conditions can usually be applied after approximating the governing equations as a set of quasi-one-dimensional equations. Also for non-linear equations this boundary condition formulation can be used if the equations are first linearized and if terms are included to ensure that the solution does not drift due to the non-linearities in the conversions between characteristic and dependent variables.

3.2.2 Euler equation boundary conditions

In this section, the three different boundary condition types which are required for Euler flow calculations over airfoil-like bodies are described: the far-field, solid wall, and the trailing edge conditions. The first of these conditions involves a straightforward application of the general formulation presented in the previous section. In the latter two conditions this general formulation cannot be applied directly, resulting in the derivation of slightly different conditions.

For reference, the dependent variable and flux vectors for the two-dimensional unsteady Euler equations are given by

$$\mathbf{U} = \begin{pmatrix} \rho \\ \tilde{u} \\ \tilde{v} \\ E \end{pmatrix} \quad \mathbf{F} = \begin{pmatrix} \tilde{u} \\ \frac{\tilde{u}^2}{\rho} + p \\ \frac{\tilde{u}\tilde{v}}{\rho} \\ (E+p)\frac{\tilde{u}}{\rho} \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} \tilde{v} \\ \frac{\tilde{u}\tilde{v}}{\rho} \\ \frac{\tilde{v}^2}{\rho} + p \\ (E+p)\frac{\tilde{v}}{\rho} \end{pmatrix} \tag{3.54}$$

where ρ, $\tilde{u}$, $\tilde{v}$, and E are the mass, axial momentum, tangential momentum, and energy per unit volume and where the pressure is given by

3.2.2.1 Far-field condition

$$p = (\gamma - 1)\left[E - \frac{1}{2}\frac{\tilde{u}^2 + \tilde{v}^2}{\rho}\right] \tag{3.55}$$

In order to apply the characteristic boundary conditions, the two-dimensional Euler equations must first be converted to quasi-one-dimensional form. This involves rotating the governing equations to a new coordinate system, the orientation of which is somewhat arbitrary. Most researchers[84,79] choose to rotate to a direction which is normal to the far-field boundary. They argue that the waves normal to the boundary can be captured and properly passed out of the computational domain. Unfortunately the boundary condition approximation is strictly then a function of the arbitrarily-drawn boundary shape.

Alternatively, it seems more reasonable to choose the local flow direction (around which the characteristic cone is centered) as that around which the local one-dimensional approximation is to be made. In this way the boundary condition application at an arbitrarily chosen boundary is dependent only on the local flow direction and is independent of the local boundary shape. It is the latter approach which is taken here.

In section 2.4, the Euler equations were cast in intrinsic coordinates, yielding the quasi-one-dimensional equations

$$\frac{\partial \hat{\mathbf{U}}}{\partial t} + \hat{\mathbf{A}}\frac{\partial \hat{\mathbf{U}}}{\partial s} + \hat{\mathbf{B}} = 0 \tag{3.56}$$

where

$$\hat{\mathbf{U}} = \begin{pmatrix} \rho \\ \tilde{q} \\ \theta \\ E \end{pmatrix} = \begin{pmatrix} \rho \\ \sqrt{\tilde{u}^2 + \tilde{v}^2} \\ \arctan \frac{\tilde{v}}{\tilde{u}} \\ E \end{pmatrix} \tag{3.57}$$

Here ρ, $\tilde{q}$, θ, and E represent the density, streamwise momentum, streamline inclination, and total energy respectively, with their relationships with the corresponding Cartesian variables also indicated in (3.57). The matrix $\hat{\mathbf{A}}$ is only a function of $\hat{\mathbf{U}}$ whereas $\hat{\mathbf{B}}$ is both a function of $\hat{\mathbf{U}}$ and derivatives in the cross-flow direction.

In addition to rotating into quasi-one-dimensional form, the Euler equations have to be linearized before the eigenvalues and eigenvectors can be found. Though the choice of the condition around which to linearize is not critical, best results are obtained when the linearization and local flow conditions are close to each other. This being the criterion, there are two logical choices for the basis of the linearization at each boundary node: the local flow condition (say from the previous iteration) or the free-stream condition. In general the former will be slightly more accurate, although the accuracy difference will be small for far-field conditions in airfoil-type problems where the far-field boundary is at least five chord-lengths from the body (see section 2.6.3).

The other difference between the two linearization choices is the efficiency with which the linearizations can be implemented. If the basis of the linearization is constant (for example the free-stream condition), then the linearization matrix and its inverse need be computed only once and stored for use with each iteration. It is largely based upon this latter observation that the free-stream conditions are used as the basis of the linearization.

This results in $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ being constant matrices, with all entries being evaluated with free-stream values. Following section 2.5.1, the eigenvalue

and left-eigenvector matrices of $\hat{\mathbf{A}}$ are

$$\mathbf{\Lambda} = \begin{pmatrix} \frac{\tilde{q}}{\rho} - a & 0 & 0 & 0 \\ 0 & \frac{\tilde{q}}{\rho} & 0 & 0 \\ 0 & 0 & \frac{\tilde{q}}{\rho} & 0 \\ 0 & 0 & 0 & \frac{\tilde{q}}{\rho} + a \end{pmatrix} \tag{3.58}$$

and

$$\mathbf{L} = \begin{pmatrix} \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} + \frac{a\tilde{q}}{\rho} & -(\gamma-1)\frac{\tilde{q}}{\rho} - a & 0 & \gamma - 1 \\ \frac{\gamma+1}{2}\frac{\tilde{q}^2}{\rho^2} - \gamma\frac{E}{\rho} & -\frac{\tilde{q}}{\rho} & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} - \frac{a\tilde{q}}{\rho} & -(\gamma-1)\frac{\tilde{q}}{\rho} + a & 0 & \gamma - 1 \end{pmatrix} \tag{3.59}$$

where all quantities are evaluated at the free-stream conditions and the speed of sound, a, is given by

$$a = \sqrt{\gamma(\gamma-1)\left(\frac{E}{\rho} - \frac{\tilde{q}^2}{2\rho^2}\right)} \tag{3.60}$$

The corrections in the characteristic and dependent variables are then related by

$$\delta\mathbf{C} = \mathbf{L}\,\delta\hat{\mathbf{U}} \qquad \delta\hat{\mathbf{U}} = \mathbf{L}^{-1}\,\delta\mathbf{C} \tag{3.61}$$

Notice that the number of positive and negative eigenvalues depends on whether the flow is subsonic or supersonic and whether the flow is entering or exiting the domain at any particular boundary point. Thus four different far-field boundary treatments are required, each of which are discussed in turn in the following paragraphs.

Subsonic inflow Examination of (3.58) indicates that for subsonic inflow ($0 < \tilde{q}/\rho < a$) there is one negative and three positive eigenvalues, and hence three physical parameters can be prescribed at the boundary. According to (3.41), these physical parameters must not be a linear combination of the outgoing characteristics; they also must be linearly independent to ensure that the resulting matrix equation for $\delta\hat{\mathbf{U}}$ is not singular.

Unfortunately these are necessary but not sufficient restrictions on the choice of physical parameters. An eigenmode analysis of perturbations about the steady state solution of the Euler equations in a one-dimensional channel by Giles[39] shows the results of inappropriate choices of prescribed parameters at inflow and outflow boundaries. He shows that for certain boundary condition combinations, unsteady waves can bounce between the inlet and exit, growing in time. Parameters which he found yielded stable conditions at subsonic inflows were entropy and stagnation enthalpy or equivalently stagnation pressure and temperature; the former two are used here.

Written in terms of the dependent variables, the stagnation enthalpy and entropy are given by

$$(h_o)_{\text{pres}} = \left[-\frac{\gamma - 1}{2} \frac{\tilde{q}^2}{\rho^2} + \gamma \frac{E}{\rho} \right]_{\text{fs}} \tag{3.62}$$

and

$$(s)_{\text{pres}} = (\gamma - 1) \left[\frac{E}{\rho^\gamma} - \frac{1}{2} \frac{\tilde{q}^2}{\rho^{\gamma+1}} \right]_{\text{fs}} \tag{3.63}$$

respectively, where the subscript $()_{\text{fs}}$ denotes free-stream conditions. By Assumptions 2.7 and 2.8, these equations are equivalent to the corresponding equations evaluated with far-field values.

For the eigenvalue in two dimensions which does not have a direct correspondence in one dimension (that is λ_3), another condition needs to be prescribed. The structure of $\mathbf{L}$ indicates that the third eigenvalue and eigenvector are completely uncoupled from the others; this pair is associated with the change in inclination (curvature) as a function of the cross-stream pressure gradient. Since λ_3 enters the domain at subsonic inlet points, one can prescribe the information that it carries, that is the local flow inclination, θ. An appropriate value for θ_{pres} is computed by evaluating the far-field flow approximation developed in section 2.6.3 based upon the surface force coefficients measured on the body after the previous time step.

To implement the subsonic inflow boundary condition, the first step is to separate the quasi-one-dimensional information from the flow inclination information by deleting the third row and column from $\mathbf{L}$, leaving

$$\begin{pmatrix} \delta c_1 \\ \delta c_2 \\ \delta c_4 \end{pmatrix} = \begin{pmatrix} \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} + \frac{a\tilde{q}}{\rho} & -(\gamma-1)\frac{\tilde{q}}{\rho} - a & (\gamma-1) \\ \frac{\gamma+1}{2}\frac{\tilde{q}^2}{\rho^2} - \gamma\frac{E}{\rho} & -\frac{\tilde{q}}{\rho} & 1 \\ \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} - \frac{a\tilde{q}}{\rho} & -(\gamma-1)\frac{\tilde{q}}{\rho} + a & (\gamma-1) \end{pmatrix}_{\text{fs}} \begin{pmatrix} \delta\rho \\ \delta\tilde{q}_s \\ \delta E \end{pmatrix} \tag{3.64}$$

where the subscript $()_{\text{fs}}$ indicates that the whole matrix is evaluated at the free-stream conditions. The corrections in the streamwise and cross-flow momentum, ($\delta\tilde{q}$ and $\delta\tilde{q}_n$), are related to the corrections in the Cartesian momentum components by

$$\delta\tilde{q} = +\delta\tilde{u}\cos\theta_{\text{old}} + \delta\tilde{v}\sin\theta_{\text{old}} \tag{3.65a}$$

$$\delta\tilde{q}_n = -\delta\tilde{u}\sin\theta_{\text{old}} + \delta\tilde{v}\cos\theta_{\text{old}} \tag{3.65b}$$

where θ_{old} is the flow inclination at the node before applying any corrections.

Now the prescription boundary condition (3.52) can be created using the chain-rule form of (3.62) and (3.63) as well as the first equation of (3.64), yielding

$$\begin{pmatrix} \delta h_o \\ \frac{\rho^\gamma}{\gamma-1}\delta s \\ \delta c_1 \end{pmatrix} = \mathbf{M}_{\text{sub in}} \begin{pmatrix} \delta\rho \\ \delta\tilde{q} \\ \delta E \end{pmatrix}_{\text{bc}} \tag{3.66a}$$

where

$$\mathbf{M}_{\text{sub in}} = \begin{pmatrix} (\gamma-1)\frac{\tilde{q}^2}{\rho^3} - \gamma\frac{E}{\rho^2} & -(\gamma-1)\frac{\tilde{q}}{\rho^2} & \frac{\gamma}{\rho} \\ -\gamma\frac{E}{\rho} + \frac{\gamma+1}{2}\frac{\tilde{q}^2}{\rho^2} & -\frac{\tilde{q}}{\rho} & 1 \\ \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} + \frac{a\tilde{q}}{\rho} & -(\gamma-1)\frac{\tilde{q}}{\rho} - a & (\gamma-1) \end{pmatrix}_{\text{fs}} \tag{3.66b}$$

which can be inverted to yield

$$\begin{pmatrix} \delta\rho \\ \delta\tilde{q} \\ \delta E \end{pmatrix}_{\text{bc}} = \frac{\tilde{\mathbf{M}}^{-1}_{\text{sub in}}}{\tilde{q}_{\text{fs}} + \rho_{\text{fs}}a_{\text{fs}}} \begin{pmatrix} (h_o)_{\text{pres}} - (h_o)_{\text{old}} \\ \frac{\rho^\gamma_{\text{fs}}}{\gamma-1}\left[(s)_{\text{pres}} - (s)_{\text{old}}\right] \\ (\delta c_1)_{\text{pred}} \end{pmatrix} \tag{3.67a}$$

where

$$M^{-1}_{\text{sub in}} = \tag{2.67b}$$
$$\begin{pmatrix} \frac{\rho^2}{a} & -(\gamma-1)\frac{\tilde{q}}{a^2} - \gamma\frac{\rho}{a} & \frac{\tilde{q}}{a^2} \\ \rho^2 + \frac{\rho\tilde{q}}{a} & -\rho - (\gamma-1)\frac{\tilde{q}^2}{\rho a^2} - \gamma\frac{\tilde{q}}{a} & -\rho + \frac{\tilde{q}^2}{\rho a^2} \\ \rho\tilde{q} + \gamma\frac{E\rho}{a} - \frac{\gamma-1}{2}\frac{\tilde{q}^2}{a} & -q - \frac{\gamma-1}{2}\frac{\tilde{q}^3}{\rho^2 a^2} - \gamma\frac{E}{a} & -q + \gamma\frac{E\tilde{q}}{\rho a^2} - \frac{\gamma-1}{2}\frac{\tilde{q}^3}{a^2\rho^2} \end{pmatrix}_{\text{fs}}$$

where the subscript $()_{\text{bc}}$ identifies the corrections in the dependent variables which are the output of the boundary condition processing. The terms $(h_o)_{\text{old}}$ and $(s)_{\text{old}}$ are computed by

$$(h_o)_{\text{old}} = \left[-\frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} + \gamma\frac{E}{\rho}\right]_{\text{old}} \tag{3.68}$$

and

$$(s)_{\text{old}} = (\gamma-1)\left[\frac{E}{\rho^\gamma} - \frac{1}{2}\frac{\tilde{q}^2}{\rho^{\gamma+1}}\right]_{\text{old}} \tag{3.69}$$

which are similar to (3.62) and (3.63) except that they are calculated using local flow conditions before any corrections are applied. The predicted correction in the first characteristic is computed using the first row of (3.64), or

$$\begin{aligned}(\delta c_1)_{\text{pred}} = &\left[\frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} + \frac{a\tilde{q}}{\rho}\right]_{\text{fs}}(\delta\rho)_{\text{pred}} \\ &+ \left[-(\gamma-1)\frac{\tilde{q}}{\rho} - a\right]_{\text{fs}}(\delta\tilde{q})_{\text{pred}} + [\gamma-1]_{\text{fs}}(\delta E)_{\text{pred}}\end{aligned} \tag{3.70}$$

Once the corrections given by the quasi-one-dimensional characteristics are computed, the corrections in the cross-flow momentum, $\delta\tilde{q}_n$, can be computed using

$$\delta\tilde{q}_n = [(\tilde{q})_{\text{old}} + (\delta\tilde{q})_{\text{bc}}]\tan(\theta_{\text{pres}} - \theta_{\text{old}}) \tag{3.71}$$

where $(\theta_{\text{pres}} - \theta_{\text{old}})$ is the angle through which the local flow must be rotated.

Finally the corrections in the Cartesian momentum components are computed by

$$\delta\tilde{u} = \cos\theta_{\text{old}}\,\delta\tilde{q} - \sin\theta_{\text{old}}\,\delta\tilde{q}_n \tag{3.72a}$$

$$\delta\tilde{v} = \sin\theta_{\text{old}}\,\delta\tilde{q} + \cos\theta_{\text{old}}\,\delta\tilde{q}_n \tag{3.72b}$$

Subsonic outflow Examination of (3.58) indicates that for subsonic outflow ($0 > \tilde{q}/\rho > -a$) there are three negative and one positive eigenvalue, and hence only one physical parameter can be prescribed at the boundary; Giles[39] shows that an appropriate choice for this condition is the static pressure

$$(p)_{\text{pres}} = (\gamma - 1)\left[E - \frac{1}{2}\frac{\tilde{q}^2}{\rho}\right]_{\text{ff}} \tag{3.73}$$

where the subscript $()_{\text{ff}}$ indicates that the conditions are based upon the far-field approximation developed in section 2.6.3.

In like manner to the subsonic inflow section above, the prescription condition is formed from the chain-rule representation of (3.73) as well as the second and third rows of (3.64), or

$$\begin{pmatrix} \delta p \\ \delta c_2 \\ \delta c_4 \end{pmatrix} = \mathcal{M}_{\text{sub out}} \begin{pmatrix} \delta\rho \\ \delta\tilde{q}_s \\ \delta E \end{pmatrix}_{\text{bc}} \tag{3.74a}$$

where

$$\mathcal{M}_{\text{sub out}} = \begin{pmatrix} \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} & -(\gamma-1)\frac{\tilde{q}}{\rho} & (\gamma-1) \\ \frac{\gamma+1}{2}\frac{\tilde{q}^2}{\rho^2} - \gamma\frac{E}{\rho} & -\frac{\tilde{q}}{\rho} & 1 \\ \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} - \frac{a\tilde{q}}{\rho} & -(\gamma-1)\frac{\tilde{q}}{\rho} + a & (\gamma-1) \end{pmatrix}_{\text{ff}} \tag{3.74b}$$

which is inverted to give

$$\begin{pmatrix} \delta\rho \\ \delta\tilde{q}_s \\ \delta E \end{pmatrix}_{\text{bc}} = \mathcal{M}^{-1}_{\text{sub out}} \begin{pmatrix} p_{\text{pres}} - p_{\text{old}} \\ (\delta c_2)_{\text{pred}} \\ (\delta c_4)_{\text{pred}} \end{pmatrix} \tag{3.75a}$$

where

$$\mathcal{M}^{-1}_{\text{sub out}} = \begin{pmatrix} \frac{1}{a^2} & -\frac{\gamma-1}{a^2} & 0 \\ -\frac{1}{a} + \frac{\tilde{q}}{\rho a^2} & -(\gamma-1)\frac{\tilde{q}}{\rho a^2} & \frac{1}{a} \\ -\frac{\tilde{q}}{\rho a} + \gamma\frac{E}{\rho a^2} - \frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2 a^2} & -\frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2 a^2} & \frac{\tilde{q}}{\rho a} \end{pmatrix}_{\text{ff}} \tag{3.75b}$$

where the terms on the right hand side are evaluated by

$$(p)_{\text{old}} = (\gamma-1)\left[E - \frac{1}{2}\frac{\tilde{q}^2}{\rho}\right]_{\text{old}} \tag{3.76}$$

$$(\delta c_2)_{\text{pred}} = \left[\frac{\gamma+1}{2}\frac{\tilde{q}^2}{\rho^2} - \gamma\frac{E}{\rho}\right]_{\text{ff}} (\delta\rho)_{\text{pred}} + \left[-\frac{\tilde{q}}{\rho}\right]_{\text{ff}} (\delta\tilde{q})_{\text{pred}} + [1]_{\text{ff}} (\delta E)_{\text{pred}} \tag{3.77}$$

and

$$\begin{aligned}(\delta c_4)_{\text{pred}} = &\left[\frac{\gamma-1}{2}\frac{\tilde{q}^2}{\rho^2} - \frac{a\tilde{q}}{\rho}\right]_{\text{ff}} (\delta\rho)_{\text{pred}} \\ &+ \left[-(\gamma-1)\frac{\tilde{q}}{\rho} + a\right]_{\text{ff}} (\delta\tilde{q})_{\text{pred}} + [\gamma-1]_{\text{ff}} (\delta E)_{\text{pred}}\end{aligned} \tag{3.78}$$

Again the streamwise and cross-flow components are related to the Cartesian components using (3.65).

Since the flow inclination information is propagated out of the domain, the correction in the cross-flow momentum is assumed unaltered by the boundary condition application.

Supersonic inflow Examination of (3.58) indicates that for supersonic inflow ($\tilde{q}/\rho > a$) all eigenvalues are positive and hence the boundary values

are set completely by the boundary condition. This makes implementation considerably simpler than for the previous subsonic conditions, especially if the assumption is made that the prescribed boundary values are steady. That being the case here, this boundary condition simply becomes

$$(\delta\rho)_{\text{bc}} = (\delta\tilde{u})_{\text{bc}} = (\delta\tilde{v})_{\text{bc}} = (\delta E)_{\text{bc}} = 0 \tag{3.79}$$

Supersonic outflow For supersonic outflow ($\tilde{q}/\rho < -a$), all eigenvalues are negative in (3.58) and hence no conditions can be prescribed by the boundary condition. Using the assumption that corrections propagating out of the domain are correctly predicted by the distribution formulae, the boundary condition processing can be reduced to not altering the corrections predicted by the scheme itself, or

$$\begin{pmatrix} \delta\rho \\ \delta\tilde{u} \\ \delta\tilde{v} \\ \delta E \end{pmatrix}_{\text{bc}} = \begin{pmatrix} \delta\rho \\ \delta\tilde{u} \\ \delta\tilde{v} \\ \delta E \end{pmatrix}_{\text{pred}} \tag{3.80}$$

3.2.2.2 Solid surface condition

On solid surfaces, application of the above general boundary condition is complicated by the fact that the characteristics generally coincide with the solid surface and thus capturing the effect of the solid surface is difficult. Therefore an alternative approach is taken for implementing solid surface boundary conditions, consisting of three separate operations which are described in turn.

The first operation consists of rotating the corrections predicted by the distribution formulae at boundary nodes so that they are aligned with the wall at the nodes. Here the wall inclination is computed directly from parametric cubic splines of the body nodes, where the parameter is related to the

node indexing. The surface splines are discussed in more detail in section 5.3.1.1.

Given the wall inclination, θ_{wall}, the alignment of the corrections takes the form

$$(\delta\tilde{u})_{\text{bc}} = (\delta\tilde{q})_{\text{wall}} \cos\theta_{\text{wall}} \tag{3.81a}$$

$$(\delta\tilde{v})_{\text{bc}} = (\delta\tilde{q})_{\text{wall}} \sin\theta_{\text{wall}} \tag{3.81b}$$

where

$$(\delta\tilde{q})_{\text{wall}} = (\delta\tilde{u})_{\text{pred}} \cos\theta_{\text{wall}} + (\delta\tilde{v})_{\text{pred}} \sin\theta_{\text{wall}} \tag{3.81c}$$

It is this step which is contained with the processing labeled **BCON** in the overall description of the scheme.

The second operation consists of ensuring that there is no mass flux through solid surfaces and hence only pressure terms contribute to the fluxes along solid surfaces. This is accomplished by modifying (3.24) within the **FLUX** processing to specifically exclude those terms which are associated with mass fluxes through solid surfaces.

It may seem that the above two steps are equivalent, that is if the flow is everywhere tangent to the surface then there is no flow through the surface. Both numerically (in the limit of vanishing step size) and analytically they are equivalent. But when a curved surface is described discretely, a flux through a face bounded by two boundary nodes is possible even if the flow is tangent to the (different) wall slopes at the nodes.

The third operation is concerned with the propagation of corrections from the boundary cell center to the boundary nodes. The basis for the change is that the solid surface is simply another streamline of the inviscid flow. Thus the approximation made in the vicinity of a solid surface should be the same as that near any streamline.

Consider figure 3.4, where the cross-hatched line represents the solid surface, cell A is a boundary cell within the flow domain, and cell B is

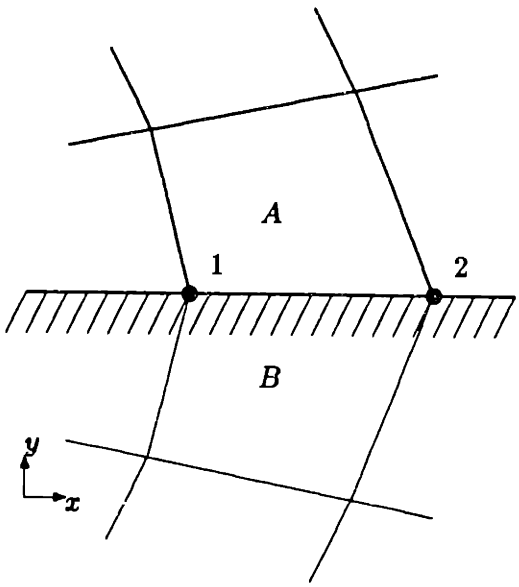


Figure 3.4: Boundary cells

a "ghost" cell outside the domain formed by reflecting cell A about the solid surface. Here the coordinate system has been rotated such that the segment of the solid surface which is of interest is aligned with the x-axis. Any distortions in the reflection which are caused by the solid surface being curved are ignored.

The corrections in the dependent variables at boundary nodes 1 and 2 due to cells A and B are given by the distribution formulae (3.25), or

$$\delta \mathbf{U}_1 = \frac{1}{4}\left[\Delta \mathbf{U}_A - \Delta f_A - \Delta g_A\right] + \frac{1}{4}\left[\Delta \mathbf{U}_B - \Delta f_B + \Delta g_B\right] \qquad (3.82a)$$

and

$$\delta \mathbf{U}_2 = \frac{1}{4}\left[\Delta \mathbf{U}_A + \Delta f_A - \Delta g_A\right] + \frac{1}{4}\left[\Delta \mathbf{U}_B + \Delta f_B + \Delta g_B\right] \qquad (3.82b)$$

Since the solid wall is a streamline, the flow along the wall (the $\tilde{u}$ component) is essentially the same in cells A and B whereas the flow normal to the wall (the $\tilde{v}$ component) is essentially opposite in sign in cells A and B.

Substituting these observations into (3.24) yields

$$\Delta \mathbf{U}_B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Delta \mathbf{U}_A \tag{3.83a}$$

$$\Delta f_B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Delta f_A \tag{3.83b}$$

$$\Delta g_B = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \Delta g_A \tag{3.83c}$$

Substituting (3.83) into (3.82) yields

$$\delta \mathbf{U}_1 = \begin{pmatrix} \delta\rho_1 \\ \delta\tilde{u}_1 \\ \delta\tilde{v}_1 \\ \delta E_1 \end{pmatrix} = \begin{pmatrix} 2(\delta\rho_1)_A \\ 2(\delta\tilde{u}_1)_A \\ 0 \\ 2(\delta E_1)_A \end{pmatrix} \tag{3.84a}$$

and similarly

$$\delta \mathbf{U}_2 = \begin{pmatrix} \delta\rho_2 \\ \delta\tilde{u}_2 \\ \delta\tilde{v}_2 \\ \delta E_2 \end{pmatrix} = \begin{pmatrix} 2(\delta\rho_2)_A \\ 2(\delta\tilde{u}_2)_A \\ 0 \\ 2(\delta E_2)_A \end{pmatrix} \tag{3.84b}$$

Hence for the cells adjacent to solid surface boundaries, the appropriate distribution formulae for the boundary nodes consists of doubling the

corrections predicted by the standard distribution formulae (3.25) and then setting the correction in the normal momentum to zero.

In terms of the corrections for arbitrary boundary orientation, this takes the form

$$(\delta\rho)_{\text{bc}} = 2(\delta\rho)_{\text{pred}} \tag{3.85a}$$

$$(\delta\tilde{u})_{\text{bc}} = 2(\delta\tilde{q})_{\text{pred}} \cos\theta_{\text{wall}} \tag{3.85b}$$

$$(\delta\tilde{v})_{\text{bc}} = 2(\delta\tilde{q})_{\text{pred}} \sin\theta_{\text{wall}} \tag{3.85c}$$

$$(\delta E)_{\text{bc}} = 2(\delta E)_{\text{pred}} \tag{3.85d}$$

where

$$(\delta\tilde{q})_{\text{pred}} = (\delta\tilde{u})_{\text{pred}} \cos\theta_{\text{wall}} + (\delta\tilde{v})_{\text{pred}} \sin\theta_{\text{wall}} \tag{3.86}$$

3.2.2.3 Trailing edge condition

In many respects the boundary condition applied at a sharp trailing edge is more troublesome than either the far-field or solid surface conditions. The general flow boundary condition derived in section 3.2.1 is inappropriate because there are two streams merging at the trailing edge point and choosing a single direction along which to apply the characteristic analysis is impossible. Similarly the fact that locally there are two wall slopes (the upper and lower surface inclinations) precludes application of the solid wall boundary condition described above. In fact, section 2.6.2 shows that the angle at which the flow leaves a sharp wedge-shaped trailing edge depends on the difference between the stagnation pressures of the two flows (upper and lower) just upstream of the trailing edge.

Fortunately it has been observed that if the conditions at the trailing edge are allowed to float (that is if their values are set by the distribution formulae from the surrounding cells without any implicit boundary condition treatment), the resulting solution at the trailing edge point appears acceptable. The exact mechanism by which appropriate values are computed at

a trailing edge node is unknown, but it has been conjectured that it is related to the artificial viscosity in the scheme in much the same way that the physical viscosity is responsible for establishing a Kutta condition[84,69]. Whatever the mechanism, this condition is used here, resulting in

$$(\delta \mathbf{U})_{\text{bc}} = (\delta \mathbf{U})_{\text{pred}} \tag{3.87}$$

at the trailing edge.

3.3 Properties of Ni's scheme

In this section, the integration scheme's order-of-accuracy and consistency, conservation, stability, and wave propagation properties are examined. These properties are examined here because they have not previously been explored rigorously even though the scheme has been in use since 1981.

For simplicity, some of the following analyses are based upon the general one-dimensional hyperbolic equation

$$\mathbf{U}_t + \mathbf{F}_x = 0 \tag{3.88}$$

where the subscripts $()_t$ and $()_x$ represent differentiation. A specific form of (3.88) which is to be examined in detail is the simple one-dimensional wave equation

$$u_t + a u_x = 0 \tag{3.89}$$

for which

$$\mathbf{U} = u \quad \mathbf{F} = au \quad \frac{\partial \mathbf{F}}{\partial \mathbf{U}} = a \tag{3.90}$$

where a is a constant.

Ni's scheme applied to the one-dimensional computational grid shown in figure 3.5 is given by

FLUX for each cell, compute the change in the cell using a one-dimensional form of the flux balance equation (3.24). For example, the flux balance

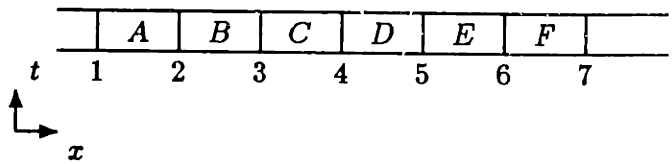


Figure 3.5: Simple grid in space and time

for cell B is given by

$$\Delta \mathbf{U}_B = \frac{\Delta t_B}{\Delta x_B}(\mathbf{F}_2 - \mathbf{F}_3) \tag{3.91}$$

where $\Delta x_B = x_3 - x_2$ and where Δt_B is the time step associated with cell B. For the simple wave equation (3.89) this becomes

$$\Delta u_B = \lambda(u_2 - u_3) \tag{3.92}$$

where

$$\lambda \equiv \frac{a\,\Delta t}{\Delta x} \tag{3.93}$$

In general, the time step parameter λ is taken to be constant across the domain.

DIST for each cell, assign the change to the nodes by the distribution formulae. Again for cell B this becomes

$$(\delta \mathbf{U}_2)_B = \frac{1}{2}\left[1 - \frac{\Delta t_B}{\Delta x_B}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right|_B\right]\Delta \mathbf{U}_B \tag{3.94a}$$

$$(\delta \mathbf{U}_3)_B = \frac{1}{2}\left[1 + \frac{\Delta t_B}{\Delta x_B}\left.\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right|_B\right]\Delta \mathbf{U}_B \tag{3.94b}$$

For the simple wave equation (3.89), these take the form

$$(\delta u_2)_B = \frac{1}{2}[1 - \lambda]\Delta u_B \tag{3.95a}$$

$$(\delta u_3)_B = \frac{1}{2}[1 + \lambda]\Delta u_B \tag{3.95b}$$

UPDT for each node, update the dependent variables by including the corrections from the two adjoining cells. For example for node 3

$$\mathbf{U}_3^* = \mathbf{U}_3 + (\delta\mathbf{U}_3)_B + (\delta\mathbf{U}_3)_C \tag{3.96}$$

Here the superscript $()^*$ indicates the dependent variable at the new time level.

3.3.1 Order-of-accuracy and consistency

The order of accuracy and consistency of a difference formula are both determinable through the creation and examination of its modified differential equation. The latter is the partial differential equation which is actually solved by the difference formula. The difference between the modified and the original differential equations is called the truncation error. The lowest order terms of the truncation error contains the smallest power of the computational grid spacing (in both time and space) and determines the order-of-accuracy of the difference approximation. If these largest terms vanish as the computational grid is refined, the difference formula is said to be consistent with the differential equation.

The modified differential equation is created by substituting Taylor-series expansions into the difference equation and collecting terms of like power of the grid spacing. These Taylor-series expansions

$$\mathbf{U}|_2^* = \mathbf{U}|_2 + \Delta t\, \mathbf{U}_t|_2 + \frac{(\Delta t)^2}{2}\, \mathbf{U}_{tt}|_2 + \frac{(\Delta t)^3}{6}\, \mathbf{U}_{ttt}|_2 + \cdots \tag{3.97}$$

$$\mathbf{F}|_1 = \mathbf{F}|_2 - \Delta x_A\, \mathbf{F}_x|_2 + \frac{(\Delta x_A)^2}{2}\, \mathbf{F}_{xx}|_2 - \frac{(\Delta x_A)^3}{6}\, \mathbf{F}_{xxx}|_2 + \cdots \tag{3.98}$$

and

$$\left.\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right|_B = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right|_2 + \frac{\Delta x_B}{2} \left.\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right)_x\right|_2 + \frac{(\Delta x_B)^2}{8} \left.\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right)_{xx}\right|_2 + \cdots \tag{3.99}$$

are all written about the node 2 and about the old time level.

For $\Delta x_A = \Delta x_B = \Delta x$ and $\Delta t_A = \Delta t_B = \Delta t$, direct substitution yields

$$\mathbf{U}_t + \mathbf{F}_x = \Delta t \left[-\frac{\mathbf{U}_{tt}}{2} + \frac{1}{2} \left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}} \mathbf{F}_x \right)_x \right] \qquad (3.100)$$
$$+(\Delta t)^2 \left[-\frac{\mathbf{U}_{ttt}}{6} \right] + (\Delta x)^2 \left[-\frac{\mathbf{F}_{xxx}}{6} \right] + \cdots$$

where all terms are evaluated at node 2 and the old time. The $\mathbf{U}_{tt}$ on the right side of (3.100) can be eliminated by differentiating the governing equation (3.88), or

$$\mathbf{U}_{tt} + \mathbf{F}_{xt} = \mathbf{U}_{tt} + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}} \mathbf{U}_t \right)_x = \mathbf{U}_{tt} - \left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}} \mathbf{F}_x \right)_x = 0 \qquad (3.101)$$

so that (3.100) becomes

$$\mathbf{U}_t + \mathbf{F}_x = (\Delta t)^2 \left[-\frac{\mathbf{U}_{ttt}}{6} \right] + (\Delta x)^2 \left[-\frac{\mathbf{F}_{xxx}}{6} \right] + \cdots \qquad (3.102)$$

which is the modified differential equation for Ni's scheme in one dimension with constant Δx and Δt. It should be noted that the exact terms in the brackets on the right-hand side of (3.102) will be slightly different if the modified (instead of the original) differential equation is used to eliminate $\mathbf{U}_{tt}$, but that the order of the approximation will not be altered.

Equation (3.102) indicates that Ni's scheme is second order accurate in both time and space, that is the leading truncation error terms are $O[(\Delta x)^2, (\Delta t)^2]$. Additionally, as the computational grid is refined, the truncation error vanishes as long as Δx and Δt both approach zero, implying that the scheme forms a consistent approximation to (3.88), as is expected for Lax-Wendroff methods.

In two dimensions, the modified differential equation is developed in the same way but is algebraically much more complicated due to the added complexity of the flux balance and distribution formulae as well as the fact that the Taylor series expansions involve two space dimensions. For an evenly spaced grid (constant but possible unequal Δx Δy and Δt) as shown in figure 3.6a, the direct substitution of the Taylor series into the flux balance

(3.24), distribution (3.25), and update (3.26) formulae yields (after simplification with the governing equation and its derivatives)

$$\mathbf{U}_t + \mathbf{F}_x + \mathbf{G}_y = (\Delta x)^2 \left[\frac{\mathbf{U}_{txx}}{6} - \frac{\mathbf{G}_{xxy}}{12} \right] + (\Delta y)^2 \left[\frac{\mathbf{U}_{tyy}}{6} - \frac{\mathbf{F}_{xyy}}{12} \right] + (\Delta t)^2 \left[-\frac{\mathbf{U}_{ttt}}{6} \right] + \cdots \tag{3.103}$$

Again the Ni scheme provides a consistent approximation to the governing equations, and is second order accurate in both time and space.

In general, Ni's scheme is not applied to uniform grids but rather to grids which are stretched, skewed, curved, and convergent as shown in figures 3.6b-e. It is known that each of these grid properties can have a detrimental effect on the accuracy of finite difference and finite volume approximations. In the following, the accuracy of Ni's scheme is determined as a function of each of these effects. The zero-th and first order terms of the resulting modified differential equations are presented in table 3.1. In each case it is understood that second and higher order terms are also present.

Stretched grid As shown in figure 3.6b, the grid is stretched independently in the two coordinate directions, with the stretching factors given by α and β. The modified differential equation is given on the second line of table 3.1. Notice that the approximation is now inconsistent due to the $\Delta t/\Delta x$ and $\Delta t/\Delta y$ terms. Fortunately however, these terms vanish in the steady state and the solution is consistent with that for the steady state of the governing equation. One can also note that all the first order terms similarly vanish in the steady state, leaving a second order accurate approximation.

Skewed grid Figure 3.6c depicts a skewed grid, where the grid intersection angle is ϕ (an orthogonal grid is given by $\phi = \pi/2$). The modified differential equation, as given in table 3.1, indicates that the order of the truncation error is unaltered by skew and hence the approximation remains consistent and second order accurate.

Curved grid Another grid non-uniformity in two dimensions results

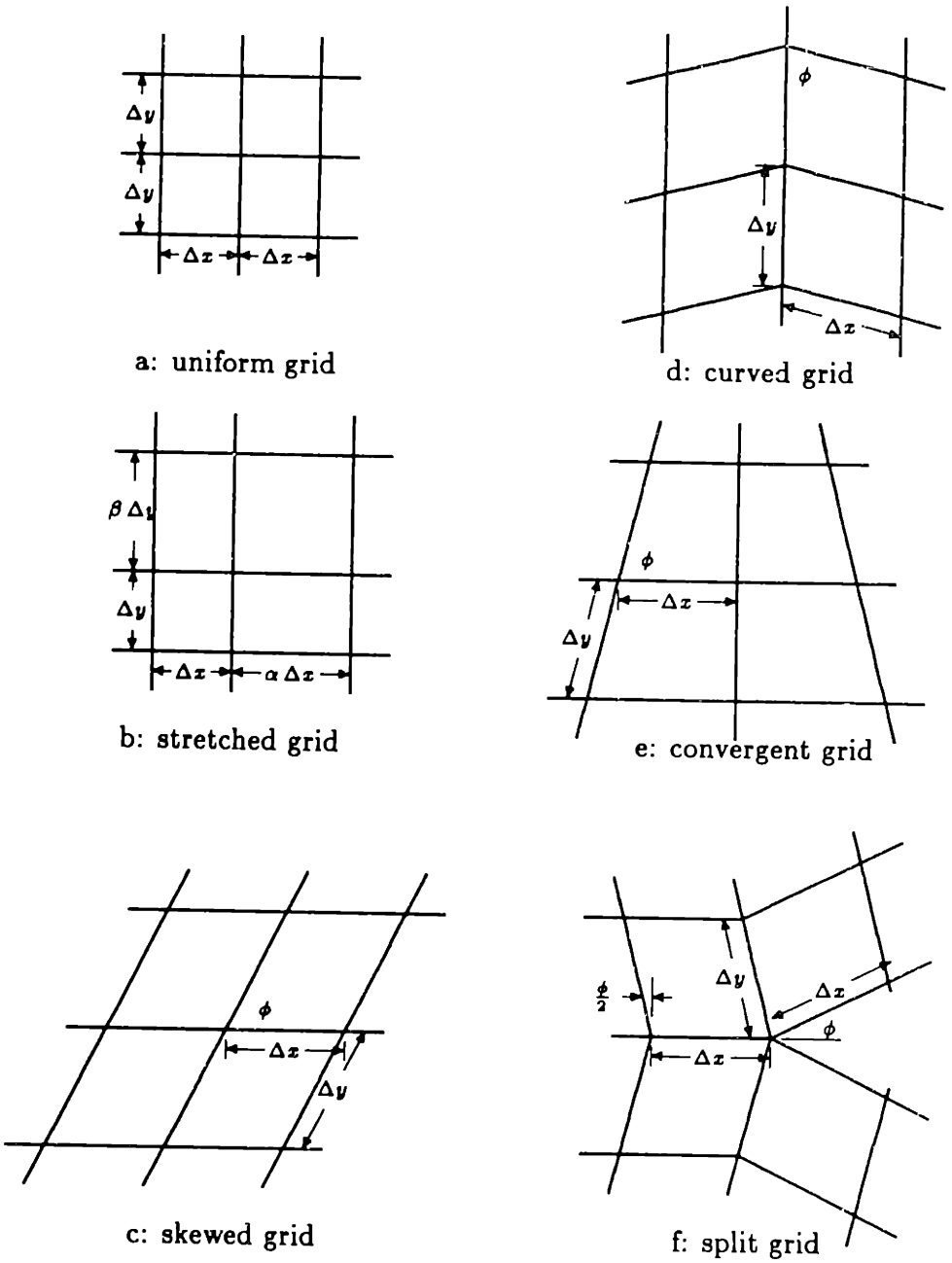


Figure 3.6: Two-dimensional grids

	uniform	stretched	skewed	curved
$\frac{\Delta t}{\Delta x}$	0	$\frac{\alpha-1}{2\alpha}\mathbf{F}_t$	0	0
$\frac{\Delta t}{\Delta y}$	0	$\frac{\beta-1}{2\beta}\mathbf{G}_t$	0	0
$\frac{\Delta y}{\Delta x}$	0	0	0	0
Δx	0	$\frac{\alpha-1}{4}\mathbf{U}_{tx}$	0	$\frac{\cos\phi}{2}(\mathbf{F}_{yy}+\mathbf{G}_{yy})$
Δy	0	$\frac{\beta-1}{4}\mathbf{U}_{ty}$	0	0
$\frac{\Delta t\Delta y}{\Delta x}$	0	$\frac{(\alpha-1)(\beta-1)}{8\alpha}\mathbf{F}_{ty}$	0	0
$\frac{\Delta t\Delta x}{\Delta y}$	0	$\frac{(\alpha-1)(\beta-1)}{8\beta}\mathbf{G}_{tx}$	0	0
$\frac{\Delta t(\Delta y)^2}{(\Delta x)^2}$	0	0	0	0
$\frac{(\Delta y)^2}{\Delta x}$	0	0	0	0

	convergent	split
1	$\frac{\alpha+\beta-2\alpha\beta}{2\alpha\beta}\mathbf{U}_t$	0
$\frac{\Delta t}{\Delta x}$	$\frac{\alpha^2+\beta^2}{4\alpha^2\beta^2}\sin\phi\cos\phi\,\mathbf{G}_t$	0
$\frac{\Delta t}{\Delta y}$	$\frac{\beta^2-\alpha^2}{2\alpha^2\beta^2}\mathbf{G}_t$	$\frac{\sin\frac{\phi}{2}}{2\cos\frac{\phi}{2}}\mathbf{F}_t$
$\frac{\Delta y}{\Delta x}$	$\frac{\beta-\alpha}{4\alpha\beta}\sin\phi\cos\phi\,\mathbf{U}_t$	0
Δt	$\chi\mathbf{U}_{tt}+\chi\mathbf{G}_{xx}$	0
Δx	$\frac{\alpha+\beta}{8\alpha\beta}\cos\phi\,\mathbf{G}_{xx}$	$\frac{\cos\phi-1}{4}\mathbf{U}_{tx}+\frac{\cos\phi(1-\cos\phi)}{4}\mathbf{G}_{xy}$
Δy	$\frac{\alpha-\beta}{4\alpha\beta}\left(\sin^2\phi\,\mathbf{U}_{ty}-\frac{1}{2}\cos^2\phi\mathbf{G}_{xx}\right)$	$-\frac{\sin\frac{\phi}{2}}{2}\mathbf{U}_{tx}$
$\frac{\Delta t\Delta y}{\Delta x}$	$\frac{\beta^2-\alpha^2}{4\alpha^2\beta^2}\sin^2\phi\cos\phi\mathbf{U}_{tt}$	0
$\frac{\Delta t\Delta x}{\Delta y}$	$\frac{\beta^2-\alpha^2}{8\alpha^2\beta^2}\cos\phi\frac{\partial\mathbf{G}}{\partial\mathbf{U}}\mathbf{G}_{xx}$	$\frac{\sin 2\phi}{2\cos\frac{\phi}{2}}(\mathbf{F}_{tx}-\mathbf{G}_{ty})$
$\frac{\Delta t(\Delta y)^2}{(\Delta x)^2}$	$\frac{\alpha^2+\beta^2}{8\alpha^2\beta^2}\sin^2\phi\cos^2\phi\left(-\mathbf{F}_{tx}-\frac{\partial\mathbf{F}}{\partial\mathbf{U}}\mathbf{G}_{xy}\right)$	0
$\frac{(\Delta y)^2}{\Delta x}$	$-\frac{\alpha+\beta}{4\alpha\beta}\sin^2\phi\cos\phi\,\mathbf{U}_{ty}$	0

Note: χ represents a complicated combination of the independent parameters.

Table 3.1: Zero-th and first order terms of the modified differential equation.

from a curvature of the grid as shown in figure 3.6d. Here the curvature is measured by ϕ, where an uncurved grid corresponds to $\phi = \pi/2$. The modified differential equation shows that this approximation is consistent with the differential equation but is only first order accurate, even at steady state. Fortunately for small curvatures ($\phi - \pi/2 = O(\Delta x)$), the $\cos\phi$ term is proportional to Δx and the accuracy reverts to second order in space.

Convergent grid Here one set of grid lines are not parallel but instead converge as shown in figure 3.6e. The amount of convergence is measured by ϕ where the grid lines are parallel when $\phi = \pi/2$. Due to the complexity of the area terms for such a configuration, define the terms

$$\alpha = \sin\phi + \frac{1}{2}\frac{\Delta y}{\Delta x}\sin\phi\cos\phi \tag{3.104a}$$

$$\beta = \sin\phi - \frac{1}{2}\frac{\Delta y}{\Delta x}\sin\phi\cos\phi \tag{3.104b}$$

The modified differential equation which results again shows the approximation to be inconsistent in time but reverts to a consistent approximation in the steady state. The order of accuracy is first order in the steady state due to the terms of the form G_{xx}. Fortunately each of these terms are multipled by at least $\cos\phi$ so that for mildly convergent grids ($\phi - \pi/2 \approx \Delta x$), second order accuracy is recovered as above.

Split grid This grid configuration, shown in figure 3.6f, is the type which occurs at leading edges for H-type grids and at trailing edges for either H-type of C-type grids. Here the parameter ϕ is the semi-wedge angle of the body; all grid lines of the opposite family are assumed to be inclined at $\phi/2$. The modified differential equation which results for this singular node is consistent only in the steady state and is then only formally first order accuracy. As above, the order of accuracy can be increased to second order if $\phi \approx 0$. Unfortunately for this case, ϕ is set by the geometry and in general cannot be adjusted arbitrarily in order to achieve more accurate results. For example at a blunt edge, $\phi \to \pi/2$ as the mesh is refined, resulting in only

first order accuracy. Fortunately however, this error only occurs at a single node; its effect on the global solution diminishes as the mesh is refined (that is its effect is felt over a smaller portion of the entire domain), resulting in globally second order accurate solutions.

Care must be taken in the limiting processes above. Often the grid becomes smoother (more uniform) as the grid is refined. Thus in many instances, the terms α, β, etc. approach their limiting values as the mesh is refined, resulting in a higher order of accuracy than that discussed above.

The order of accuracy of the scheme can be demonstrated numerically by evaluating the errors for solutions made on successively refined grids. For this purpose the model problem shown in figure 3.7a was employed, consisting of a parallel-walled channel of length $3L$ and height L with a bump along the lower wall given by the equation

$$y = 0.10 \sin^2 \left(\frac{(x - L)\pi}{L} \right) \qquad L \leq x \leq 2L \tag{3.105}$$

The free-stream flowing left to right at a Mach number of 0.50. For this case the flow should be completely subsonic and hence the stagnation pressure loss, defined by

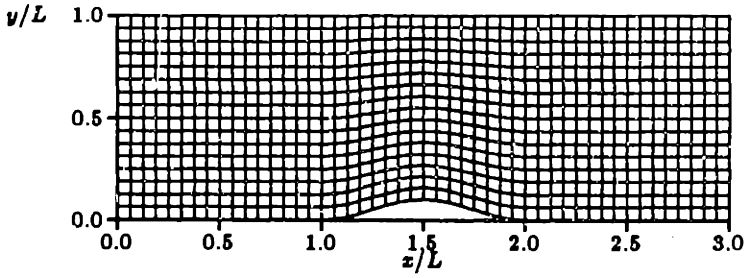
$$\eta = \frac{(p_o)_{\text{upstream}} - (p_o)_{\text{local}}}{(p_o)_{\text{upstream}}} \tag{3.106}$$

should remain constant across the domain. A suitable measure of accuracy is then the L_2-norm of η, or

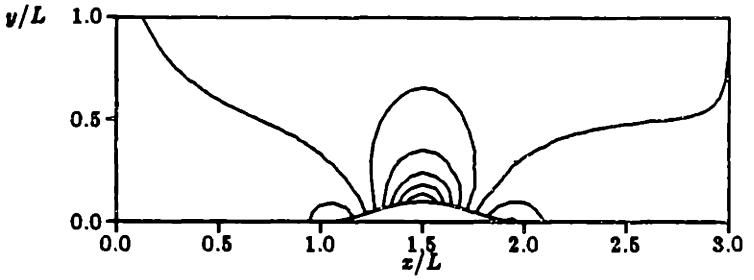
$$\epsilon = \sqrt{\frac{\sum_{i=N} \eta_i^2}{N}} \tag{3.107}$$

where N is the total number of nodes in the computational domain.

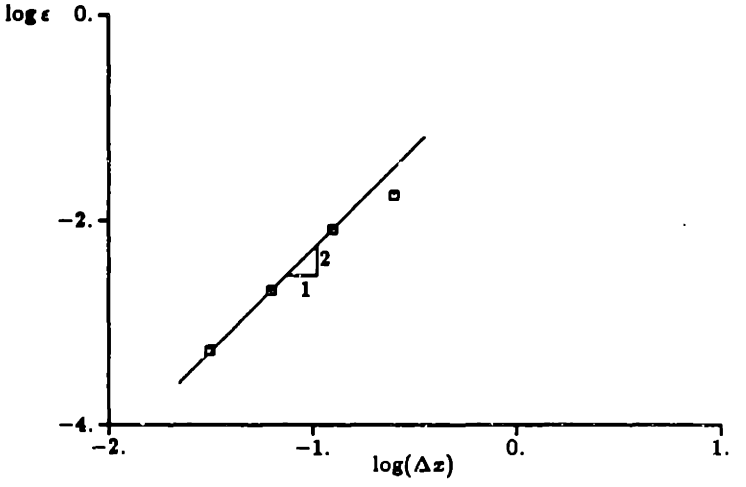
Computations were performed on 13×5, 25×9, 49×17, and 97×33 grids, the third of which is shown in figure 3.7a, with the Mach number contours on the 97×33 grid shown in figure 3.7b. The solution appears very smooth and symmetric as expected for this case except near the exit



a: computational grid, 49 × 17



b: Mach number contours on 97 × 33 grid, $\Delta M = 0.05$



c: accuracy versus grid resolution

Figure 3.7: Demonstrated accuracy on subsonic test case – nearly uniform grid

boundary plane where the discrepancy is due to the imposition of a constant exit static pressure.

The accuracy of the corresponding solutions as shown in figure 3.7c along with a line of slope 2. The closeness of the computed errors to the line indicate that second order accuracy has been achieved.

3.3.2 Stability

The stability of a difference approximation refers to its ability to damp out errors which arise either from the initial conditions or from arithmetic round-offs occurring through the iterative (time integration) process. Various methods have been developed for analyzing the stability of a difference approximation, including the Fourier or von Neumann analysis and the energy methods.

In the von Neumann analysis, which is applicable only for linear equations and periodic boundary conditions, the error is represented as a Fourier series. The difference approximation is said to be stable if the magnitude of the amplification factor for each wave number in the Fourier series is less than unity. This stability method is limited in that it cannot include the effects of boundary conditions and uneven grid spacing.

In the energy method, one begins by forming some norm of the solution vector. The difference approximation is said to be stable if the norm decreases at each time step. This technique can easily include the effects of variable coefficients, boundary conditions, and unequal grid spacing. The difficulty with this technique in general is that for complex schemes and/or complex governing equations, the selection of an appropriate norm is extremely difficult. Fortunately in the simplest cases, the solution's energy is a suitable choice (and hence the name of the method). Since the current governing equations and Ni's scheme fall into this latter category, the energy method is pursued here.

shift	$T_x^n u_i \equiv u_{i+n}$
average	$\mu_{nx} u_i \equiv \frac{1}{2}\left(u_{i+\frac{n}{2}} + u_{i-\frac{n}{2}}\right) = \frac{1}{2}\left(T_x^{\frac{n}{2}} + T_x^{-\frac{n}{2}}\right) u_i$
central difference	$\delta_{nx} u_i \equiv u_{i+\frac{n}{2}} - u_{i-\frac{n}{2}} = \left(T_x^{\frac{n}{2}} - T_x^{-\frac{n}{2}}\right) u_i$
inner product	$(u,v)_r^s \equiv \sum_{i=r}^{s} u_i v_i$
norm	$\|u\|_r^s \equiv \sqrt{(u,u)_r^s}$

Table 3.2: Definition of operators

$\delta_{2x} u = (2\delta_x \mu_x) u$	I-1
$\delta_x^2 u = (4\mu_x^2 - 4) u$	I-2
$\delta_x (uv) = (\mu_x u)(\delta_x v) + (\mu_x v)(\delta_x u)$	I-3
$\mu_{2x} u = (2\mu_x^2 - 1) u$	I-4
$\mu_x^2 u = (\frac{1}{4}\delta_x^2 + 1) u$	I-5
$\mu_x (uv) = \frac{1}{4}(\delta_x u)(\delta_x v) + (\mu_x u)(\mu_x v)$	I-6
$\|\delta_x u\|^2 = 4\|u\|^2 - 4\|\mu u_x\|^2$	I-7
$(\delta_x^a \mu_x^b u, \delta_x^c \mu_x^d v) = (-1)^a (u, \delta_x^{a+c} \mu_x^{b+d} v) = (-1)^c (\delta_x^{a+c} \mu_x^{b+d} u, v)$	I-8

Table 3.3: Operator identities

To begin, it is convenient to introduce the operators defined in Table 3.2 which are used in the stability analysis. Here u_i denotes the dependent variable u at $x = i\,\Delta x, \quad i = 0, \ldots, I$. The primary operator is the shift operator, T_x^n, from which one can construct the averaging operator μ_{nx} and the central difference operator δ_{nx}. The final two definitions in Table 3.2 are for the inner product and the norm. In addition, some useful operator identities are listed in Table 3.3. Each of these identities are derived by writing the operators in terms of the shift operator T_x^n and using the associative, commutative, and distributive properties.

For the simple one-dimensional wave equation (3.89), the flux balance (3.92)

can be written as

$$(\Delta u)_{i+\frac{1}{2}} = \{-\lambda\delta_x\}\, u_{i+\frac{1}{2}} \tag{3.108}$$

and the distribution (3.95) and update (3.96) formulae by

$$u_i^* = u_i + \mu(\Delta u)_i - \frac{\lambda}{2}\delta(\Delta u)_i \tag{3.109}$$

Equations (3.108) and (3.109) can be combined to yield Ni's scheme in operator notation

$$u_i^* = \left[1 - \lambda\mu_x\delta_x + \frac{\lambda^2}{2}\delta_x^2\right] u_i \tag{3.110}$$

As stated above, it has been found that an appropriate norm for this governing equation and Ni's scheme is simply the total system energy, $\|u\|^2$. The energy of the solution vector after a time step is given by

$$\|u^*\|^2 = (u^*, u^*) \tag{3.111}$$

which using (3.110) becomes

$$\|u^*\|^2 = \left([1 - \lambda\mu\delta + \frac{\lambda^2}{2}\delta^2]u, [1 - \lambda\mu\delta + \frac{\lambda^2}{2}\delta^2]u\right) \tag{3.112}$$

where the x-subscript on μ and δ has been deleted because of the lack of ambiguity in this case. Expanding (3.112) out term-by-term yields

$$\begin{aligned}\|u^*\|^2 &= (u,u) + (-\lambda\mu\delta u, -\lambda\mu\delta u) + \left(\frac{\lambda^2}{2}\delta^2 u, \frac{\lambda^2}{2}\delta^2 u\right) \\ &\quad +2\,(u, -\lambda\mu\delta u) + 2\left(u, \frac{\lambda^2}{2}\delta^2 u\right) + 2\left(-\lambda\mu\delta u, \frac{\lambda^2}{2}\delta^2 u\right) \\ &= (u,u) + \lambda^2(\mu\delta u, \mu\delta u) + \frac{\lambda^4}{4}(\delta^2 u, \delta^2 u) \\ &\quad -2\lambda(u, \mu\delta u) + \lambda^2(u, \delta^2 u) - \lambda^3(\mu\delta u, \delta^2 u)\end{aligned} \tag{3.113}$$

Using identity I-8, one finds that

$$(\delta^a\mu^b u, \delta^c\mu^d u) = \begin{cases} (-1)^c(u, \delta^{a+c}\mu^{b+d}u) & a+c = \text{even} \\ 0 & a+c = \text{odd} \end{cases} \tag{3.114}$$

so that the fourth and sixth terms on the right hand side of (3.113) vanish, leaving (after applying I-8 to the remaining terms)

$$\|u^*\|^2 = (u,u) - \lambda^2(u,\mu^2\delta^2 u) + \frac{\lambda^4}{4}(u,\delta^4 u) + \lambda^2(u,\delta^2 u) \qquad (3.115)$$

Now I-5 can be applied to the second term on the right hand side of (3.115), yielding

$$\|u^*\|^2 = (u,u) + \frac{\lambda^2}{4}(\lambda^2 - 1)(\delta^2 u, \delta^2 u) \qquad (3.116)$$

To prove stability, it has to be shown that this norm decreases in time, or

$$\|u^*\|^2 \leq \|u\|^2 \qquad (3.117)$$

Combining (3.116) and (3.117) yields

$$\|u^*\|^2 - \|u\|^2 = \frac{\lambda^2}{4}(\lambda^2 - 1)\left\|\delta^2 u\right\|^2 \leq 0 \qquad (3.118)$$

from which it follows that

$$\lambda \leq 1 \quad \Rightarrow \quad \Delta t \leq \frac{\Delta x}{a} \qquad (3.119)$$

This constraint is the so-called Courant-Friedrichs-Lewy (CFL) condition and is the same as determined by a von Neumann (Fourier) analysis. This latter point, which at first glance seems trivial, is not so in general since the von Neumann analysis yields necessary stability conditions whereas the energy method yields only sufficient conditions for stability. Richtmyer and Morton suggest that the energy method's inability to provide actual stability may be tied to either the "lack of ingenuity in devising a norm or because of the inherent limitations of the method"[73, pg 133].

A stability proof for the simple wave equation in two-dimensions

$$u_t + au_x + bu_y = 0 \qquad (3.120)$$

is more complicated, resulting in a final expression of the form

$$\|u^*\|^2 = \|u\|^2 + \alpha\left\|\delta_x^2 u\right\|^2 + \beta\left\|\delta_y^2 u\right\|^2 + \gamma\left\|\delta_x\delta_y u\right\|^2 + \cdots \qquad (3.121)$$

where α, β, and γ are complex functions of a, b, Δt, Δx, and Δy. Without a firm method for relating the magnitude of the norms on the right hand side, it is impossible to make any statements about the stability of the method except for the overly restrictive requirement that $\alpha \leq 0$, $\beta \leq 0$, and $\gamma \leq 0$.

Fortunately Usab[84] provides a von Neumann analysis on Ni's scheme in two dimensions, yielding the stability limit

$$\Delta t \leq \sqrt{\frac{1}{\left(\frac{a}{\Delta x}\right)^2 + \left(\frac{b}{\Delta y}\right)^2}} \tag{3.122}$$

He further extends this stability result to the Euler equations by locally using the maximum eigenvalue in place of a and b in (3.122). After much algebra, the final stability condition for the Euler equations becomes

$$\Delta t \leq \min\left\{\frac{2A}{|u\Delta y^l - v\Delta x^l| + a\Delta l}, \frac{2A}{|u\Delta y^m - v\Delta x^m| + a\Delta m}\right\} \tag{3.123}$$

where A is the cell area, u, v, and a are the x-velocity, y-velocity, and speed of sound respectively at the cell center, and where

$$\Delta x^l = -x_{sw} - x_{se} + x_{ne} + x_{nw} \tag{3.124a}$$

$$\Delta y^l = -y_{sw} - y_{se} + y_{ne} + y_{nw} \tag{3.124b}$$

$$\Delta x^m = -x_{sw} + x_{se} + x_{ne} - x_{nw} \tag{3.124c}$$

$$\Delta y^m = -y_{sw} + y_{se} + y_{ne} - y_{nw} \tag{3.124d}$$

$$\Delta l = \sqrt{(\Delta x^l)^2 + (\Delta y^l)^2} \tag{3.124e}$$

$$\Delta m = \sqrt{(\Delta x^m)^2 + (\Delta y^m)^2} \tag{3.124f}$$

This the the same stability restriction presented by Ni[67] for his transonic computations. If higher Mach number solutions are required (for example hypersonics), Ni has found that the stability requirement above must include the $1/\sqrt{2}$ term as in (3.122) above[68]. In practice for transonic flows, a slightly smaller time is used (say 95 percent of the theoretical maximum) in order to ensure that non-linearities in the Euler equations do not

yield numerical instabilities. For all computations done in this work, this smaller time step is used, yielding stable computations for all configurations attempted.

3.3.3 Conservation

In section 3.3.1 it was shown that the Ni scheme forms a consistent approximation to the governing equations at each computational node and is at least first order accurate in the steady state. It is also important to determine the global accuracy of the scheme. Since the governing equations are statements of conservation of mass, momentum, and energy, a global measure of the accuracy consists of measuring how well these quantities are globally conserved; this is known as a scheme's *conservation property.*

To examine the conservation property of a difference scheme, one sums (over all nodes in the domain) the corrections in the dependent variables predicted by the difference formulae. The scheme is said to be conservative if the only contributions after the summation come from boundary nodes and cells.

For the Ni scheme, the corrections at each node (which are denoted by $\delta\mathbf{U}$) are the sums of the distributed corrections from the adjoining cells as given by the distribution formulae (3.25). The sum of the total correction over all nodes in the computational domain ($\sum_{\text{all nodes}}(\delta\mathbf{U})$) is thus equivalent to the sum of the distributed corrections from each cell (plus perhaps boundary conditions), or

$$\sum_{\text{all nodes}} (\delta\mathbf{U}) = \sum_{\text{all cells}} \left(\sum_{\text{adjoining nodes}} (\delta\mathbf{U}) \right) + \text{boundary conditions} \quad (3.125)$$

The inner sum on the right hand side is simply the sum of the distributed corrections from any one cell, or

$$\sum_{\text{adjoining nodes}} (\delta\mathbf{U}) = + \frac{1}{4}[\Delta\mathbf{U} - \Delta f + \Delta g]$$

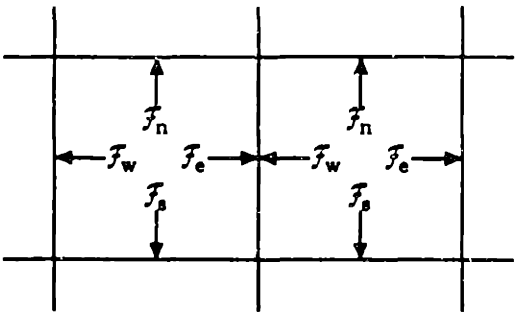


Figure 3.8: Flux contributions on adjacent cells

$$+\frac{1}{4}[\Delta\mathbf{U}+\Delta f-\Delta g] \tag{3.126}$$
$$+\frac{1}{4}[\Delta\mathbf{U}+\Delta f+\Delta g]$$
$$+\frac{1}{4}[\Delta\mathbf{U}-\Delta f-\Delta g]$$

which is

$$\sum_{\text{adjoining nodes}}(\delta\mathbf{U})=\Delta\mathbf{U} \tag{3.127}$$

Thus (3.125) can be rewritten in the form

$$\sum_{\text{all nodes}}(\delta\mathbf{U})=\sum_{\text{all cells}}(\Delta\mathbf{U})+\text{boundary conditions} \tag{3.128}$$

which is simply the sum of the flux balances over all cells.

To determine the sum of the flux balances (3.24) over all cells, consider the two cells shown in figure 3.8. Here the appropriate contributions to the flux balances

$$\Delta\mathbf{U}=-\frac{\Delta t}{A}[\mathcal{F}_s+\mathcal{F}_e+\mathcal{F}_n+\mathcal{F}_w] \tag{3.129}$$

are shown for each of the cells. Notice that for the common face, the sense of the flux $\mathcal{F}$ for adjoining cells is opposite to one another so that for a sum over all cells, those fluxes would cancel as long as the time step ratio $\Delta t/A$ was constant. This then leaves

$$\sum_{\text{all cells}}(\Delta\mathbf{U})=\frac{\Delta t}{A}\sum_{\text{boundary faces}}\mathcal{F} \tag{3.130}$$

so that (3.128) becomes

$$\sum_{\text{all nodes}} (\delta \mathbf{U}) = \frac{\Delta t}{A} \sum_{\text{boundary faces}} \mathcal{F} + \text{boundary conditions} \qquad (3.131)$$

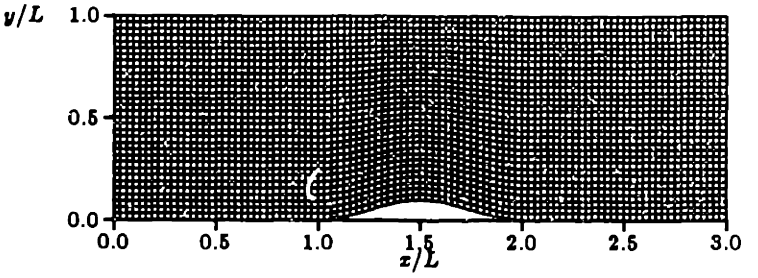
Equation (3.131) indicates that the only contributions to the sum of the corrections over the entire domain are boundary effects and thus Ni's scheme is conservative.

A numerical demonstration follows from the computed solution for the subsonic channel flow case discussed above. A 97×33 computational grid is shown in figure 3.9a and the corresponding Mach number contours shown in figure 3.9b. To show that this scheme is conservative, the mass flux rate across the channel is plotted as a function of axial position; the constant value shows the degree of conservation in the scheme.

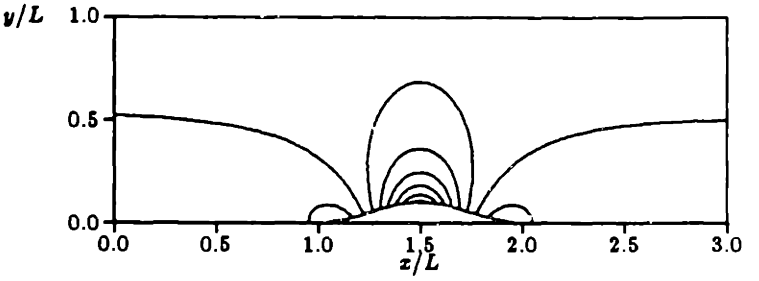
3.3.4 Wave propagation

Ni's scheme has been derived for a system of hyperbolic, first order partial differential equations. These equations are wave-like in nature, propagating information as predicted by the theory of characteristics. In this section, the wave propagation characteristics of Ni's scheme are examined. For illustrative purposes, the one-dimensional wave equation (3.89) is considered.

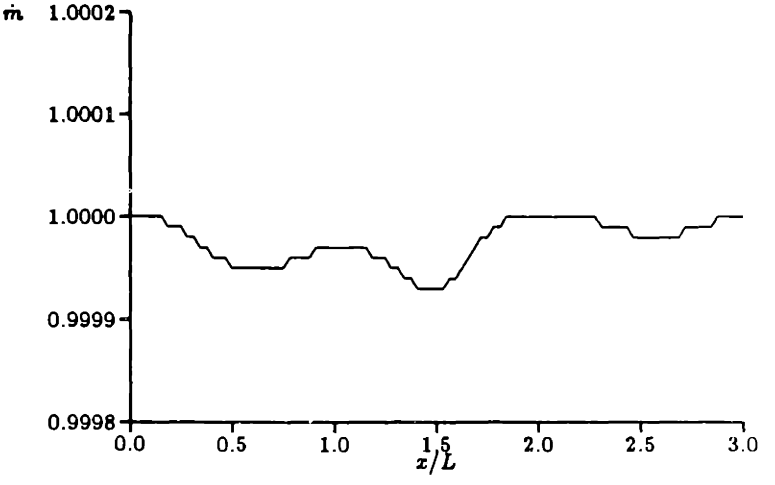
Consider now figure 3.10 in which a simple discontinuity (shown on the second line of the figure) propagates to the right. For this uniform grid, a constant CFL-number $\lambda = 1$ is used. Applying the flux balance equations (3.92) results in the computation of the "change" (Δu) for each cell as shown on the next line; here only the cell with the discontinuity has a non-zero flux balance. The distribution formulae (3.95) are then applied, resulting in the cell change (Δu) being completely distributed to the upstream node. Finally the dependent variable is updated with the "correction" δu. As a result of one time step, the original discontinuity has propagated one cell to the right



a: computational grid 97 × 33



b: Mach number contours, $\Delta M = 0.05$



c: mass flux rate versus axial position

Figure 3.9: Demonstrated conservation for the subsonic test case

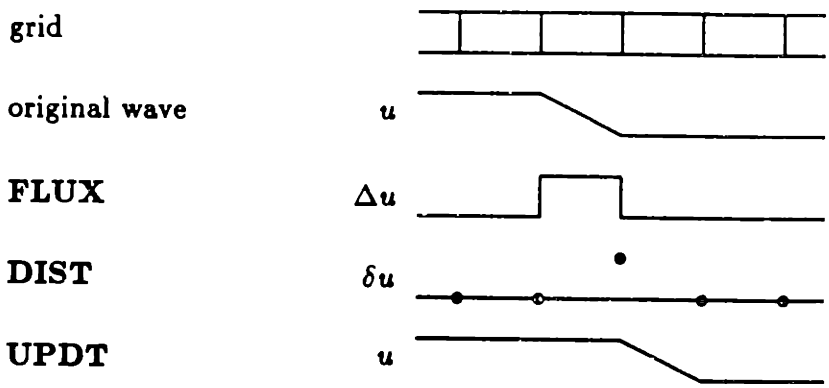


Figure 3.10: Propagation of simple discontinuity, $\lambda = 1.0$

without any dispersion or dissipation.

In general for systems of equations, different physical quantities propagate at different wave speeds. Thus with a single time step, Δt, only the fastest wave can have an effective $\lambda = 1$ and all the others will have smaller effective values of λ. To examine the effect of this, the case discussed above is repeated in figure 3.11, but this time with $\lambda = 0.7$. Here it can be seen that the flux balance does not capture the total "change" in the original wave, and further that the distribution formulae (3.95) now propagates some information upstream (which is non-physical). The net effect is a propagation which is both dispersive and dissipative.

To demonstrate this propagating character, a series of cases were computed with 3, 6, 12, 24, and 48 cells, with the solution for the 12-cell and 48-cell cases shown in figures 3.12a and 3.12b respectively. Here the solution at every third time step is shown, shifted slightly up and to the right. It can be seen that the wave front diffuses on successive iterations.

The results of each of the above cases is summarized in figure 3.13 in which the logarithm of the maximum correction in the dependent variable (over the whole domain) is plotted versus iteration number. For each case, the figure clearly shows an initial plateau while the wave propagates to the

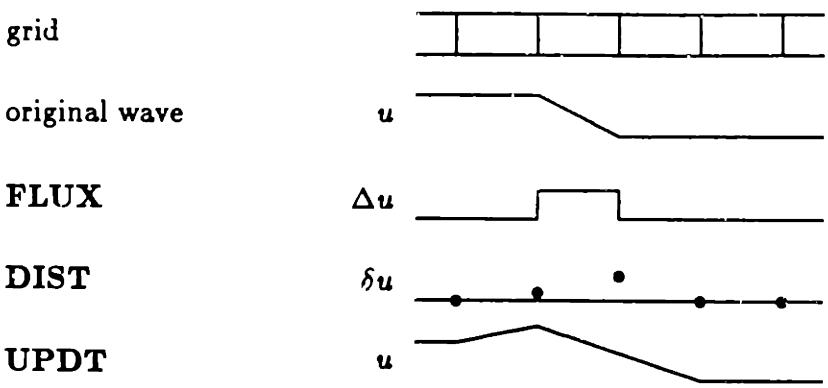
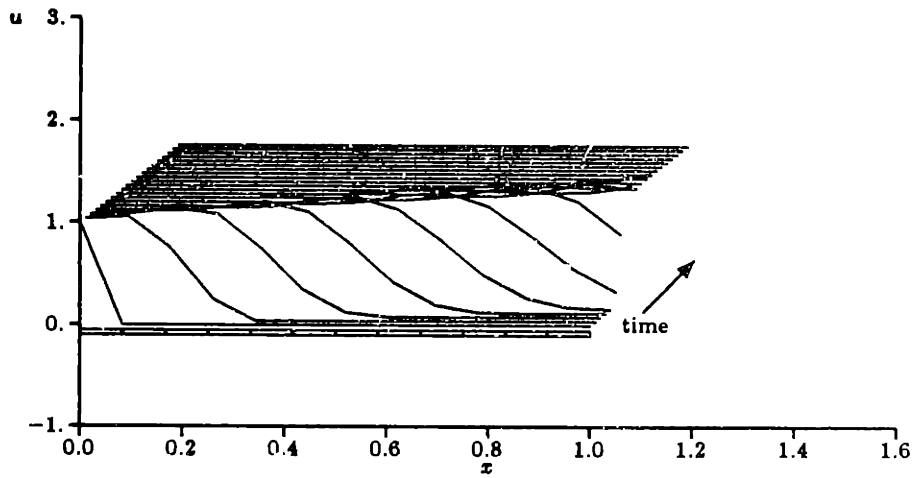


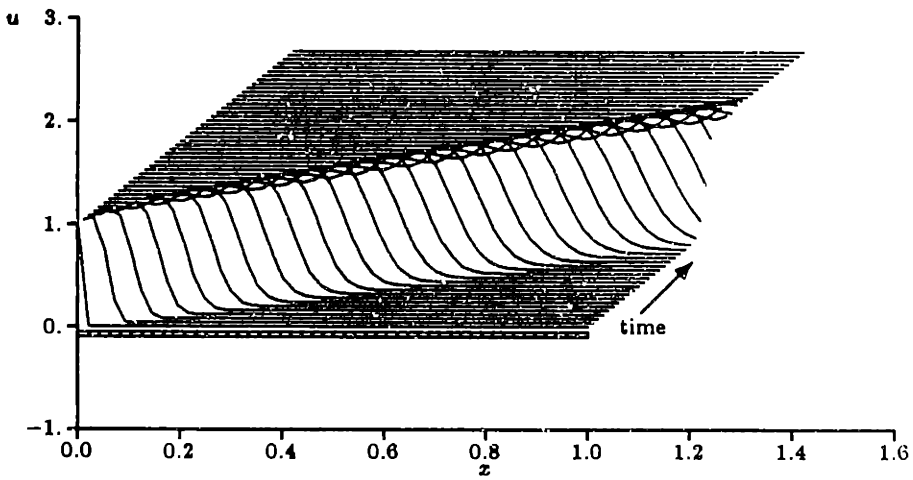
Figure 3.11: Propagation of simple discontinuity, $\lambda = 0.7$

domain exit, followed by a decreasing maximum as the errors introduced through the diffusive mechanism are dissipated. In each curve, the end of the plateau exactly corresponds to $\frac{\text{number of cells}}{\lambda}$ as expected. Also, the rate at which the errors decay is a weak function of the number of cells in the domain.

Similar results are computed for one-dimensional, subsonic Euler flow in a convergent-divergent nozzle as shown in figure 3.14, where here the ordinate represents the logarithm of the maximum correction in Mach number. The most striking difference between this figure and figure 3.13 is that the initial plateaus are missing. This is primarily due to the fact that the prescription boundary conditions are used for the Euler calculations and hence waves are constantly being created at the boundaries to drive the flow to the desired conditions. This effect can be seen clearly by the beating in the convergence history plots as the waves bounce between the nozzle's inlet and exit.



a: 12-cell domain



b: 48-cell domain

Figure 3.12: Demonstrated wave propagation for $u_t + au_x = 0$, $\lambda = 0.7$

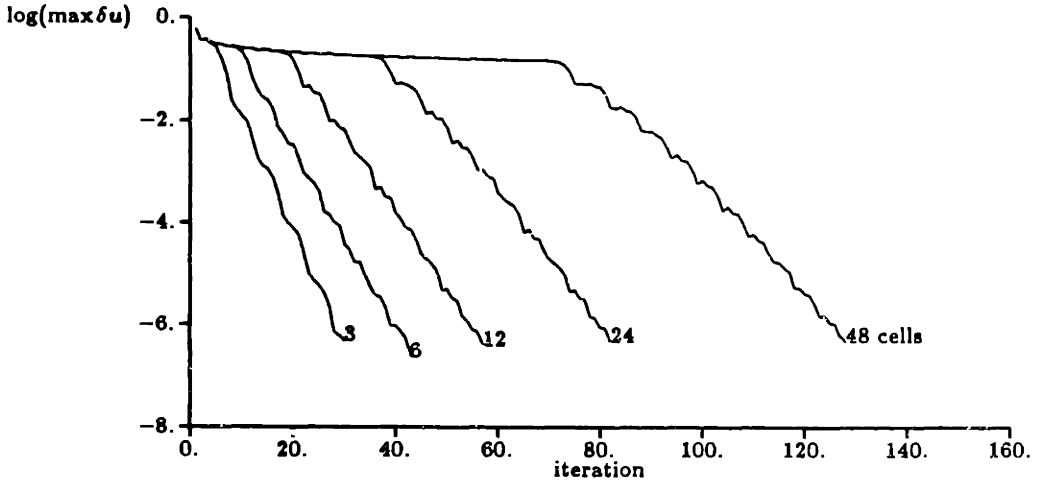


Figure 3.13: Demonstrated convergence histories for $u_t + au_x = 0$, $\lambda = 0.7$

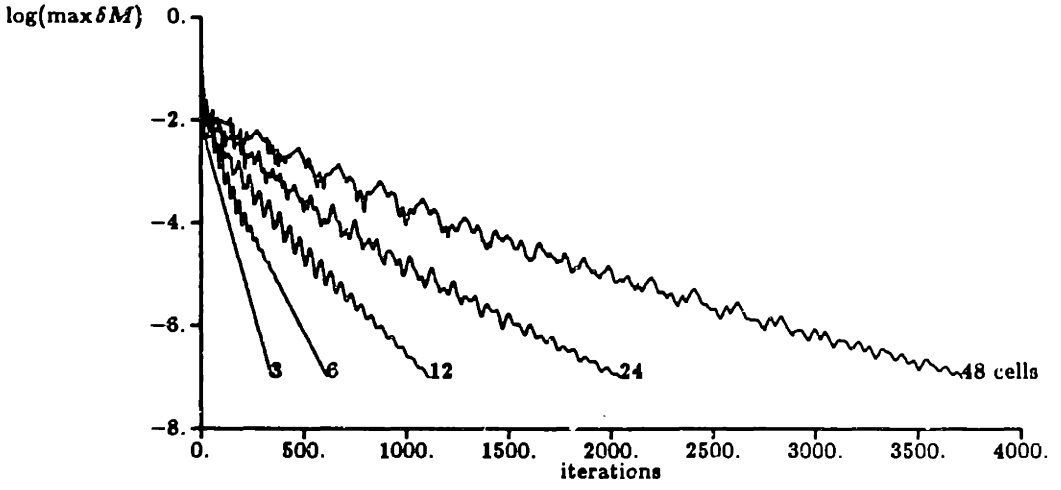


Figure 3.14: Demonstrated convergence histories for 1-D subsonic Euler flow

3.4 Multiple-Grid Accelerator

As seen in the previous section, the computational work required to reach convergence (steady state) is strongly affected by the number of cells in the computational domain. Part of this difference is due to the convergence rate differences (as seen in figures 3.13 and 3.14) which affects the number of iterations needed to achieve convergence. The other part is due to the fact that the number of operations which must be performed per iteration is directly proportional to the number of cells. Hence for purposes of efficiency, one wants to perform the calculation on the coarsest allowable grid; unfortunately the solution accuracy suffers as a result.

To circumvent this problem, Ni[67] presented a multiple-grid technique for hyperbolic equations which is somewhat patterned after the multigrid work of Brandt[14] for elliptic equations. In both techniques, solutions on coincident grids of differing spatial resolutions are coupled, with the finest grid supplying fine spatial accuracy and the coarse grid supplying rapid convergence rates. The exact mechanisms of this coupling in the two approaches are different however. On the one hand for the multigrid approach, the smoothing rates on the different grids are coupled to effectively smooth all frequency components simultaneously. On the other hand for the multiple-grid technique, the "corrections" which are calculated based upon discretization of the governing equations on the finest grid are further propagated by employing the distribution formulae on the coarser grids. The similarities and differences of these two approaches are further discussed in detail by Ni[66].

3.4.1 Basic development

As mentioned above, a series of coincident grids of varying spatial resolution are required. The simplified approach used here for generating these grids begins with the fine grid, hereinafter referred to as the "level 0" grid,

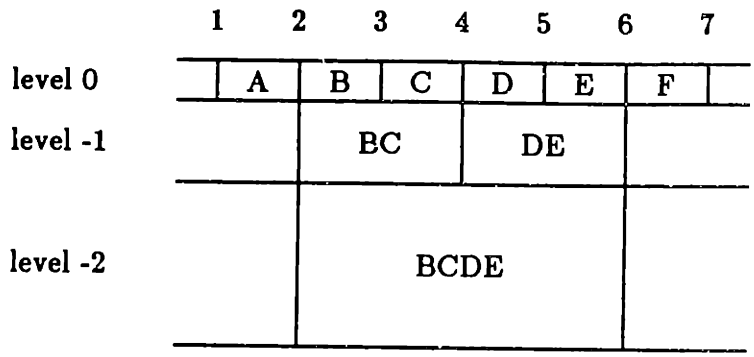


Figure 3.15: Multiple grids in space and pseudo-time

which for simplicity is taken to be a one-dimensional grid. The next coarser grid, called the "level -1" grid, is generated by deleting every other level 0 grid line in each computational direction, essentially creating a grid whose spacing is twice the fine grid spacing. This grid spacing ratio $(2:1)$ was chosen primarily because it simplifies the multiple-grid accelerator (as shown below).

The process is repeated recursively to create still coarser grids, resulting in the grids shown in figure 3.15. Here the level 0 (fine) grid is shown at the top of the figure, with the level -1 grid shown below it and the level -2 grid below that.

As stated above, the basic philosophy of the multiple-grid accelerator is to use the coarse grid to propagate the corrections computed on the fine grid. Thus after a fine grid calculation has been performed as described in the sections 3.1 and 3.2, the multiple-grid accelerator is employed, consisting of three operations: transport, propagate, and interpolate/update. Each of these operations are described fully in the following paragraphs.

Transport the corrections on the fine grid to the underlying coarse cell. The fine grid's corrections are denoted by $\delta \mathbf{U}\|_0$ and the coarse grid's changes by $\Delta \mathbf{U}\|_{-1}$, where in both cases the subscript denotes the grid level. In

operator form, the transport operator can be written as

$$\Delta \mathbf{U}\|_{-1} = \mathcal{T}\|_{-1}^{0}\, \delta \mathbf{U}\|_{0} \tag{3.132}$$

The transport operator $\mathcal{T}\|_{-1}^{0}$ can be implemented in a variety of ways. The simplest is injection, that is direct use of the fine grid corrections at the center of the coarse cell. For example $\mathcal{T}\|_{-1}^{0}$ for the grids of figure 3.15 is

$$\Delta \mathbf{U}_{BC}\|_{-1} = \delta \mathbf{U}_3\|_0 \quad \text{and} \quad \Delta \mathbf{U}_{DE}\|_{-1} = \delta \mathbf{U}_5\|_0 \tag{3.133}$$

Propagate the changes on the coarse grid to the coarse grid nodes (and hence create coarse grid corrections). For this purpose Ni uses the same distribution formulae, which for example take the form

$$(\delta \mathbf{U}_4)_{BC}\|_{-1} = \frac{1}{2}[1+\lambda]\, \Delta \mathbf{U}_{BC}\|_{-1} \tag{3.134a}$$

$$(\delta \mathbf{U}_4)_{DE}\|_{-1} = \frac{1}{2}[1-\lambda]\, \Delta \mathbf{U}_{DE}\|_{-1} \tag{3.134b}$$

Here $(\delta \mathbf{U}_4)_{BC}$ denotes the correction at node 4 from cell BC (a level -1 cell). These distribution formulae are equivalent to the distribution formulae (3.95) derived previously for fine cell calculations. The only difference is that the "change" for a fine cell is computed through a flux balance whereas for coarse cells it is computed by the transport operator.

Interpolate the corrections computed by the distribution formulae above to the intervening nodes on the fine cells. In other words, the corrections at the corners of the coarse grid cells must be interpolated to the face and center nodes. Here, this is accomplished by a linear interpolation in computational space, or

$$(\delta \mathbf{U}_3)\|_{-1} = \frac{1}{2}\left[(\delta \mathbf{U}_2)\|_{-1} + (\delta \mathbf{U}_4)\|_{-1}\right] \tag{3.135}$$

After the interpolation, the corrections are added to the dependent variables $\mathbf{U}$.

Another coarse grid level (level -2) can be employed by first performing the level 0 and level -1 processing as above, then transporting the level -1

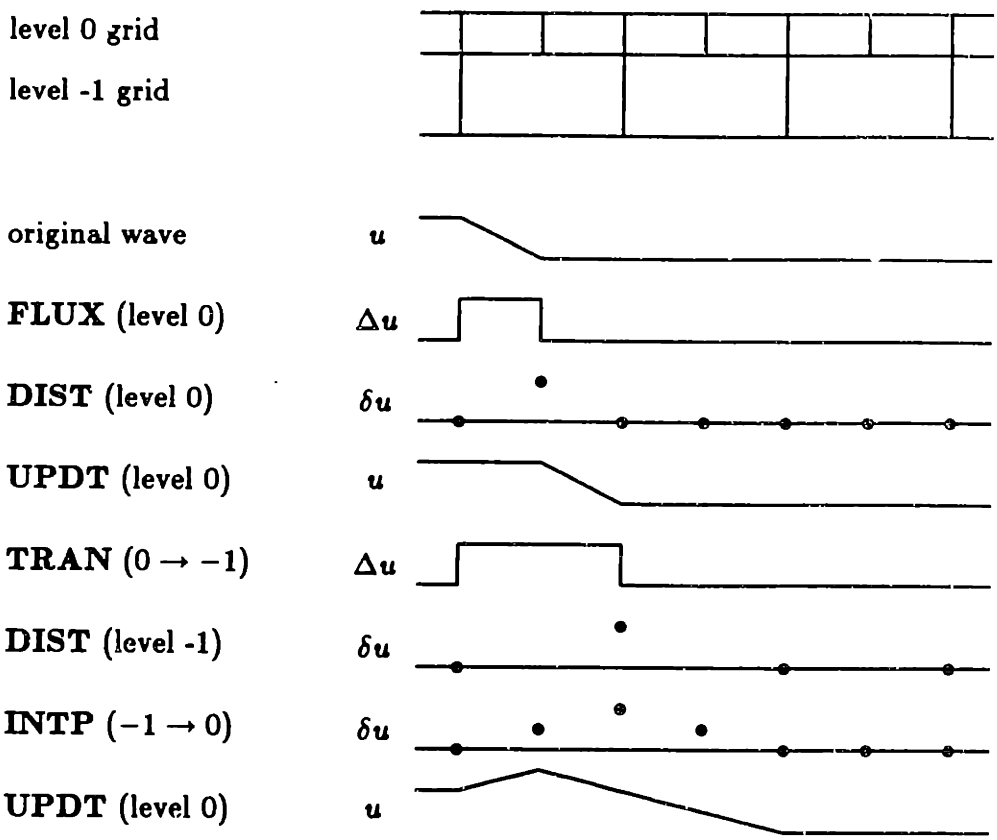


Figure 3.16: Propagation of simple discontinuity by $u_t + au_x = 0$, Case A, $\lambda = 1$, transport by simple injection

corrections to level -2, and finally interpolating the level -2 corrections back to level 0 before updating.

To examine the operation of the multiple-grid accelerator, consider the propagation of a simple wave on the two-level system shown in figure 3.16. (The format of this figure is very similar to figure 3.10). At the top of the figure are the level 0 and level -1 grids.

The first case to be examined, *Case A*, has the initial wave shape shown on the second line of the figure. This is then followed by the change Δu computed by the flux balance equation (3.92), the corrections δu computed

by the distribution formulae (3.95), and the wave shape after the application of the corrections. Up to this point the propagation of this wave is identical to the propagation shown in figure 3.10.

The next operation is the application of the transport operator from level 0 to level -1. Using injection, the change in each coarse grid cell is equal to the correction at the node located at the center of the cell. For the case shown here, only the left-most coarse cell contains a correction at its center and hence the change Δu shown. The distribution formulae (3.95) applied to the coarse grid cells results in the corrections δu shown on the following line. Linear interpolation is then used to define the corrections at the coarse cell centers.

Notice that the wave front has now advanced three cells in the multiple-grid cycle, but that it has been distorted by the interpolation process. This high frequency error due to the interpolation can be eliminated through use of a distribution-like interpolation formula which essentially biases the interpolation in the downstream direction. While for this case that would vastly improve the solution (because the wave shape would not get distorted), in general for integration with a smaller time step parameter ($\lambda < 1$), the error introduced by the linear interpolation is no worse than the errors introduced in the distribution step (for example, see figure 3.11); numerical tests using the Euler equations support this latter result.

Consider now a second case, *Case B*, in which the initial wave is the same shape as in case A, but where it is displaced one cell to the right as shown in figure 3.17. The fine grid flux balance, distribution, and update formulae produce results similar to case A; unfortunately, the multiple-grid accelerator does not produce similar results. This can be traced to the fact that the injection operator fails to generate any coarse grid changes because none of the fine grid corrections are located at the center of a coarse grid cell. Hence the multiple-grid accelerator is ineffective for this case.

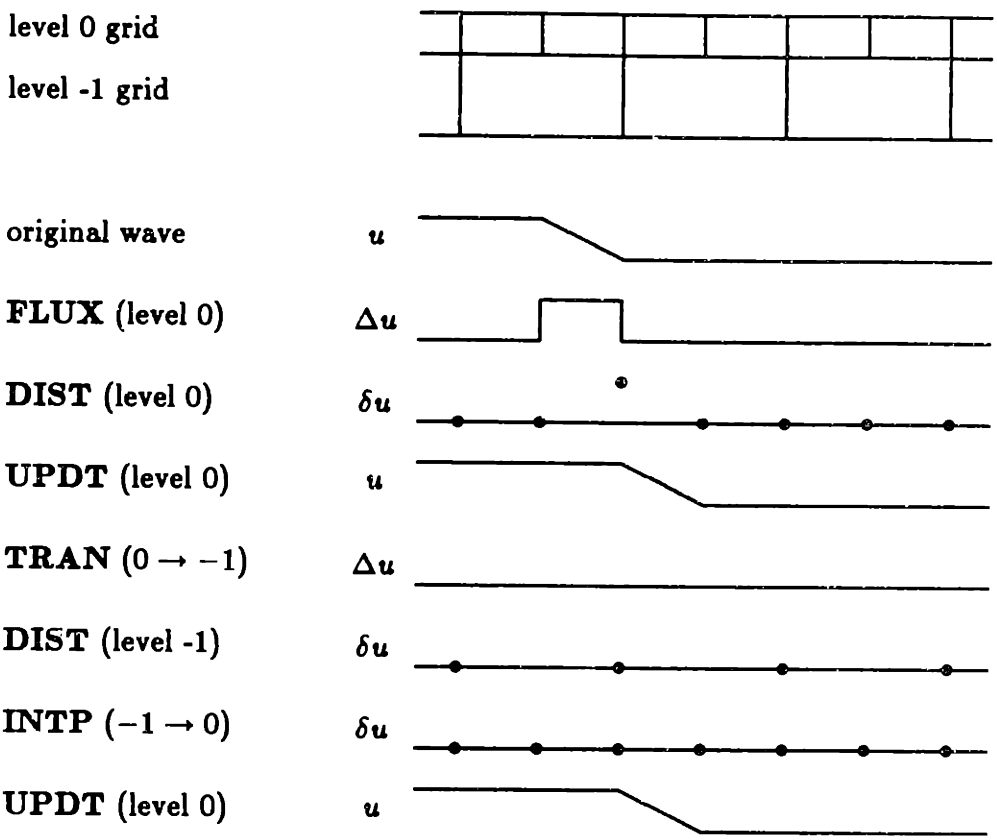


Figure 3.17: Propagation of simple discontinuity by $u_t + au_x = 0$, Case B, $\lambda = 1$, transport by simple injection

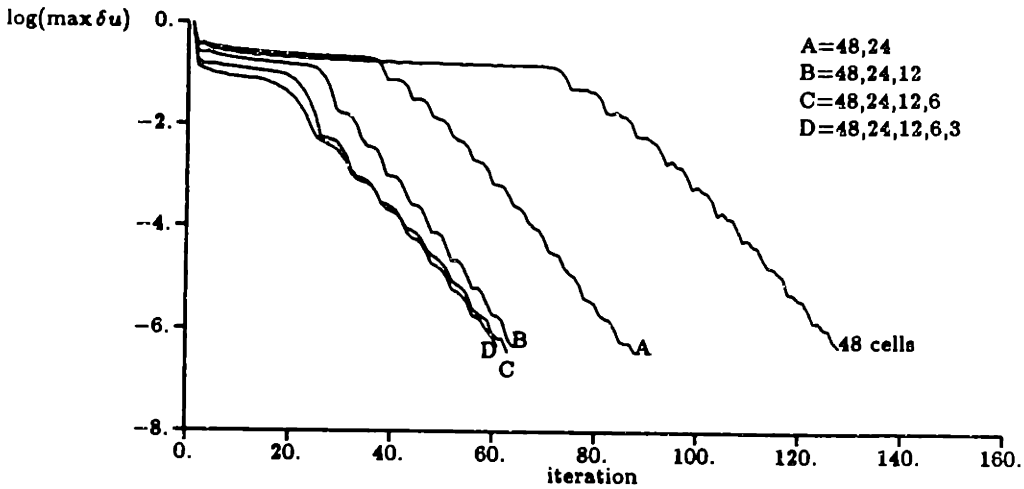


Figure 3.18: Demonstrated convergence histories with multiple-grid for $u_t + au_x = 0$, injection only

The convergence histories for the one-dimensional wave equation in a 48-cell domain is shown in figure 3.18, where the abscissa represents the number of iterations (multiple-grid cycles) and the ordinate the logarithm of the maximum correction of u. The right-most curve is for a computation without a multiple-grid accelerator; successive curves to the left represent increasing levels of the multiple-grid generator, from 1 to 4 coarse levels. Notice that the length of the plateaus decreases with increasing levels of the multiple-grid accelerator, indicating that the propagation of the initial wave is being hastened. Unfortunately the convergence rate once the initial wave has exited the domain (as indicated by the downward slope of the lines) is relatively independent of the number of levels of the multiple-grid accelerator, indicating its relative ineffectiveness in propagating the errors created by both the distribution and interpolation operators out of the domain.

After inspecting the results of both cases A and B, one can notice that the wave location for case A after one multiple-grid cycle is coincident with the case B initial wave location and vice versa. Hence on alternating cycles, the

multiple-grid accelerator as described above will be ineffective, yielding only half of the potential convergence acceleration. To circumvent this problem, one needs to use a transport operator which "sees" fine grid corrections at the coarse grid nodes.

One such method is suggested by Usab[84]. Here a distribution-like formula of the form

$$\Delta \mathbf{U}_{BC}\|_{-1} = \frac{1}{2}\left[1 + \frac{\Delta t}{\Delta x}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\Big|_{BC}\right]\delta \mathbf{U}_2\Big\|_0 + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\Big|_{BC}\right]\delta \mathbf{U}_4\Big\|_0 \tag{3.136}$$

is used, where the first term on the right hand side is the propagation (distribution) of the correction at the upstream node and the second term is the propagation from the downstream node. For clarity, this transport operator will be called *inward distribution.*

Applying the inward-distribution to cases A and B results in the propagations shown in figures 3.19 and 3.20. Notice that here the multiple-grid accelerator is ineffective for case A even though the problem with case B has been eliminated. The convergence histories for various levels of the multiple-grid accelerator, which are shown in figure 3.21, again indicate shortened plateaus for the initial wave but no improvement in the terminal convergence rates.

To correct this problem, a combination of injection and inward distribution is required to capture all fine grid corrections. This results in

$$\begin{aligned}\Delta \mathbf{U}_{BC}\|_{-1} &= \sigma_i\, \delta \mathbf{U}_3\|_0 \\ &+ \sigma_d \left\{\frac{1}{2}\left[1 + \frac{\Delta t}{\Delta x}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\Big|_{BC}\right]\delta \mathbf{U}_2\|_0 + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\Big|_{BC}\right]\delta \mathbf{U}_4\|_0\right\}\end{aligned} \tag{3.137}$$

where σ_i and σ_d are the weights given to injection and inward distribution respectively. A stability proof (given below) indicates that a straightforward sum of the injection and inward distribution operators ($\sigma_i = \sigma_d = 1$) results in a stable scheme. Using these values results in propagation of both

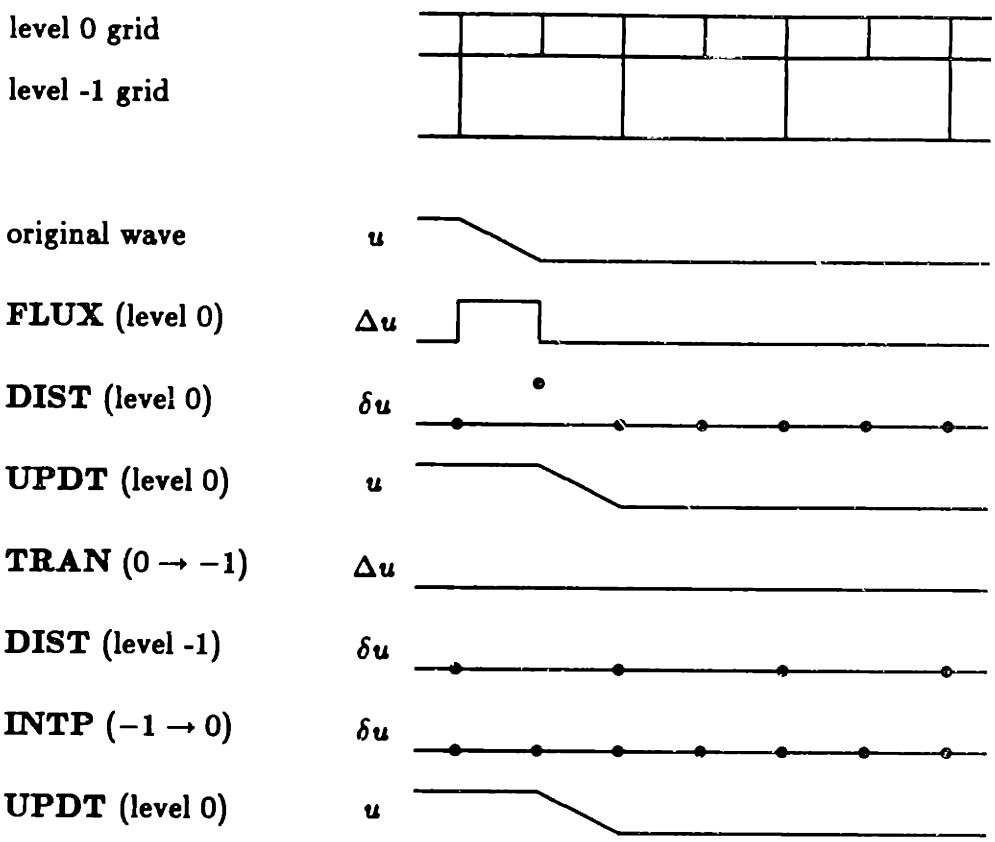


Figure 3.19: Propagation of simple discontinuity by $u_t + au_x = 0$, Case A, $\lambda = 1$, transport by inward distribution

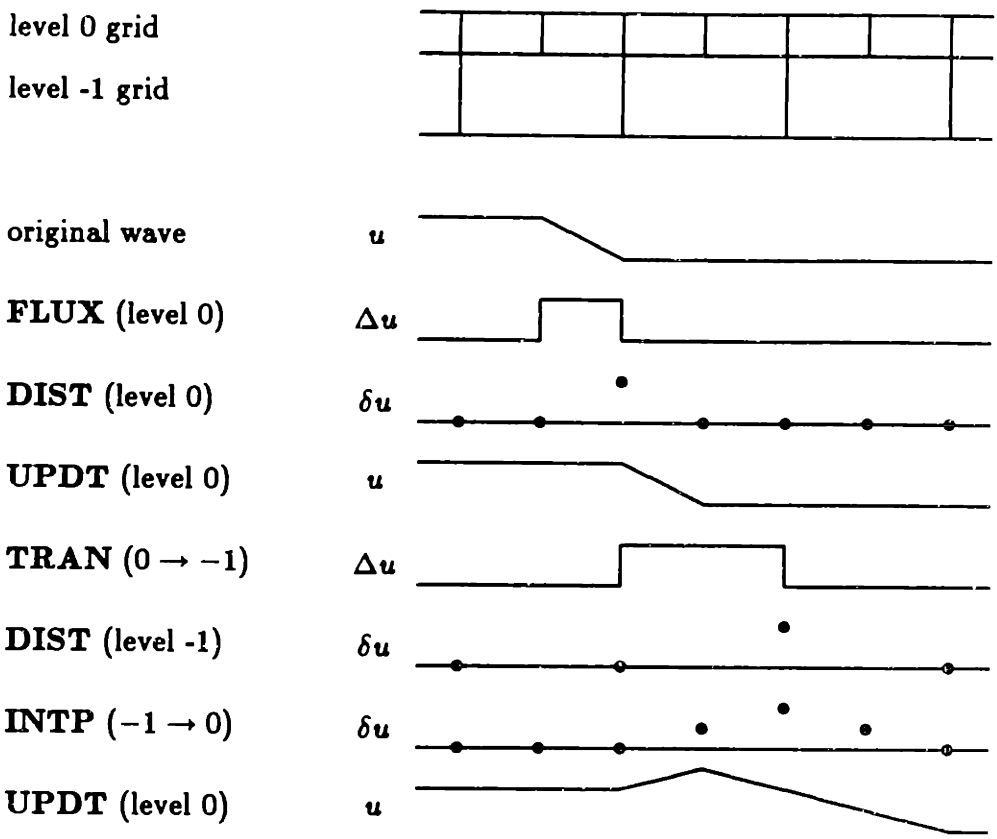


Figure 3.20: Propagation of simple discontinuity by $u_t + au_x = 0$, Case B, $\lambda = 1$, transport by inward distribution

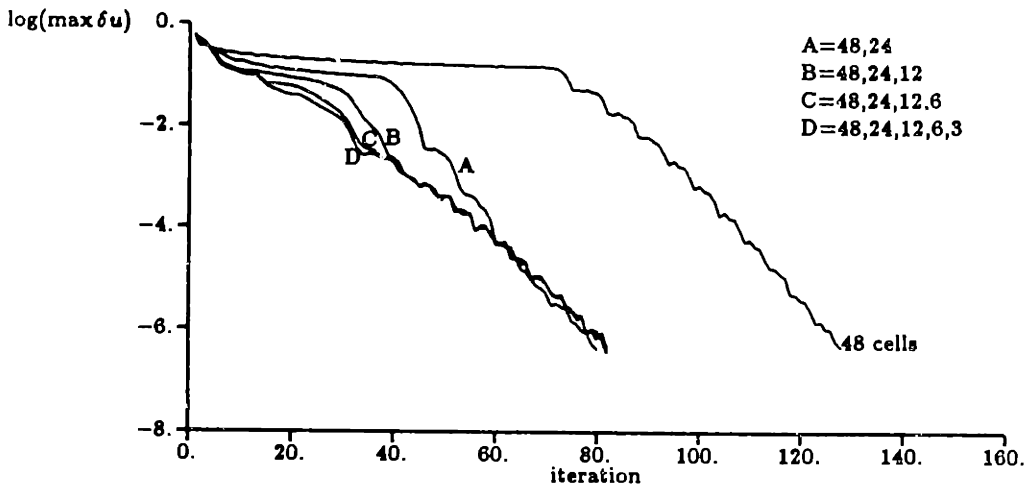


Figure 3.21: Demonstrated convergence histories with multiple-grid for $u_t + au_x = 0$, inward distribution only

waves as shown in figures 3.22 and 3.23. Furthermore, computations on a 48-cell domain with various levels of the multiple-grid accelerator, as shown in figure 3.24, indicate both a shortening of the initial plateau and an increase in the terminal convergence rate as the number of multiple-grid levels increases.

Table 3.4 contains a summary of the number of iterations required to reach convergence for the above cases. Notice that the composite transport operator ($\sigma_i = \sigma_d = 1$) yielded significantly faster convergence rates, especially as the number of level of the multiple-grid accelerator increases. Also, except for the coarsest case, the number of iterations to reach convergence with the multiple-grid accelerator on a 48-cell domain actually was fewer than that required for the convergence on much coarser grids. This is due to the fact that the effective propagation speed with the multiple-grid accelerator is the sum of the propagation rates on each of the coupled grids. Unfortunately some of the full effect of the multiple-grid accelerator is lost due to the errors introduced, such as in the interpolation step.

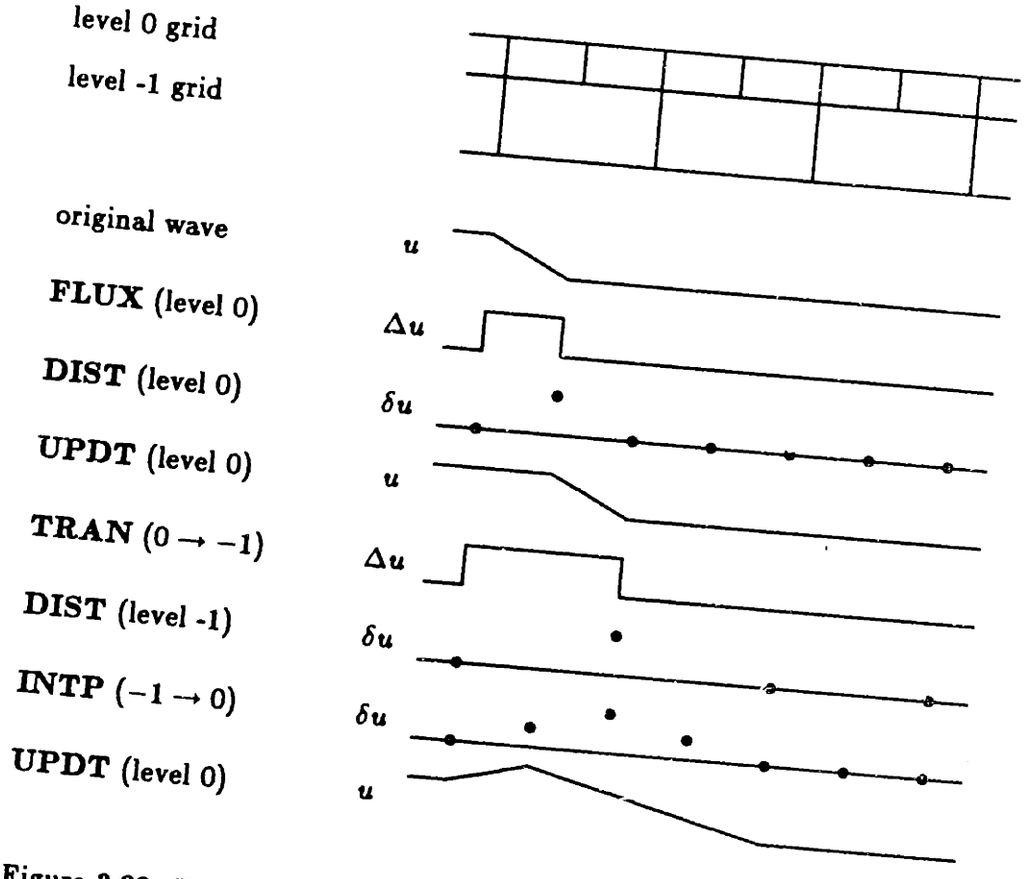


Figure 3.22: Propagation of simple discontinuity by $u_t + au_x = 0$, Case A, $\lambda = 1$, transport by injection and inward distribution

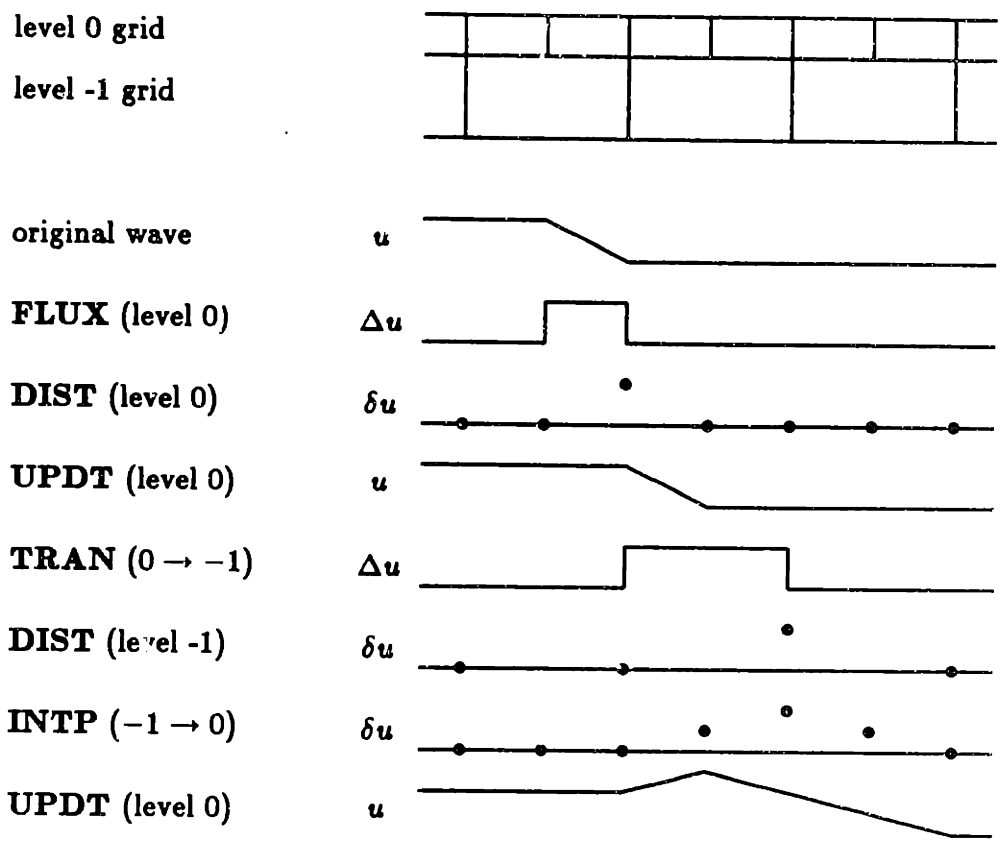


Figure 3.23: Propagation of simple discontinuity by $u_t + au_x = 0$, Case B, $\lambda = 1$, transport by injection and inward distribution

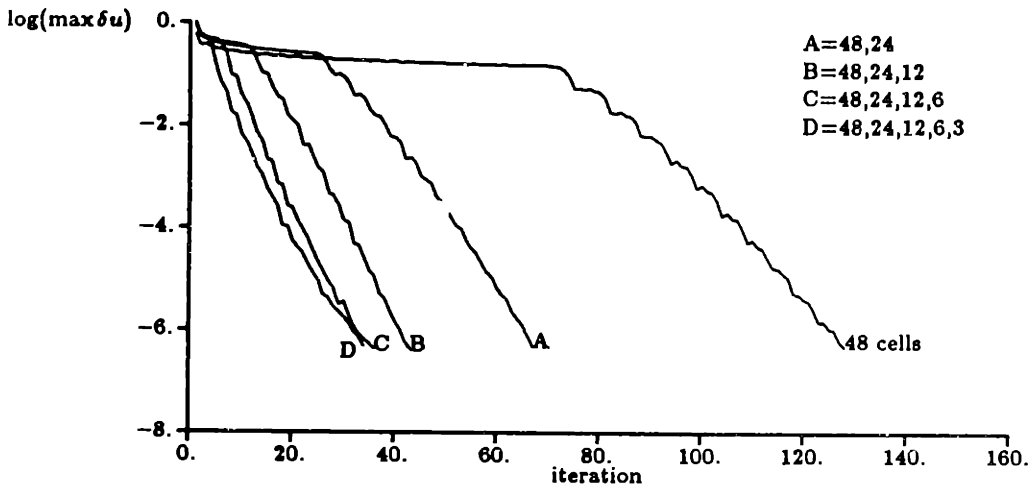


Figure 3.24: Demonstrated convergence histories with multiple-grid for $u_t + au_x = 0$, transport by injection and inward distribution ($\sigma_i = \sigma_d = 1$)

number of cells in coarsest grid	iterations			
	fine grid	$\sigma_i = 1$ $\sigma_d = 0$	$\sigma_i = 0$ $\sigma_d = 1$	$\sigma_i = 1$ $\sigma_d = 1$
3	30	61	82	34
6	43	63	82	36
12	57	64	82	43
24	82	88	80	67
48	128	128	128	128

Table 3.4: Iterations required to convergence for various multiple-grid accelerators

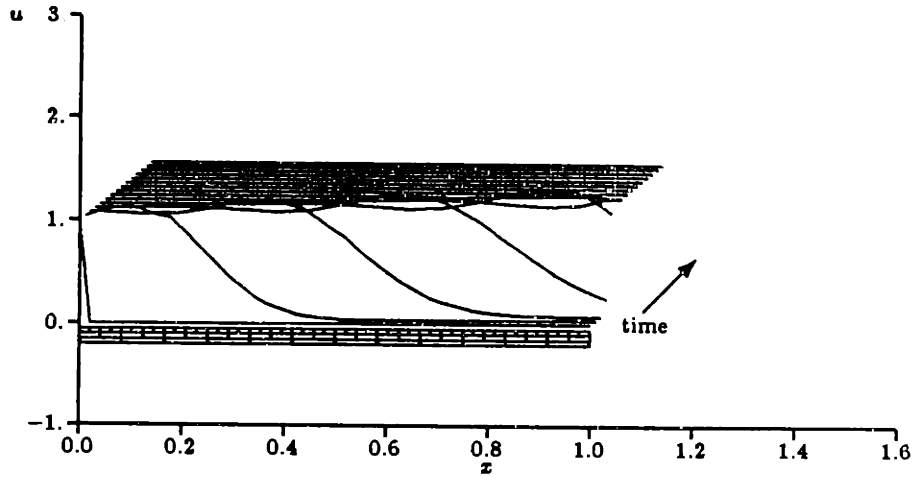


Figure 3.25: Demonstrated wave propagation for $u_t + au_x = 0$, $\lambda = 0.7$, 2 multiple-grid levels

One note of caution is required here. The multiple-grid accelerator hastens convergence to steady state by altering the time evolution of the solution. Figure 3.25 shows the propagation of a simple wave on a 48-cell domain using two levels of the multiple grid accelerator. Notice the differences between this figure and the time evolution shown in figure 3.12 where the intermediate solutions are quite different. This confirms that the multiple-grid accelerator should not be used if time accuracy is required of the solution.

3.4.2 Two dimensions

The operation of the multiple-grid accelerator in two dimensions is a straightforward extension of the one-dimensional scheme discussed above. The grid is generated in a similar manner, yielding coincident grids as shown in figure 3.26. Here the fine grid is shown at the top of the figure with successively coarser grids following below. The nodes which are common to all three grids are shown as filled circles; the dashed lines link the various

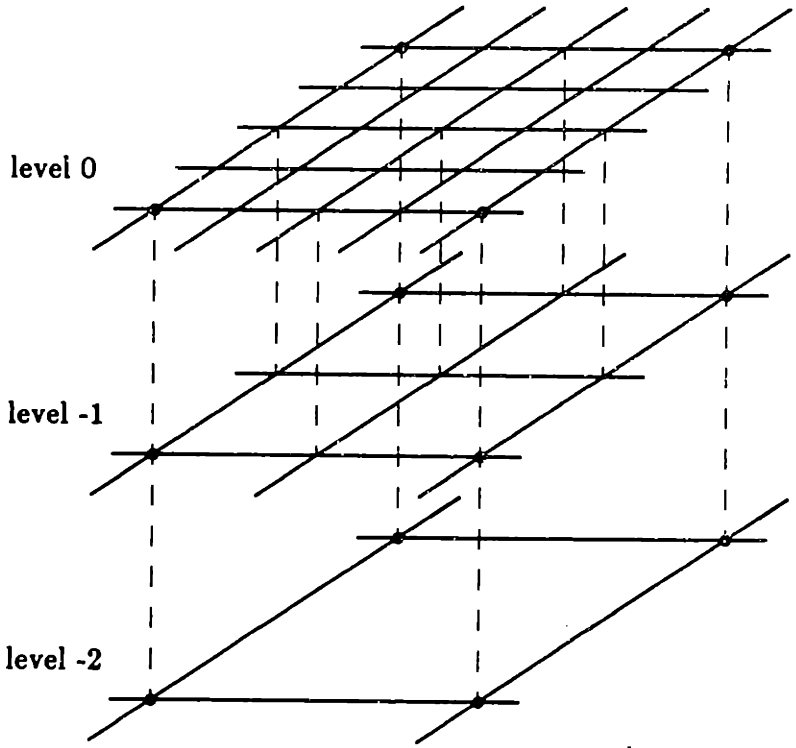


Figure 3.26: Multiple grids, perspective view

images of these nodes. Dashed lines also link the five nodes which are shared only by the level 0 and level -1 grids.

The level 0 (fine) and level -1 (coarse) cells are shown side-by-side in figure 3.27, where the filled circles denote the same nodes in the two pictures. Notice that the coarse grid cell is defined by its four corner nodes (denoted $()_{sw}$, $()_{se}$, $()_{ne}$, and $()_{nw}$), its four face nodes (denoted $()_s$, $()_e$, $()_n$, and $()_w$), and its center node (denoted $()_c$). In addition, the naming conventions for each of the nodes as viewed from each of the fine cells is indicated in the left-hand figure.

The multiple-grid accelerator is inserted into the Ni scheme after a fine grid calculation. Specifically after **FLUX**, **DIST**, **BCON**, and **UPDT**, the following operations are performed on successively coarser grid levels:

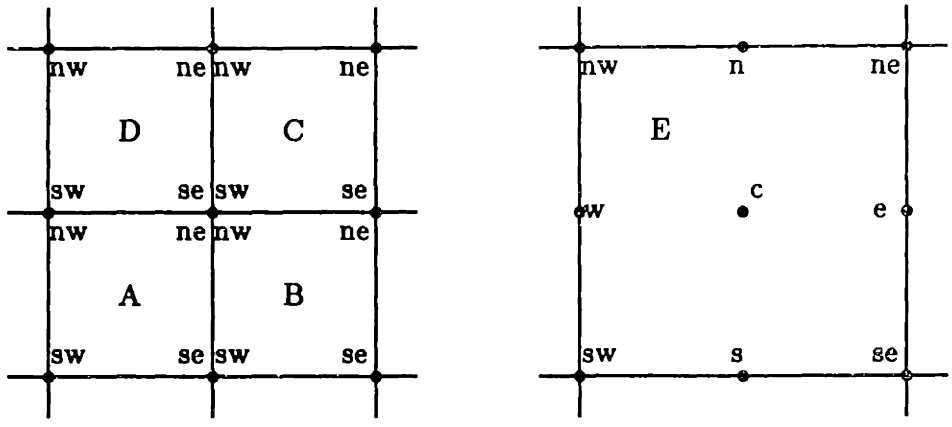


Figure 3.27: Successive multiple grids with nodes and cells labeled

TRAN for all cells on a given level, transport the finer grid corrections to the coarse grid's center node by

$$\begin{aligned}(\Delta\mathbf{U})_c\|_{i-1} = \sigma_i\,(\delta\mathbf{U})_c\|_i &+ \frac{\sigma_d}{4}\left[(\delta\mathbf{U}) + \delta f + \delta g\right]_{sw}\Big\|_i \\ &+ \frac{\sigma_d}{4}\left[(\delta\mathbf{U}) - \delta f + \delta g\right]_{se}\Big\|_i \\ &+ \frac{\sigma_d}{4}\left[(\delta\mathbf{U}) - \delta f - \delta g\right]_{nc}\Big\|_i \\ &+ \frac{\sigma_d}{4}\left[(\delta\mathbf{U}) + \delta f - \delta g\right]_{nw}\Big\|_i\end{aligned} \tag{3.138}$$

where

$$\delta f = \frac{\partial \mathbf{F}}{\partial \mathbf{U}}\delta\mathbf{U}, \qquad \delta g = \frac{\partial \mathbf{G}}{\partial \mathbf{U}}\delta\mathbf{U} \tag{3.139}$$

Here the change on the left side of (3.138) is at the coarse grid level while all the terms on the right are corrections from the fine grid level.

PROP propagate the coarse grid change to the corners of the coarse cells with the distribution formulae

$$(\delta\mathbf{U})_{sw} = \frac{1}{4}\left[\Delta\mathbf{U} - \Delta f - \Delta g\right] \tag{3.140a}$$

$$(\delta \mathbf{U})_{se} = \frac{1}{4}[\Delta \mathbf{U} + \Delta f - \Delta g] \quad (3.140b)$$

$$(\delta \mathbf{U})_{ne} = \frac{1}{4}[\Delta \mathbf{U} + \Delta f + \Delta g] \quad (3.140c)$$

$$(\delta \mathbf{U})_{nw} = \frac{1}{4}[\Delta \mathbf{U} - \Delta f + \Delta g] \quad (3.140d)$$

Here all terms refer to the coarse grid changes and corrections. The total coarse grid correction at each coarse grid node is computed by summing the corrections from each of the adjoining cells. The ratio $\Delta t/A$ is the same for both coarse and fine grid calculations.

BCON boundary conditions (see section 3.2) are applied to every coarse grid boundary node. This step is necessary before the interpolation so that interior nodes near boundaries contain proper corrections.

INTP the total coarse grid corrections are bilinearly interpolated to all intervening coarse grid side and center nodes by

$$(\delta \mathbf{U})_{s} = \frac{1}{2}[(\delta \mathbf{U})_{sw} + (\delta \mathbf{U})_{se}] \quad (3.141a)$$

$$(\delta \mathbf{U})_{e} = \frac{1}{2}[(\delta \mathbf{U})_{se} + (\delta \mathbf{U})_{ne}] \quad (3.141b)$$

$$(\delta \mathbf{U})_{n} = \frac{1}{2}[(\delta \mathbf{U})_{ne} + (\delta \mathbf{U})_{nw}] \quad (3.141c)$$

$$(\delta \mathbf{U})_{w} = \frac{1}{2}[(\delta \mathbf{U})_{nw} + (\delta \mathbf{U})_{ne}] \quad (3.141d)$$

$$(\delta \mathbf{U})_{c} = \frac{1}{4}[+(\delta \mathbf{U})_{sw} + (\delta \mathbf{U})_{se} + (\delta \mathbf{U})_{ne} + (\delta \mathbf{U})_{nw}] \quad (3.141e)$$

This interpolation, which is bilinear in computational space, is applied recursively if on a coarser level than level -1 until the corrections are interpolated to all level 0 nodes.

BCON boundary conditions are again applied to the side nodes of the coarse grid cells which are on boundaries. In practice, for logical simplicity the boundary conditions are applied to all boundary nodes. If recursive interpolation is required (that is there is more than one

multiple-grid level), boundary conditions are applied after each interpolation step.

UPDT the total corrections are then added to the dependent variables at every node.

If a coarser grid level computation is required, it is applied now (exactly as above), using the corrections left by the previous coarse grid calculation. The effectiveness of the multiple-grid accelerator with more than one coarse grid level is discussed in the following section.

3.4.3 Properties

In section 3.3, the order of accuracy, stability, and conservation of the basic scheme were explored. In this section, these properties are determined for the basic scheme in combination with the multiple-grid accelerator.

3.4.3.1 Order-of-accuracy and consistency

The creation of the modified differential equation for the scheme when coupled with the multiple-grid accelerator is conceptually the same as that for the basic scheme alone but is considerably more complicated due to the complexity of the operators in the multiple-grid accelerator. Therefore for simplicity, the following analysis is developed in one-dimension employing the nomenclature in figure 3.15.

As in section 3.3.1, the modified differential equation is created by substituting Taylor series expansions about a common point for the various occurrences of the dependent variable $\mathbf{U}$, the flux vectors $\mathbf{F}$, and the flux Jacobians $\frac{\partial \mathbf{F}}{\partial \mathbf{U}}$. Here all the expansions are taken about node 4 at the unstarred time level. The only new complication here is in choosing Δt_{mg} for the expansion of $\mathbf{U}$ in time. For the present analysis, $\Delta t_{\text{mg}} = \alpha \Delta t_{\text{fine}}$ is used.

Substituting the Taylor-series expansions into the fine and coarse grid operators yields the modified differential equation

$$\alpha \mathbf{U}_t + (1+\sigma_i+\sigma_d)\mathbf{F}_x$$
$$= \Delta t \left[\frac{-18\mathbf{U}_{tt} + (2+3\sigma_d+6\sigma_i)\left.\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right|_x \mathbf{F}_x + (2+10\sigma_d+6\sigma_i)\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\mathbf{F}_{xx}}{4} \right]$$
$$+ (\Delta x)^2 \left[\frac{-1-4\sigma_i-7\sigma_d}{6}\mathbf{F}_{xxx} \right] + O\left[(\Delta t)^2\right] \qquad (3.142)$$

To make this consistent with the differential equation requires that $\alpha = (1+\sigma_i+\sigma_d)$. For $\sigma_i = \sigma_d = 1$, (3.142) becomes

$$\mathbf{U}_t + \mathbf{F}_x = \Delta t \left[\frac{25}{12} \left.\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right|_x \mathbf{F}_x \right] + (\Delta x)^2 \left[-2\mathbf{F}_{xxx}\right] + O\left[(\Delta t)^2\right] \qquad (3.143)$$

Hence Ni's scheme with the multiple-grid accelerator is consistent with first-order temporal accuracy and second-order spatial accuracy. The fact that the scheme is only first order accurate in time is unimportant since the purpose of the multiple-grid accelerator is to hasten convergence to a steady state.

The total time step for one multiple-grid cycle is the sum of the fine and coarse grid steps, or

$$\Delta t_{\text{mg}} = \Delta t_{\text{fine}} + \Delta t_{\text{coarse}} \qquad (3.144)$$

from which one obtains $\Delta t_{\text{coarse}} = 2\Delta t_{\text{fine}}$. This is consistent with the assumption that the time step parameter λ is the same on the coarse and fine grid levels.

To demonstrate that the multiple-grid accelerator does not affect the steady-state accuracy, the model problem shown in figure 3.7 was repeated using the multiple-grid accelerator. Here computations are performed with a common 97×33 fine grid but with various levels of multiple-grid. These new results, which are plotted in figure 3.28, are superimposed on the data of figure 3.7c. For the cases with the multiple-grid accelerator, the abscissa represents the coarsest grid size; in the other cases it simply represents the

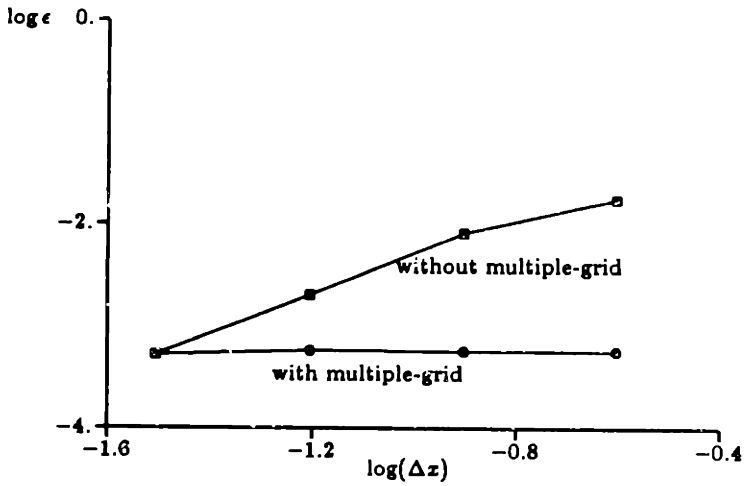


Figure 3.28: Demonstrated accuracy versus grid resolution for the subsonic test case

grid size. In both cases, the ordinate represents the L_2-norm of the total pressure error. As can be seen, the accuracy of the cases using the multiple-grid accelerator is the same as the fine grid case without the multiple-grid accelerator.

3.4.3.2 Stability

Stability analyses of integration schemes using the von Neumann (Fourier) techniques require that a single uniform grid be used in the integration. The complex interactions of the operations on the coupled grids in the multiple-grid accelerator thus makes Fourier stability analyses untenable. Fortunately the energy stability method allows for this type of coupling and hence is the approach taken here.

As in section 3.3.2, the stability analysis begins by writing the full scheme in operator notation. Recall that the fine grid procedure is given by (3.110), or

$$u_i^* = u_i + D_i \tag{3.145}$$

all nodes, that is ignoring the stabilizing effect of the interpolation. This assumption is necessary in order to employ identities I-7 and I-8 in table 3.3.

After much algebra, the final simplified result is

$$\|u^{**}\|^2 - \|u\|^2 = +A_1\|\delta_x u\|^2 + A_2\|\delta_{2x} u\|^2 + A_3\|\delta_{3x} u\|^2 + A_4\|\delta_{4x} u\|^2 + A_5\|\delta_{5x} u\|^2 + A_6\|\delta_{6x} u\|^2 \quad (3.152)$$

where $A_j (j = 1, \ldots, 6)$ are complicated functions of σ_i, σ_d, λ, and λ^*.

For stability, it is required that $\|u^{**}\|^2 - \|u\|^2 \leq 0$. This can only be strictly satisfied if $A_1 \leq 0$, $A_2 \leq 0$, ..., $A_6 \leq 0$. These inequalities in turn require that $\sigma_i = \sigma_d = 0$ and $\lambda \leq 1$ which is a trivial solution.

For a non-trivial result, some method is required for relating the magnitudes of the various norms which appear in (3.152). The technique used here is to assume that u can be represented by the Fourier series

$$u = \sum_{n=0}^{I} a_n \sin \frac{n\pi i}{I} \qquad 0 \leq i \leq I \quad (3.153)$$

where I is the number of nodes in the domain. For each mode n, the norms of the various differences of u are related by

$$\frac{\|\delta_x^m u\|^2}{\|\delta_x u\|^2} = \left[4 \sin^2 \frac{n\pi}{I}\right]^{m-1} \quad (3.154)$$

Substituting (3.154) into (3.152) yields

$$\|u^{**}\|^2 - \|u\|^2 = \left\{\sum_{k=1}^{6} A_k \left[4 \sin^2 \frac{n\pi}{I}\right]^{k-1}\right\} \|\delta_x u\|^2 \quad (3.155)$$

The scheme is thus stable if the term in braces is less than or equal to 0 for all permissible values of n. This region of stable operation has been determined by evaluating (3.155) numerically for various values of σ_i, σ_d, and λ; it is assumed here that $\lambda^* = \lambda$.

The *predicted* stability region is plotted in figures 3.29a-b for $\lambda = 0.75$ and 0.95. In each plot the abscissa represents the inward-distribution coefficient σ_d and the ordinate represents the injection coefficient σ_i. The

predicted region of stable operation is indicated by the +. An important point to note is that $\sigma_i = \sigma_d = 1$ is stable in both cases.

In addition, these figures contain the *demonstrated* stability bounds found by integrating the simple wave equation (3.89) on a 48-cell domain with a random initial distribution of u. The points of stable integration are shown as $\times$ symbols in the figures. In both cases, the demonstrated stability bound is much larger than the predicted stability bound. But since the energy stability method only provides a necessary stability bound (due to a possibly less than optimal choice of an energy norm), this discrepancy should not be alarming.

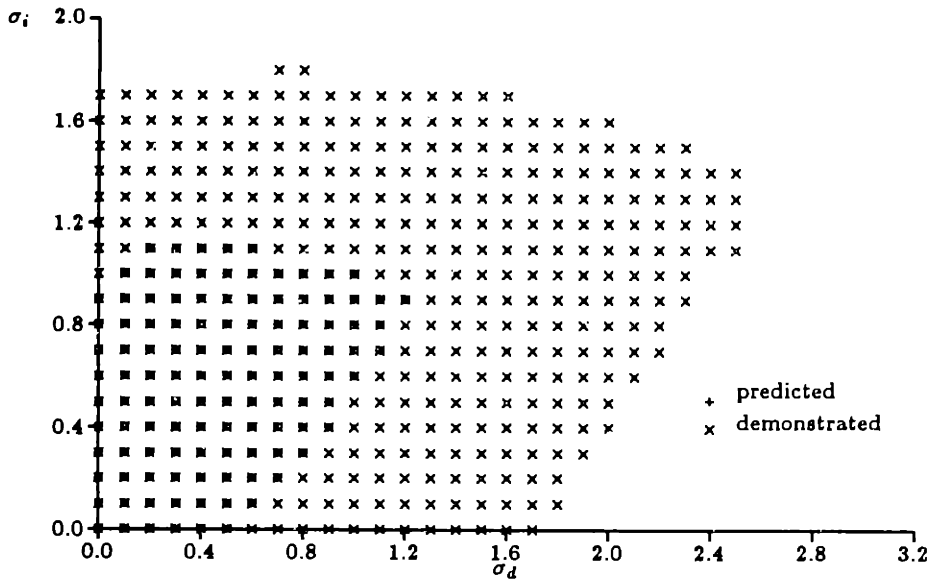
3.4.3.3 Conservation

The multiple-grid accelerator is used to hasten the propagation of the corrections computed on the fine grid. As long as no net changes are created by the multiple-grid accelerator, one should expect the conservation property of the combined scheme to be the same as that of the fine-grid-alone solution.

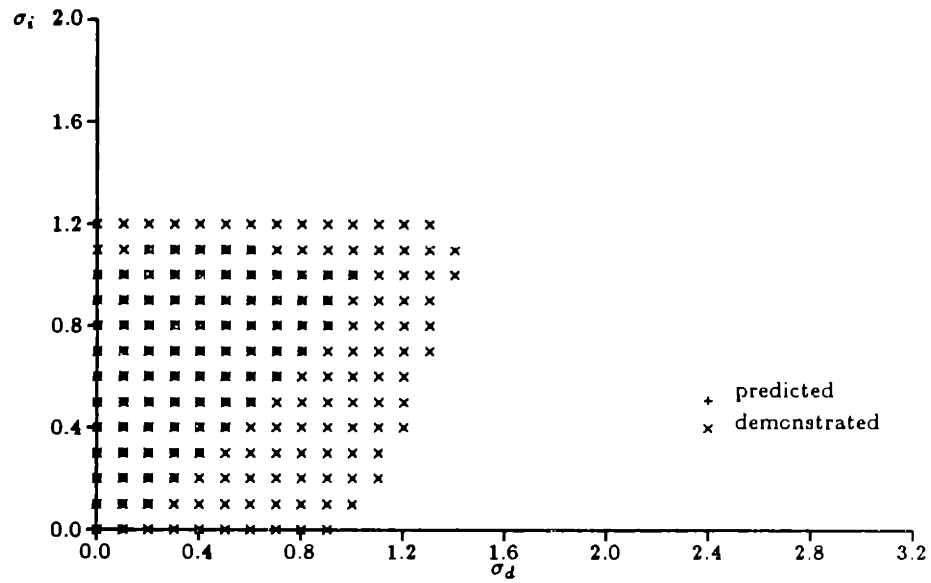
This has been demonstrated by recomputing the solution of figure 3.9, but this time employing two multiple-grid levels. The resulting flux integrals are plotted in figure 3.30. As before, the flux integrals are nearly constant as a function of axial position, indicating the scheme's conservative character.

3.5 Smoothing

As with most numerical procedures for time-marching hyperbolic equations to steady state, the Ni scheme requires the explicit addition of an artificial dissipation (or smoothing). It is required both to damp out spurious oscillatory solutions in smooth regions of the flow and to capture discontinuities such as shocks[70]. For future reference, these requirements will be



a: $\lambda = 0.75$



b: $\lambda = 0.95$

Figure 3.29: Predicted and demonstrated stability bounds for $u_t + au_x = 0$ using one level of multiple-grid

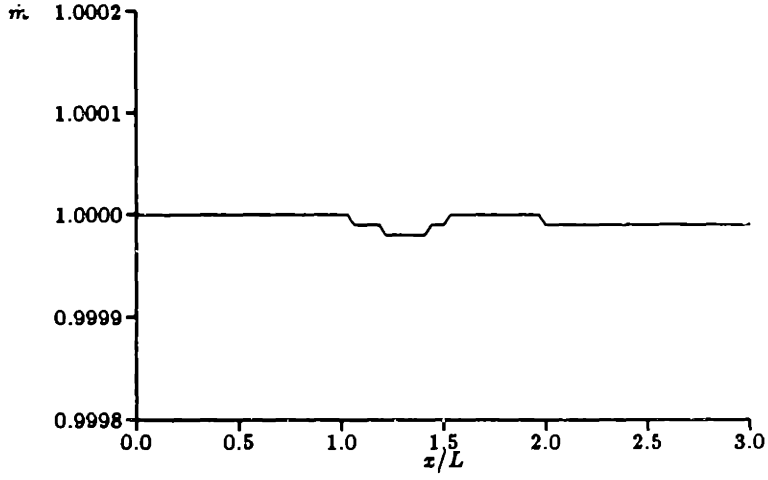


Figure 3.30: Mass flux integrals versus axial position for the subsonic channel test case computed with two multiple-grid levels

called ***background*** and ***shock*** smoothing respectively.

In early integration schemes[67], these distinct requirements for smoothing were treated uniformly by adding a single second-difference smoothing operator of the form

$$\text{smooth}_i = \mu(u_{i-1} - 2u_i + u_{i+1}) \tag{3.156}$$

in one dimension. Unfortunately due to the high smoothing levels required in the vicinity of discontinuities, this resulted in excessive errors in the relatively smooth portions of the flow field.

The effect of the magnitude of the smoothing coefficient on a computed solution is shown in figures 3.31 and 3.32. In these cases, the transonic solutions over a 10 percent circular-arc bump in a channel is computed with a free-stream flow from left to right with a Mach number = 0.675. Different constant smoothing coefficients (using the smoothing operator described below) are used in these two computations. On the one hand, the low level of smoothing in figure 3.31 results in an oscillatory shock as seen in the surface

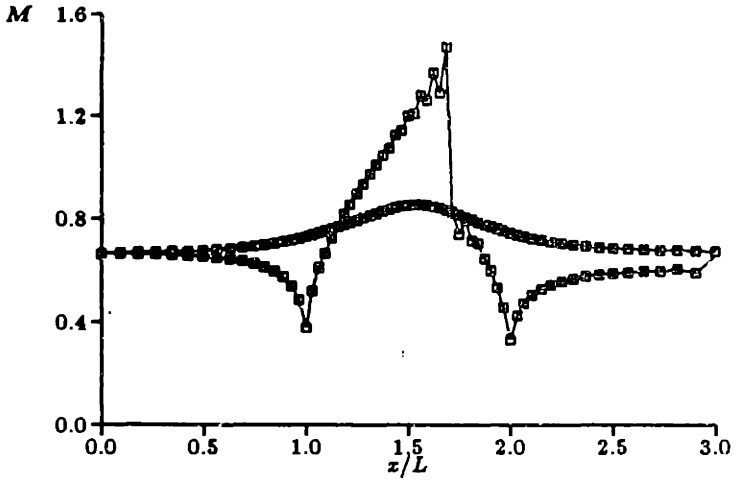
Mach number distribution. On the other hand, the high level of smoothing in figure 3.32 which yields a non-oscillatory shock creates larger spurious total pressure losses.

One recent method for circumventing this problem was developed by Schmidt *et al.*[76] in which different smoothing operators are applied to background and shock regions. In smooth regions, where the spurious numerical oscillations need to be damped, a fourth-difference operator of the form (in one dimension)

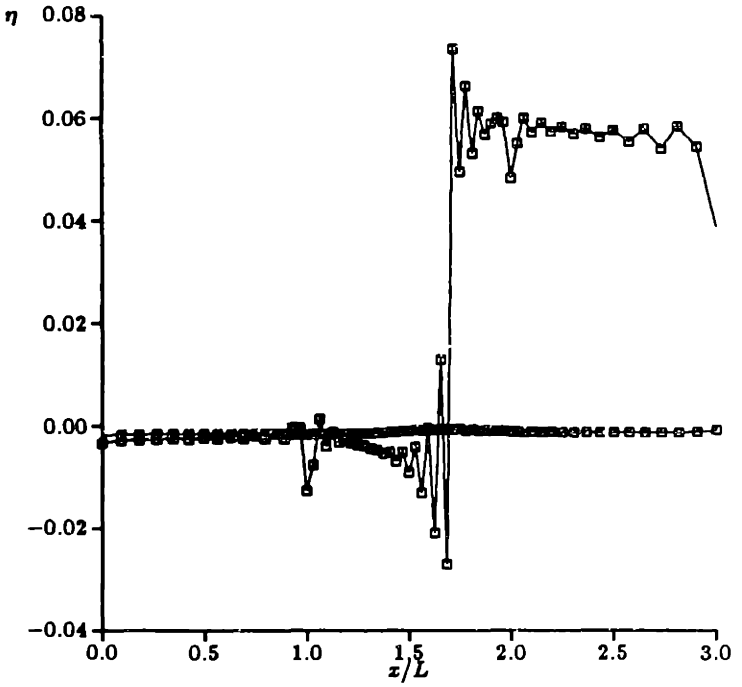
$$\text{smooth}_i = \mu_4(u_{i-2} - 4u_{i-1} + 6u_i - 4u_{i+1} + u_{i+2}) \tag{3.157}$$

is used. Near shocks the fourth-difference is ineffective and destablizing and hence the second-difference form given in (3.156) is used. The two smoothing operators are smoothly blended through appropriate distributions (in space) of μ and μ_4. According to Jameson, the main advantage of this approach is that the first-order spatial accuracy associated with the second-difference smoothing is confined to the vicinity of discontinuities. The major disadvantage however is that this scheme involves both the computation of the spatially varying μ as well as fourth-differences which can be computed either by employing the five-node stencil (in one dimension) as in (3.157) or in two passes with a three-point stencil by computing the second difference of a second difference. In either case, the fourth-difference operator is difficult to compute accurately, especially for unstructured grids and near boundaries.

An alternative approach (and the one used here) is to use only a second-difference smoothing operator globally, but to spatially adjust (adapt) the smoothing coefficient μ such that only the necessary amount of smoothing is used in the background and at discontinuities. In this way, the contamination of the solution in smooth regions of the flow can be minimized. Even though formally this technique is only first-order spatially accurate, its requirement for only a three-point stencil makes it particularly attractive for

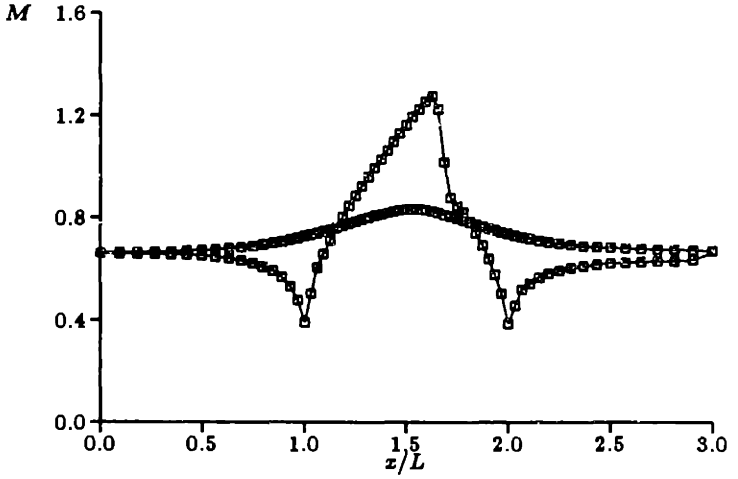


a: surface Mach number distributions

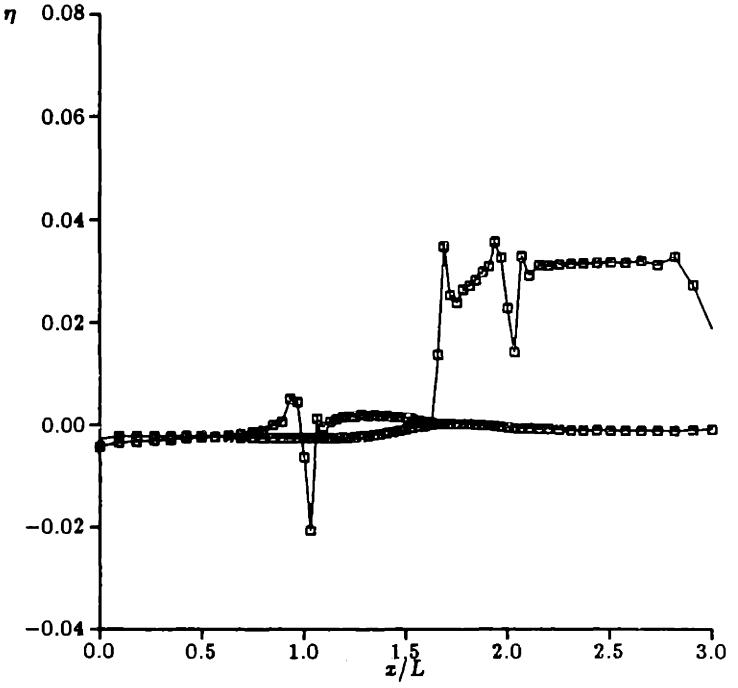


b: surface total pressure loss distributions

Figure 3.31: Surface Mach number and total pressure loss for transonic channel flow test case. smooth = 0.010



a: surface Mach number distributions



b: surface total pressure loss distributions

Figure 3.32: Surface Mach number and total pressure loss for transonic channel flow test case. smooth = 0.050

the problem at hand. A discussion of its accuracy is deferred to the end of this section.

The smoothing problem can now be separated into two tasks: formulation of the smoothing operator given a spatially varying smoothing coefficient and the development of an appropriate distribution of that coefficient. In the following sections these tasks are each first developed in one dimension followed by their two-dimensional extensions.

3.5.1 Smoothing operator

The most obvious way of implementing the smoothing operator is to simply replace the single coefficient in (3.156) by a spatially varying coefficient, or

$$\text{smooth}_i = \mu_i(u_{i-1} - 2u_i + u_{i+1}) \tag{3.158}$$

This operator is purely dissipative at each node i.

A desirable characteristic of any smoothing operator is that it not adversely impact the integration scheme's accuracy in either a differential or integral sense. As discussed in section 3.3.3, the integral accuracy of a scheme is measured by its conservation, that is the degree to which the sum of the smoothing terms over the whole domain vanish. For (3.158), this sum yields

$$\begin{aligned}\sum_{i=2}^{I-1} \text{smooth}_i = \mu_2 u_1 + (-2\mu_2 + \mu_3)u_3 + \sum_{i=3}^{I-2}(\mu_{i-1} - 2\mu_i + \mu_{i+1})u_i \\ +(\mu_{I-2} - 2\mu_{I-1})u_{I-1} + \mu_{I-1}u_I\end{aligned} \tag{3.159}$$

Notice that the contribution of the smoothing terms in the interior of the domain does not vanish for non-constant μ_i, and hence (3.158) represents a non-conservative operator. This non-conservation can be traced to the fact that the term $(u_i - u_{i-1})$ is multiplied by μ_{i-1} when added to the smoothing at node $i-1$ but is multiplied by μ_i when subtracted at node i.

An alternative smoothing operator which circumvents the above problem is given by

$$\text{smooth}_i = f(\mu_{i-1},\mu_i)(u_{i-1}-u_i) - f(\mu_i,\mu_{i+1})(u_i - u_{i+1}) \qquad (3.160)$$

in which the differences of the first differences are weighted based upon the function f. The only restriction on f is that $f(a,b) = f(b,a)$. For this smoothing operator, the sum of the smoothing over the domain is

$$\sum_{i=2}^{I-1}\text{smooth}_i = f(\mu_1,\mu_2)(u_1-u_2) - f(\mu_{I-1},\mu_I)(u_{I-1}-u_I) \qquad (3.161)$$

By design, the contributions of the interior nodes vanish, making this operator conservative. Unfortunately by weighting the first differences unequally, the operator at node i is equivalent to

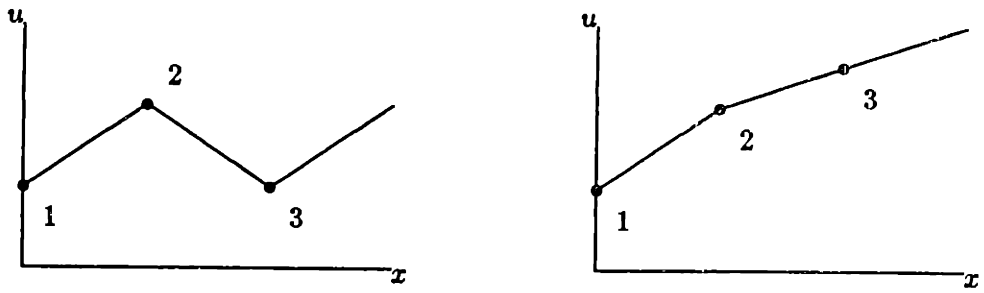
$$\begin{aligned}\text{smooth}_i = &\frac{f(\mu_i,\mu_{i+1})+f(\mu_{i-1},\mu_i)}{2}[u_{i+1}-2u_i+u_{i-1}] \\ &+\frac{f(\mu_i,\mu_{i+1})-f(\mu_{i-1},\mu_i)}{2}[u_{i+1}-u_{i-1}]\end{aligned} \qquad (3.162)$$

so that if $f(\mu_i,\mu_{i+1}) \neq f(\mu_{i-1},\mu_i)$, there is a convective part to the smoothing as given by the second term.

The choice of smoothing operator for non-constant smoothing is thus between (3.159) which is purely dissipative but non-conservative and (3.161) which is conservative but partly convective. For the work here, the latter is chosen on the grounds that the conservative property is important for global accuracy and that for small and reasonably smooth μ_i, the coefficient of the spurious convective term is much smaller than the coefficient of the diffusive term, or

$$\frac{\text{diffusion}}{\text{convection}} = \frac{\Delta x}{2}\left[\frac{f(\mu_i,\mu_{i+1})+f(\mu_{i-1},\mu_i)}{f(\mu_i,\mu_{i+1})-f(\mu_{i-1},\mu_i)}\right]\frac{u_{xx}}{u_x} \qquad (3.163)$$

is large relative to unity. Note that for finite Δx the term in brackets dominates. But for $\Delta x \to 0$, the convection term becomes increasingly



a: oscillatory solution b: smooth solution

Figure 3.33: Possible solution behaviors near boundary node 1

important; hence if very fine grids are to be used, (3.159) may be a better choice.

In addition to the construction of a smoothing operator for interior nodes, one must also construct an appropriate operator at boundary nodes where the choices are even less satisfying.

One choice is to apply the second difference operator

$$\text{smooth}_1 = \mu_1(u_1 - 2u_2 + u_3) \tag{3.164}$$

for node 1, the boundary node. As expected, this boundary operator, when coupled with the interior operator (3.160) is non-conservative. However an even worse characteristic is that it has the wrong behavior for oscillatory solutions. Consider the oscillatory sclution shown in figure 3.33. The operator in (3.164) tends to decrease the value of u_1 for this case instead of increasing it as desired; obviously (3.164) should not be used.

A second boundary smoothing operator can be derived from the point of view of conservation, yielding

$$\text{smooth}_1 = f(\mu_1, \mu_2)(u_2 - u_1) \tag{3.165}$$

Unfortunately this is completely convective (that is has no dissipative component). Additionally if this operator is applied to the smooth solution in

figure 3.33b, its net effect would be to increase u_1 to the level of u_2 which is clearly unacceptable.

The final alternative (and the one used here) is simply not to smooth normal to boundaries. The basis for this choice is the observation that there is no single operator which has the correct behavior for both cases shown in figure 3.33. Hence it is better not to smooth at all (and accept a small amount of non-conservation) than it is to add an incorrect operator. Numerical experiments in which no smoothing is applied normal to boundaries support this alternative.

In two dimensions, the interior and boundary smoothing formulae are straightforward extensions of respective one-dimensional formulae given by (3.160). The resulting two-dimensional formulae, which are applied independently to each of the dependent variables, are applied in the Ni scheme immediately after the **DIST** step.

SMTH for each cell, modify the correction by (for example)

$$\begin{aligned}\delta\mathbf{U}_{sw} = \delta\mathbf{U}_{sw} &+ \frac{1}{8}f(\mu_{sw},\mu_{se})(\mathbf{U}_{se}-\mathbf{U}_{sw}) \\ &+ \frac{1}{8}f(\mu_{sw},\mu_{ne})(\mathbf{U}_{ne}-\mathbf{U}_{sw}) \\ &+ \frac{1}{8}f(\mu_{sw},\mu_{nw})(\mathbf{U}_{nw}-\mathbf{U}_{sw})\end{aligned} \tag{3.166}$$

where for example

$$f(\mu_{sw},\mu_{se}) = \begin{cases} \max(\mu_{sw},\mu_{se}) & \text{sw not on boundary} \\ 0 & \begin{cases}\text{sw on boundary} \\ \text{se not on boundary}\end{cases} \\ 4\max(\mu_{sw},\mu_{se}) & \text{sw and se on boundary}\end{cases} \tag{3.167}$$

Cyclic permutations of (3.166) and (3.167) are used to compute the smoothing contribution at the other nodes. The first condition on the right hand side of (3.167) applies to all interior nodes. Essentially this

operator is the 9-point approximation to the undivided Laplacian of U. The maximum function is used to effectively smooth f and hence limit the magnitude of the spurious convective term. The second condition is used to ensure that there is no smoothing contribution normal to a boundary. The final condition applies along a boundary. The form of this term is similar to the interior operator; the factor 4 is needed to scale this one-dimensional second difference the same as the undivided Laplacian.

The differential accuracy and consistency of the Ni scheme combined with the above smoothing can be determined by creating the modified differential equation as in section 3.3.1. For the case with the smoothing coefficients as given in figure 3.34, this results in

$$\begin{aligned}\mathbf{U}_t + \mathbf{F}_x + \mathbf{G}_y = &\frac{\Delta x}{\Delta t}\left[-\frac{\mu}{8}\epsilon \mathbf{U}_x\right] + \frac{(\Delta x)^2}{\Delta t}\left[\frac{\mu}{8}\left(1+\frac{\epsilon}{2}\right)\mathbf{U}_{xx}\right] \\ &+ \frac{(\Delta y)^2}{\Delta t}\left[\frac{\mu}{8}\left(1+\frac{\epsilon}{4}\right)\mathbf{U}_{yy}\right] + (\Delta x)^2\left[\frac{\mathbf{U}_{txx}}{6} - \frac{\mathbf{G}_{xxy}}{12}\right] \\ &+ (\Delta y)^2\left[\frac{\mathbf{U}_{tyy}}{6} - \frac{\mathbf{F}_{xyy}}{12}\right] + (\Delta t)^2\left[-\frac{\mathbf{U}_{ttt}}{6}\right] + \cdots \end{aligned} \tag{3.168}$$

The first point to notice from (3.168) is that the magnitude of the spurious convective term can be compared to the real convective term, yielding

$$\frac{\text{spurious convection}}{\text{real convection}} = \frac{\Delta x}{\Delta t}\frac{\mu\epsilon}{8}\frac{\mathbf{U}_x}{\mathbf{F}_x} \tag{3.169}$$

For the computed solutions shown in this work, $\mu = \mathcal{O}(0.03)$ and thus if $\mathbf{U}_x \approx \mathbf{F}_x$, then the spurious convection is less than one percent of the actual convection, even for $\epsilon = \mathcal{O}(1)$ which is unlikely.

The second point to notice from (3.168) is that the formal accuracy of the scheme with the smoothing is zeroth-order (and is thus inconsistent) for spatially varying smoothing coefficients and is only first-order accurate for constant smoothing coefficient. Fortunately these statements only apply

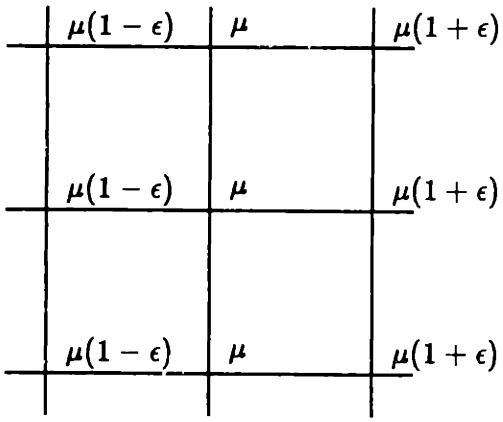


Figure 3.34: Example of spatially varying smoothing coefficients

in the limit of vanishing grid size. As long as the grid size remains finite, the magnitude of the error introduced by the smoothing is governed by the product of its coefficient and Δx.

To demonstrate this latter point, consider once again the solution and order-of-accuracy study in figure 3.7. This solution was actually computed with a spatially varying smoothing included, where $\mu_{\text{min}} \approx 0.025$ and $\mu_{\text{max}} \approx 0.032$. (The minimum-to-maximum difference would be much larger for flow fields with discontinuities). The order-of-accuracy plot shows second-order accuracy, indicating that the error term of largest magnitude is associated with the second-order terms in the basic scheme. It should be noted however that if this analysis were extended to smaller Δx, then one should expect the errors to asymptote to a horizontal line, indicating the inconsistency described above.

3.5.2 Smoothing coefficient distribution

The second task involved in implementing an adaptive smoothing operator is the generation of the spatially varying smoothing coefficient. The

technique used should produce a small value in smooth regions of the flow and a larger value near discontinuities. This can be accomplished using an equation of the form

$$(\mu_{des})_i = \mu_{min}\left[1 + \frac{\mu_{max} - \mu_{min}}{\mu_{min}}\kappa_i\right] \tag{3.170}$$

where $(\mu_{des})_i$ is the desired level of smoothing at node i, and μ_{min} and μ_{max} are the minimum (background) and maximum (discontinuity) levels of smoothing; κ_i is the discontinuity detection function which is 0 in smooth regions and 1 at discontinuities.

One characteristic of discontinuities (which is useful in their detection) is that their second-differences are considerably larger than the second-difference elsewhere in the domain. This leads to a detection function given in one dimension by

$$\kappa_i = \frac{|\phi_{i-1} - 2\phi_i + \phi_{i+1}|}{\phi_{i-1} + 2\phi_i + \phi_{i+1}} \tag{3.171}$$

The denominator in (3.171) ensures that $0 \leq \kappa_i \leq 1$.

In order to use (3.171), one must choose some physical quantity, ϕ, on which to base the detection. The appropriate choice is based upon the types of discontinuities which one expects in the flow-field. For two-dimensional Euler calculations, the expected discontinuities include shock waves and contact surfaces (wakes); an appropriate choice is thus the density. Another appropriate choice (and the one used here) is the total internal energy which varies proportionally to the density in homenthalpic regions of the flow field but which unlike density varies in thermal wakes.

As discussed in the previous section, non-constant smoothing coefficients create convective errors which should be minimized. To achieve this, the smoothing coefficient μ should be as smooth (spatially) as possible. One method of accomplishing this is to smooth μ_i before using it, or

$$\mu_i = (\mu_{des})_{i-1} + 2(\mu_{des})_i + (\mu_{des})_{i+1} \tag{3.172}$$

Unfortunately this requires two passes through all the cells in the domain: one to compute $\mu_{\rm des}$ and one to create a smoothed version μ.

An alternative approach is to write a difference equation to evolve μ in time. This approach, which is taken here, involves adding the equation

$$\mu_i^* = \mu_i + \alpha \left[\underbrace{(\mu_{\rm des})_i - \mu_i}_{\text{forcing function}} + \underbrace{\beta(\mu_{i-1} - 2\mu_i + \mu_{i+1})}_{\text{smoothing term}} \right] \tag{3.173}$$

to the set to be solved. Here μ^* is the new (smoothed) value of the smoothing coefficient, β controls the smoothness of μ, and α is an adjustable parameter to ensure that the integration of (3.173) is stable. The creation of this equation is based upon the assumption that μ does not vary rapidly in time and hence the old (previous) values of μ can be used in the smoothing function. This equation can be computed in one pass through the cells by simultaneously accumulating the terms needed for $\mu_{\rm des}$ and the smoothing term in (3.173).

An added advantage of using an equation such as (3.173) to evolve μ is that one can initialize the integration with a large value of μ. Early iterations thus have the benefit of large amounts of smoothing which enhances the overall scheme's robustness and which hasten convergence. As long as α is large enough so that the time asymptotic distribution of μ is achieved before the governing equations reach steady state, the final solution is independent of the evolution of μ.

In two dimensions, this procedure is extended in a straightforward manner. Immediately before the **SMTH** step, the following is applied

WGHT for every fine cell, accumulate the sums

$$(\delta\kappa)_{\rm sw} = 3\phi_{\rm sw} - \phi_{\rm se} - \phi_{\rm ne} - \phi_{\rm nw} \tag{3.174a}$$

$$(\overline{\delta\kappa})_{\rm sw} = 3\phi_{\rm sw} + \phi_{\rm se} + \phi_{\rm ne} + \phi_{\rm nw} \tag{3.174b}$$

$$(\delta\mu)_{\rm sw} = 3\mu_{\rm sw} - \mu_{\rm se} - \mu_{\rm ne} - \mu_{\rm nw} \tag{3.174c}$$

These formulae are cyclically permuted for the contributions to the other nodes. Then for every node, compute

$$(\mu_{des})_i = \mu_{min}\left[1 + \frac{\mu_{max} - \mu_{min}}{\mu_{min}}\frac{\sum(\delta\kappa)}{\sum(\overline{\delta\kappa})}\right] \tag{3.175}$$

and

$$\mu_i^* = \mu_i + \alpha\left[((\mu_{des})_i - \mu_i) + \beta(\sum(\delta\mu))\right] \tag{3.176}$$

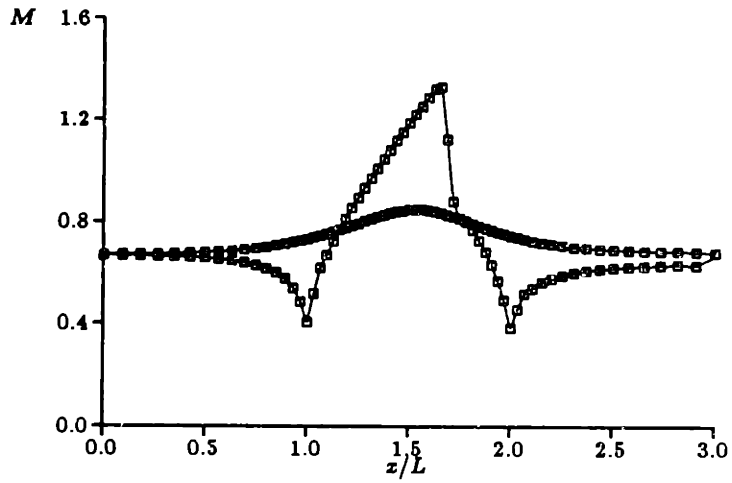
The sums here refer to the four cells (for interior nodes) adjoining the node. For all results shown, $\alpha = 0.10$, $\beta = 0.50$, $\mu_{min} = 0.025$, and $\mu_{max} = 5.00$ although values of μ greater than 0.10 are extremely rare due to the smoothing term in (3.176).

The effectiveness of the spatially varying smoothing developed here is demonstrated by recomputing the solution for the transonic problem given in figures 3.31 and 3.32. The new results are shown in figure 3.35, where parts a and b are again the surface Mach number and total pressure loss distributions. The final part of this figure (c) contains the surface distribution of the smoothing coefficient. One can clearly see that the smoothing coefficient is larger at the leading and trailing edges and at the shock. This solution contains a non-oscillatory shock as well as low spurious total pressure losses away from the shock and stagnation points and boundaries.

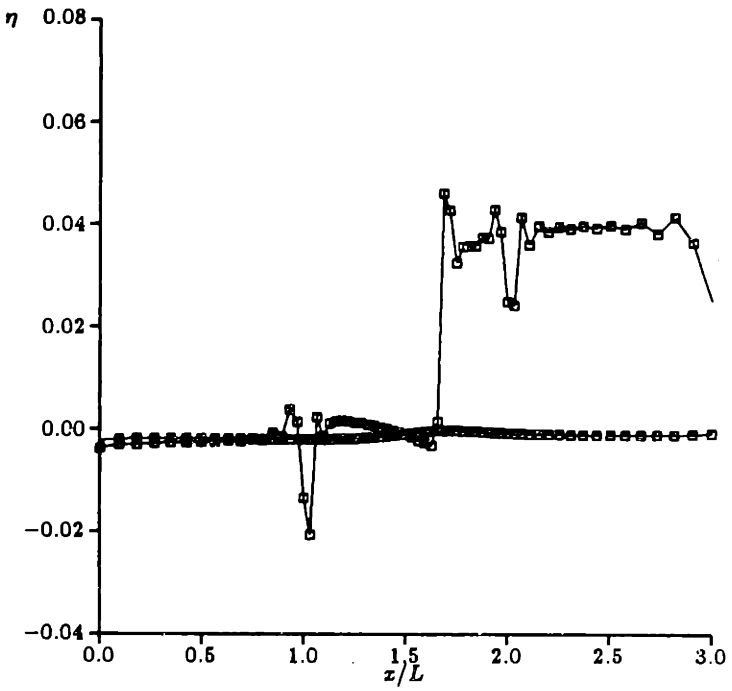
3.6 Convergence Checking

The previous sections of this chapter have been concerned with the time integration scheme used to advance the unsteady governing equations to a time-asymptotic solution. This section discusses the technique used to determine when the current solution *is* in fact an adequate approximation to the steady state solution.

The most popular method of ensuring that the time-asymptotic solution has been achieved is to measure how well the current solution satisfies the

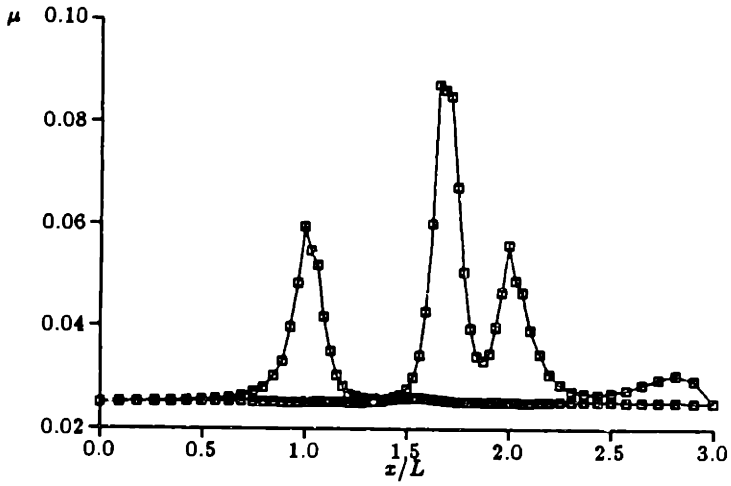


a: surface Mach number distributions



b: surface total pressure loss distributions

Figure 3.35: Surface Mach number and total pressure loss for transonic channel flow test case. Varying smoothing coefficient



c: surface smoothing coefficient distributions

Figure 3.35: Surface Mach number and total pressure loss for transonic channel flow test case. Varying smoothing coefficient

steady state part of the governing equation. The residual, which is computed by substituting the current solution into the steady part of the difference form of the governing equations, is the measure most often employed. In order that the residual vanish in the steady state, many procedures utilize a modified residual definition which includes the effects of added terms (for example smoothing). The procedure is said to converge when the magnitude of the residual falls below a specified value, typically a few orders-of-magnitude smaller than the residual of the initial guess.

An alternative approach is to monitor the changes in the dependent variables from integration cycle to cycle. If the changes in the dependent variables are small (for example, less than some specified percentage of the value of the dependent variable), then the solution is said to converge. While this approach does not explicitly state the degree to which the governing equations are satisfied, it has the advantage that it indicates when further

integration is futile. In addition, the effects of the multiple-grid accelerator are easily incorporated.

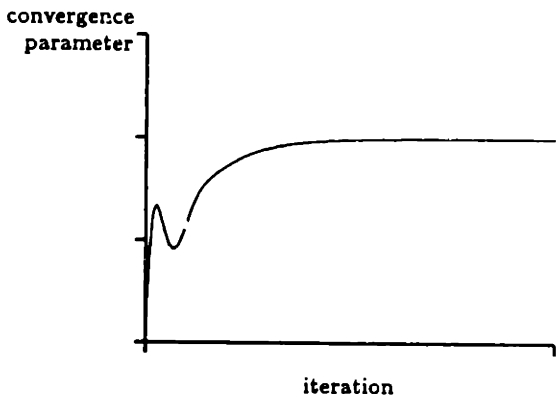
However, in most engineering calculations, the solution everywhere is not of interest but rather the determination of some parameter such as a force coefficient, shock location, or measure of heat release is the desired output. In such cases, it is not necessary that the residual or maximum change be smaller than a specified quantity, but rather that the desired parameter (hereinafter called the *convergence parameter*) reach its steady state value to within a specified tolerance. This forms the basis of the convergence algorithm used here.

The choice of an appropriate convergence parameter is governed by the nature of the problem being solved. In many cases, more than one convergence parameter is appropriate and of course should be used. For example, in isolated airfoil calculations the lift and drag coefficients are appropriate, while in cascades the turning and loss are often desired outputs.

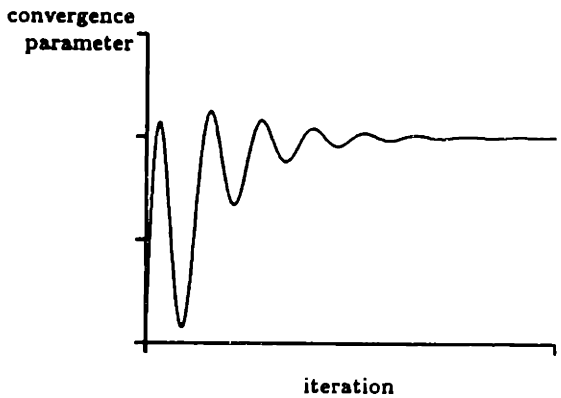
Determining if the convergence parameter has reached its steady-state value is complicated by the various approaches that it can make to its steady state value. For example, it is possible that the steady solution is approached somewhat monotonically, as shown in figure 3.36a, or as a decaying oscillation, as shown in figure 3.36b. Hence it is not sufficient to monitor the change of the parameter (the slope in figure 3.36) since it can vanish (at local extrema) many times before steady state as achieved.

The only sure way of determining that the convergence parameter is not changing is to monitor its value over the last n iterations, where n is chosen to be large enough such that it can capture at least one period of the oscillatory behavior in figure 3.36b. In practice it has been found that n should be on the order of the size of the longest dimension in the domain to allow for waves which reflect between the boundaries.

This leads to the sliding window convergence strategy used here in which



a: monotonic



b: oscillatory

Figure 3.36: Possible approaches to steady state

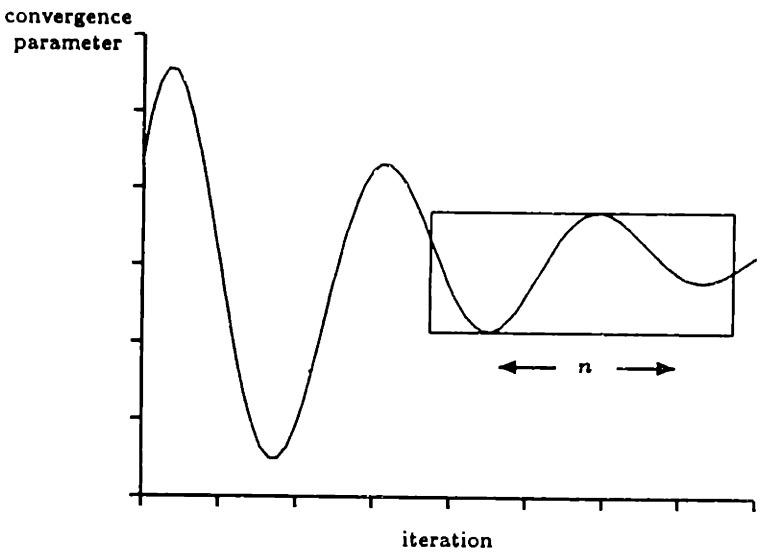


Figure 3.37: Convergence window

the values of the convergence parameter in the last n iterations is stored in a window as shown in figure 3.37. The height of the window is determined by the minimum and maximum values of the convergence parameter in the window. Convergence is said to occur if the height of the window is less than some specified tolerance.

This method of detecting convergence will succeed as long as the window is wide enough (n is sufficiently large) to capture all unsteady waves and the magnitude of the oscillations does not increase in time.

For efficiency, the above algorithm is implemented with a circular list in which the last n values are saved. Hence adding a new value only involves overwriting the value n iterations ago and moving the list pointer. Minimum and maximum value calculations are made efficient by storing the locations within the list of the current minimum and maximum values. These values are updated if the convergence parameter for the last iteration exceeds either of them or if the convergence parameter n iterations ago (the one being deleted) is the current minimum or maximum.

As a protective mechanism, the above algorithm is initiated only after the maximum change over the whole domain decreases more than a specified amount (typically to less than 10 percent of its initial value); this ensures that starting anomalies do not prematurely indicate convergence. It should be noted that this added mechanism has never been needed, but it is included for safety.

Chapter 4

Embedded Mesh Procedure

The previous chapter contained a description of the scheme used to numerically integrate the governing equations to steady state on a logically regular grid. In this chapter, that scheme is extended to computations on embedded grids.

This chapter begins by describing the embedded mesh procedure from a conceptual view-point. This is followed by a discussion of the interface formulation, first in one and then in two dimensions. The properties of the embedded mesh scheme are then derived and demonstrated. Since the interface treatment is intimately combined with the basic numerical integration scheme, the chapter concludes by presenting a full flow-chart of the entire integration scheme.

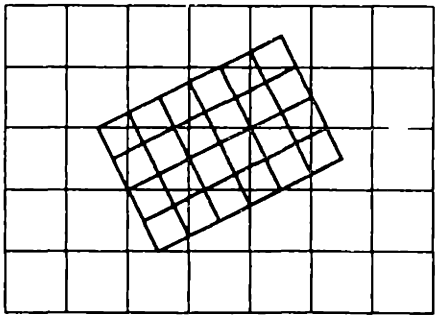
4.1 Basic Concepts

The grid adaptation method developed in this work achieves high accuracy by approximating the governing equations on a composite grid, composed of a fixed global grid and any number of embedded regions. Various means of generating embedded regions have been reported, including the superposition of a non-aligned, regularly-shaped embedded patches[11] as shown in figure 4.1a, the superposition of an aligned, regularly-shaped embedded patch[84] as shown in figure 4.1b, and the superposition of an aligned, irregularly-shaped embedded patch as shown in figure 4.1c; the latter approach is used here.

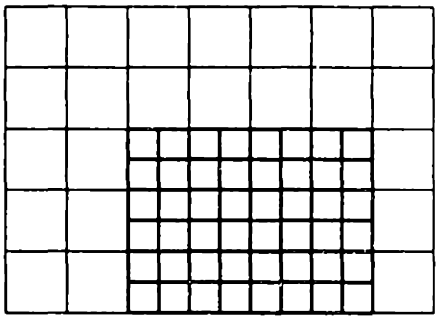
As can be seen from figure 4.1c, the irregularly-shaped embedded region can be viewed as the result of dividing some of the global grid cells. In fact in the present work, such a cell division process is used to create the embedded patches. That process is described briefly in the following paragraph with a thorough explanation contained in section 5.3.3.

For illustrative purposes, consider the global grid (level 0 grid) shown in perspective in figure 4.2a, in which four cells are labelled A, B, C, and D. Any multiple-grid levels below that global grid (for example a level -1 grid) are not shown for clarity. Part b of the figure shows the effect of dividing cells C and D. The division process begins by creating nodes at the midpoints of the faces and at the centroids of cells C and D. Four new cells for each original cell are then created by connecting these new nodes. These new cells, labelled $C1 \ldots C4$ and $D1 \ldots D4$ are said to be level 1 cells.

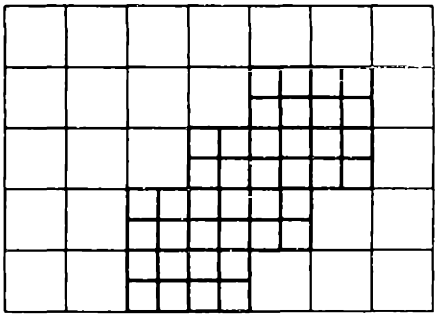
Recall from chapter 3 the distinction between 'fine' and 'coarse' grid cells: a *fine cell* is one which has the highest level number (is the finest) of all coincident cells whereas all other coincident cells are considered *coarse cells*. In other words, if one were to view the coincident grids stacked on top of each other (as in figure 4.1), the cells that one would see are the fine cells. An easy way of deciding if a cell is fine or coarse is that coarse cells have a



a: regularly-shaped, non-aligned

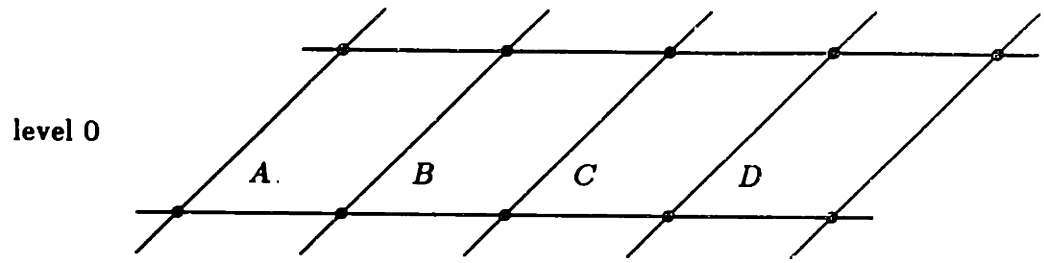


b: regularly-shaped, aligned

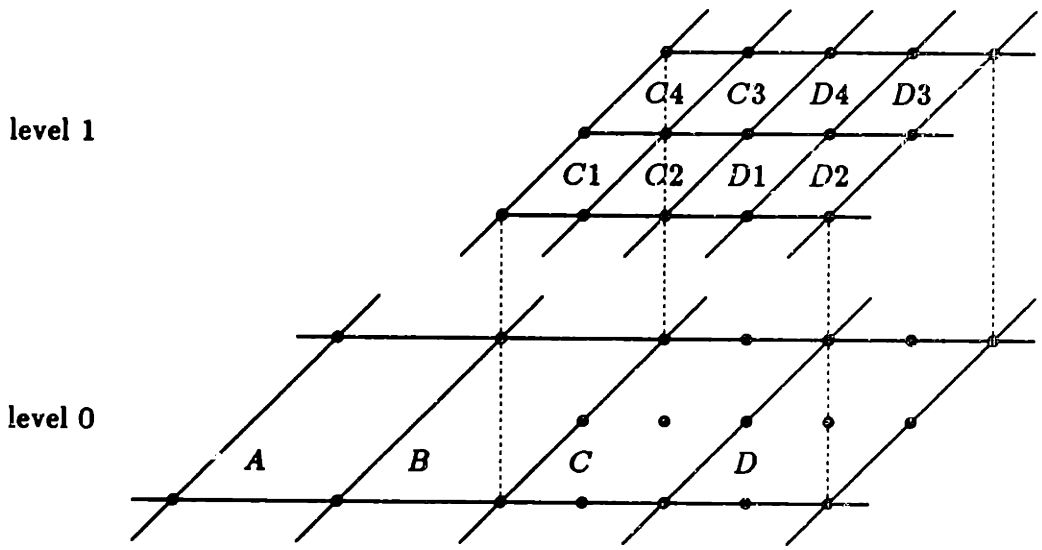


c: irregularly-shaped, aligned

Figure 4.1: Embedded mesh topologies



a: before embedding



b: after dividing cells C and D

Figure 4.2: Embedded grid creation

node at their centers whereas fine cells do not. Hence in figure 4.2b, cells A, B, $C1 \ldots C4$ and $D1 \ldots D4$ are fine whereas cells C and D are coarse.

Examining figure 4.2b, one can notice that away from the embedded mesh interface (that is away from cells B and C), the grid either looks simply like a fine grid (as in cell A) or as a multiple-grid arrangement (as in cells D and $D1 \ldots D4$). Hence from a grid topology view-point, the multiple-grid accelerator is an ideal mechanism for coupling embedded and global grids.

Also recall from chapter 3 that the integration scheme with the multiple-grid accelerator performs very similar operations on fine and coarse cells. For each cell, the scheme begins by computing a 'change' for the cell (a flux balance for fine cells and a transport of fine grid corrections for coarse cells); the scheme then 'distributes' these changes to the cell's nodes. Hence also from the integration scheme's view-point, the multiple-grid accelerator is an ideal mechanism for coupling embedded and global grid solutions.

Therefore in this work, the embedded and global grids and their solutions are coupled through the multiple-grid accelerator. The scheme employed here can be summarized as:

- Starting on the finest level
  - Visit all cells and calculate the 'change' by
    - a flux balance for fine cells
    - transport the fine grid corrections for coarse cells
  - Distribute the changes in each cell to its nodes
  - Apply boundary conditions and update the dependent variables at each node
- Repeat for all coarser levels

Obviously when operating on the finest level (in this case level 1), all 'changes' are computed by a flux balance. The only exception to the above

description occurs at the embedded mesh interface, which is the subject of the following section.

This idea of coupling embedded and global grids with multigrid is not new; it was first suggested for elliptic equations by Brandt[14] and has been implemented by Brown for the full potential equation[15]. Usab and Murman[34] were the first to use such a technique for hyperbolic equations such as the Euler equations.

Before exploring the properties of the coupling technique as discussed in section 4.3, one must consider the problem of the interface treatment, that is the special operations which are required at the edge of an embedded region.

4.2 Interface Formulation

The major difficulty with computing solutions with embedded meshes is the treatment at the interface (or edge of the embedded domain). Points there must be carefully treated in order to ensure continuity of the solution and its derivatives if possible. Also, the local accuracy and conservation should be preserved there, and the scheme should be stable and propagate waves through the interface appropriately. In addition, if an interface treatment can be derived which naturally 'fits' with the basic integration scheme, then one should expect that its application to new interface topologies should follow in a straightforward manner. These guiding principles then lead to the interface treatment discussed below.

4.2.1 One-dimensional interface

To begin, consider the one-dimensional grid shown in figure 4.3 which is the one-dimensional analog to the two-dimensional grid shown in figure 4.2. Here the nodes are numbered and the cells are lettered.

The simplest interface implementation involves doing nothing special

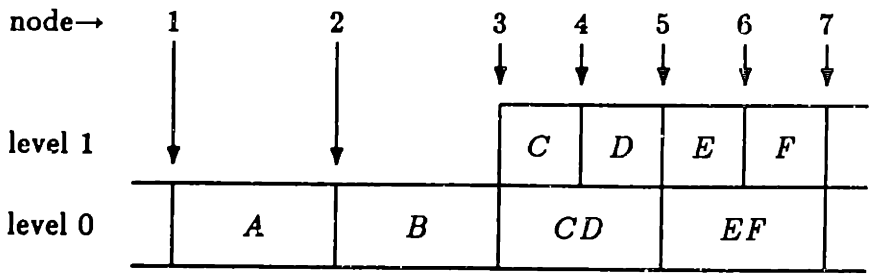


Figure 4.3: One-dimensional interface

at the interface, that is using the basic strategy as described on page 191 directly. For the grid arrangement of figure 4.3, this becomes:

1. for all cells on level 1 (that is C, D, E, and F), compute the 'change' by a flux balance
2. distribute those changes to their nodes (3, 4, 5, 6, and 7)
3. update the dependent variables at those nodes (3, 4, 5, 6, and 7)
4. compute a 'change' for all level 0 cells
   - for cells A and B by a flux balance
   - for cells CD and EF by a transport operator
5. distribute the changes to the nodes (1, 2, 3, 5, and 7)
6. interpolate the corrections to intermediate nodes 4 and 6
7. update the dependent variables at all nodes (1 through 7)

A difficulty with this formulation is that global conservation is no longer enforced because the fluxes for the cells on either side of node 3 are computed using different values of the dependent variable at that node; cell C uses the value before the integration cycle whereas cell B uses a value that

has been already updated with the correction from cell C. In addition to not enforcing conservation, this interface formulation also results in the creation of spurious waves, making the scheme converge very slowly and even sometimes diverge as demonstrated by actual calculations.

This situation can be remedied if the correction at node 3 computed from cell C is not applied to the dependent variable before the flux balance is performed in cell B. This can be accomplished by modifying step 3 above to read:

3. update the dependent variables at all *interior* level 1 nodes (4, 5, 6, and 7)

This is a conservative interface formulation (as will be shown in section 4.3.2) since the flux balances for cells B and C use the same value of the dependent variable at node 3 (the value before the entire integration step).

4.2.2 Two-dimensional interface

Now consider the extension of the above one-dimensional interface treatment to two dimensions. The part of the grid in figure 4.2b in the vicinity of the interface is shown again in figure 4.4, this time with the nodes being numbered.

For cells on the fine side of the interface (that is $C1$ and $C4$), a straightforward extension of the one-dimensional treatment can be employed directly. In other words, a flux balance and distribution are performed as for any fine cell; as in the one-dimensional case, the only difference from other fine cells is that the corrections are not applied to the dependent variables until after the flux balance has been performed on the cell just outside the embedded region.

Unfortunately, the cell just outside the embedded region (that is cell B) cannot be treated as a straightforward extension of the one-dimensional

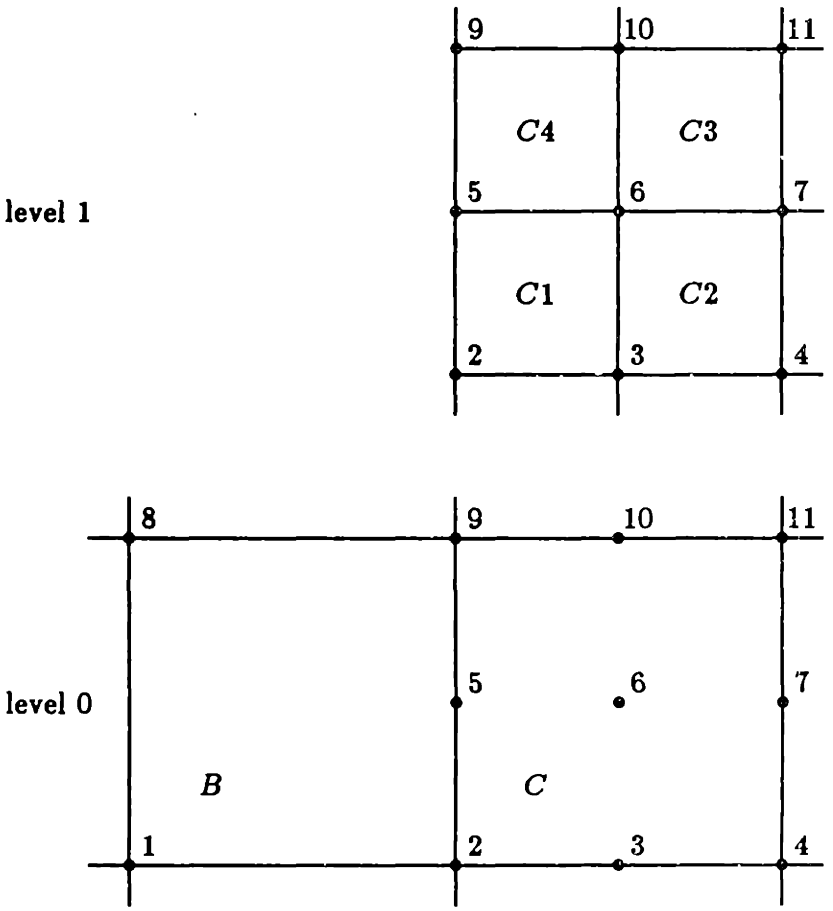


Figure 4.4: Two-dimensional cells near interface

treatment due to node 5, hereinafter termed a *hanging node*. This hanging node complicates both the flux balance and distribution formulae in cell B.

This problem can be circumvented by considering the hanging node to be part of the coarse grid[84], resulting in

$$\mathbf{U}_5 = \frac{\mathbf{U}_2 + \mathbf{U}_9}{2} \tag{4.1}$$

In this case, the flux balance and distribution on cell B is the same as that on any fine cell. Unfortunately, this results in the loss of the corrections at node 5 due to flux balances and distributions on cell $C1$ and $C4$. Hence this approach is not used here.

Instead, the hanging node (node 5) is considered as part of the fine grid, wherein the value of the dependent variables are not necessarily the average of the values of the neighboring interface nodes (2 and 9). Hence appropriate modifications to the flux balance and distribution formulae in the cell just outside the embedded region are required.

As shown in chapter 3, the change in the center of cell B is given by the flux balance

$$\Delta \mathbf{U}_c = -\frac{\Delta t_c}{A_c}\left[\mathcal{F}_s + \mathcal{F}_e + \mathcal{F}_n + \mathcal{F}_w\right] \tag{4.2}$$

where the fluxes through each face are computed by trapezoidal integration, such as

$$\mathcal{F}_s = \frac{\mathbf{F}_{sw} + \mathbf{F}_{se}}{2}(y_{se} - y_{sw}) - \frac{\mathbf{G}_{sw} + \mathbf{G}_{se}}{2}(x_{se} - x_{sw}) \tag{4.3}$$

The trapezoidal integration along the eastern edge of cell B has to be adjusted to account for the presence of node 5, yielding

$$\begin{aligned}\mathcal{F}_e &= \frac{\mathbf{F}_{se} + \mathbf{F}_e}{2}(y_e - y_{se}) - \frac{\mathbf{G}_{se} + \mathbf{G}_e}{2}(x_e - x_{se}) \\ &+ \frac{\mathbf{F}_e + \mathbf{F}_{ne}}{2}(y_{ne} - y_e) - \frac{\mathbf{G}_e + \mathbf{G}_{ne}}{2}(x_{ne} - x_e)\end{aligned} \tag{4.4}$$

If one assumes that node 5 is at the mid-point of the eastern edge (as it is by construction), then (4.4) takes the simpler form

$$\mathcal{F}_e = \frac{\mathbf{F}_{se} + 2\mathbf{F}_e + \mathbf{F}_{ne}}{4}(y_{ne} - y_{se}) - \frac{\mathbf{G}_{se} + 2\mathbf{G}_e + \mathbf{G}_{ne}}{4}(x_{ne} - x_{se}) \tag{4.5}$$

The distribution formulae given by 3.25 still are valid for cell B. Hence the corrections at nodes 2 and 9 due to cell B ($(\delta \mathbf{U}_2)_B$ and $(\delta \mathbf{U}_9)_B$) are computed by

$$(\delta \mathbf{U}_2)_B = \frac{1}{4}[\Delta \mathbf{U}_B + \Delta f_B - \Delta g_B] \tag{4.6a}$$

$$(\delta \mathbf{U}_9)_B = \frac{1}{4}[\Delta \mathbf{U}_B + \Delta f_B + \Delta g_B] \tag{4.6b}$$

However, the change at the center of the cell must also be distributed to node 5. Various techniques have been employed for this distribution, such as geometric weighting[69]. A different distribution formula is used here for the distribution to the hanging node. The basis for it is the observation that if the hanging node did not exist, then the correction at the mid-point of the cell's face would be the average correction from the two adjacent nodes. For the current case, this becomes

$$(\delta \mathbf{U}_5)_B = \frac{(\delta \mathbf{U}_2)_B + (\delta \mathbf{U}_9)_B}{2} = \frac{1}{4}[\Delta \mathbf{U}_B + \Delta f_B] \tag{4.7}$$

or in general

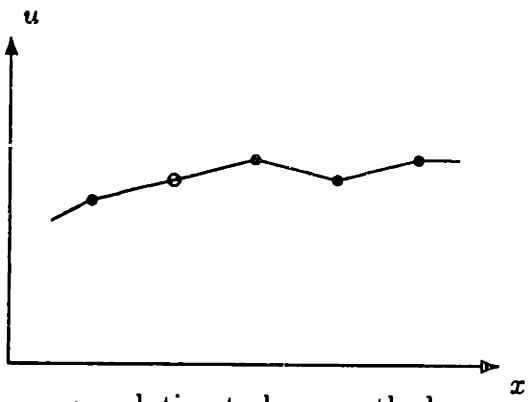
$$(\delta \mathbf{U}_s)_c = \frac{1}{4}[\Delta \mathbf{U}_c - \Delta g_c] \tag{4.8a}$$

$$(\delta \mathbf{U}_e)_c = \frac{1}{4}[\Delta \mathbf{U}_c + \Delta f_c] \tag{4.8b}$$

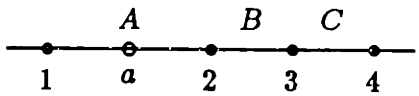
$$(\delta \mathbf{U}_n)_c = \frac{1}{4}[\Delta \mathbf{U}_c + \Delta g_c] \tag{4.8c}$$

$$(\delta \mathbf{U}_w)_c = \frac{1}{4}[\Delta \mathbf{U}_c - \Delta f_c] \tag{4.8d}$$

This is not the same as assuming that the hanging node is part of the grid outside the embedded region (as in (4.1)) because here only the part of the correction due to the cell just outside the embedded region is assumed to be the average of its adjacent nodes; the corrections at the hanging node due to the cells just inside the interface will not in general be the average of the corrections at the adjacent nodes.



a: solution to be smoothed



b: computational grid

Figure 4.5: Solution and grid in the vicinity of a one-dimensional interface. The open circle indicates a false node.

4.2.3 Smoothing terms

In section 3.5, a smoothing (artificial viscosity) operator was developed to damp high frequency errors on topologically regular grids. That operator, which is applied only on the finest grid, is essentially a weighted Laplacian in computational space. Here that algorithm is extended to embedded grids.

As discussed in section 3.5, the smoothing operator is applied only on fine grid cells, even when coarser levels exist as in the multiple-grid accelerator. This same scheme is used for embedded grid calculations, that is the smoothing is applied on a grid which is composed of only fine cells.

Consider the solution and smoothing grid in the vicinity of a one-dimensional interface as shown in figures 4.5a and 4.5b respectively. Away from the interface the smoothing operator for the embedded grid is a simple extension of the smoothing operator developed previously. However a direct application of the smoothing operator developed in section 3.5 at the inter-

face node (node 2) yields

$$\text{smooth}_2 = f_{1,2}(u_1 - u_2) - f_{2,3}(u_2 - u_3) \tag{4.9}$$

where

$$f_{1,2} = f(\mu_1, \mu_2) \quad \text{and} \quad f_{2,3} = f(\mu_2, \mu_3) \tag{4.10}$$

which in terms of differential operators becomes

$$\text{smooth}_2 = (f_{2,3} - 2f_{1,2})\,\Delta x\,\mathrm{U}_x + \left(2f_{1,2} + \frac{1}{2}f_{2,3}\right)(\Delta x)^2\,\mathrm{U}_{xx} \tag{4.11}$$

Since by construction $f_{1,2} \approx f_{2,3}$, the ratio of the diffusion and convection terms for the resulting smoothing operator is given by

$$\frac{\text{diffusion}}{\text{convection}} = -\frac{5}{2}\Delta x \frac{\mathrm{U}_{xx}}{\mathrm{U}_x} \tag{4.12}$$

which indicates that the spurious convective operator is large, at least as compared with the diffusive term. As mentioned in section 3.5, one wants the spurious convection operator to be as small as possible, which is not possible with the interface smoothing operator discussed above.

To eliminate this problem, recall from section 3.5 that (4.9) was originally generated based upon the second difference operator

$$\text{smooth}_2 = \left[f_{1,2}\frac{u_2 - u_1}{x_2 - x_1} - f_{2,3}\frac{u_3 - u_2}{x_3 - x_2}\right]\frac{x_3 - x_1}{2} \tag{4.13}$$

If the spatial resolution is roughly constant, the spatial differences can be eliminated, resulting in (4.9). However near the interface, the spatial resolution varies by approximately a factor of two and hence this approximation is no longer valid.

To circumvent this problem, one needs to revert back to (4.13), which can be rewritten in the form

$$\text{smooth}_2 = \frac{f_{1,2}}{2}\frac{x_3 - x_1}{x_2 - x_1}(u_2 - u_1) - \frac{f_{2,3}}{2}\frac{x_3 - x_1}{x_3 - x_2}(u_3 - u_2) \tag{4.14}$$

In order to use (4.14), the smoothing contribution from cells both just inside and just outside the embedded interface need to be adjusted due to the interface. In practice, this turns out to be difficult to implement, especially for complex interface shapes in two dimensions, and hence some other variant of (4.13) is needed.

An alternative way of implementing the spirit of (4.13) is to create a *false node* at the mid-point of cell A, as shown by the open circle named a in figure 4.5. The value of the dependent variable at the false node is computed by linear interpolation in cell A.

The basic smoothing operator (4.9) can now be employed using the false node, or

$$\text{smooth}_2 = f_{a,2}(u_a - u_2) - f_{2,3}(u_2 - u_3) \tag{4.15}$$

which has the same properties as the smoothing operator away from the interface, namely that it is purely diffusive for $f_{a,2} = f_{2,3}$.

The real benefit of using the false nodes is evident in two dimensions. Consider the two-dimensional grid in the vicinity of an interface, shown in figure 4.6, which is simply the collection of fine grid cells from figure 4.4. Here there are three false nodes used for smoothing, denoted by open circles and labelled a, b, and c. The values of the dependent variables at these nodes is computed by bilinear interpolation in cell B.

Implementation of the smoothing operator in the vicinity of the embedded mesh interface thus requires that cells which are just outside an embedded interface compute their smoothing contributions to the interface based upon the false nodes; in all other fine cells, the smoothing operator remains unchanged from that described in section 3.5.

In other words, the smoothing contribution to node 2 from cell $C1$ is simply

$$\delta \mathbf{U}_2 = \delta \mathbf{U}_2 \quad + \quad \frac{1}{8} f(\mu_2, \mu_3)(\mathbf{U}_3 - \mathbf{U}_2)$$

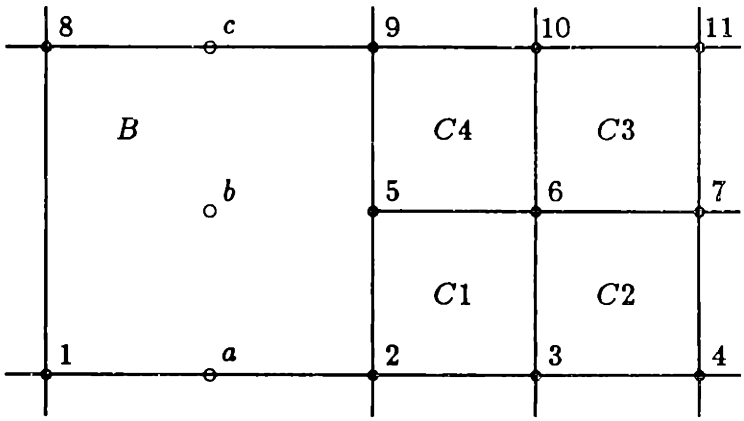


Figure 4.6: Smoothing cells near interface. Open circles indicate false nodes used only for smoothing.

$$+ \quad \frac{1}{8} f(\mu_2, \mu_6)(\mathbf{U}_6 - \mathbf{U}_2) \qquad (4.16)$$
$$+ \quad \frac{1}{8} f(\mu_2, \mu_5)(\mathbf{U}_5 - \mathbf{U}_2)$$

and the contribution from cell B is

$$\begin{aligned} \delta\mathbf{U}_2 = \delta\mathbf{U}_2 \quad &+ \quad \frac{1}{8} f(\mu_2, \mu_a)(\mathbf{U}_a - \mathbf{U}_2) \\ &+ \quad \frac{1}{8} f(\mu_2, \mu_b)(\mathbf{U}_b - \mathbf{U}_2) \qquad (4.17) \\ &+ \quad \frac{1}{8} f(\mu_2, \mu_5)(\mathbf{U}_5 - \mathbf{U}_2) \end{aligned}$$

Notice that the net smoothing change for cell B does not in general vanish and hence is non-conservative as described in section 4.3.2. Even so, this alternative was chosen purposely since the same dilemma occurs at the interface as at boundaries (that is, one can make the smoothing either diffusive or conservative, but not both). But because the embedded mesh interface can be located away from regions where the smoothing is large, this error can be controlled.

The computation of the spatially varying smoothing coefficient is extended to the embedded grid in the same way as the smoothing itself. The integration of the smoothing coefficient equation is performed on the same collection of fine cells as above, with the integration scheme unaltered away from interfaces. At interfaces, the integrator is modified using false nodes to evaluate the appropriate spatial differences. A full description of the process is contained in section 4.4.

4.3 Properties

In this section the properties of the integration scheme in the vicinity of the interface between the global and embedded meshes are examined. It is assumed that the properties of the scheme away from the interface are the same as those described in chapter 3 for the integration scheme with the multiple-grid accelerator and hence are not discussed here.

4.3.1 Accuracy and consistency

As in section 3.3.1, the local accuracy and consistency of the numerical integration scheme can be analytically determined by forming a modified differential equation and comparing it to the governing differential equation. For simplicity, this analysis is performed in one dimension, using the nomenclature of figure 4.3.

The modified differential equation which results from Taylor series expansions about node 3 is given by

$$\begin{aligned} \mathbf{U}_t + \frac{12 + 5\sigma_d + 4\sigma_i}{8\alpha}\mathbf{F}_x &= \frac{\Delta t}{\Delta x}\left[\frac{4 - 3\sigma_d - 4\sigma_i}{8\alpha}\mathbf{F}_t\right] \\ &+ \left(\frac{\Delta t}{\Delta x}\right)^2\left[\frac{-\sigma_d}{8\alpha}\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\mathbf{F}_t\right] \\ &+ \left(\frac{\Delta t}{\Delta x}\right)^3\left[\frac{-\sigma_d}{8\alpha}\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\right)^2\mathbf{F}_t\right] \end{aligned} \tag{4.18}$$

$$
\begin{aligned}
&+ \quad \Delta x \left[\frac{12 - 5\sigma_d - 8\sigma_i}{16\alpha} \mathbf{F}_{xx} \right] \\
&+ \quad \Delta t \left[-\frac{8\alpha^2 + 12\sigma_i + 23\sigma_d + 20}{16\alpha} \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \mathbf{F}_{xx} \right. \\
&\qquad \left. + \frac{16\alpha^2 + 12\sigma_i + 15\sigma_d + 20}{32\alpha} \left. \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \right|_x \mathbf{F}_x \right] \\
&+ \quad O\left(\frac{(\Delta t)^2}{\Delta x}, \frac{(\Delta t)^3}{(\Delta x)^2}, \ldots \right)
\end{aligned}
$$

where $\alpha\Delta t$ is the time step assumed for the Taylor series in time at node 3. As in chapter 3, σ_i and σ_d are respectively the injection and inward distribution coefficients for the multiple-grid accelerator.

Since only the steady state is of importance in this scheme, α can be chosen arbitrarily and hence the coefficient on the $\mathbf{F}_x$ term can be set to unity. One can also notice that all the zero-th order terms ($\Delta t/\Delta x$ and its various powers) vanish in the steady state so that the interface formulation is consistent with the governing equation. The first order terms do not vanish in time and hence the approximation is only first order accurate. But since this first order inaccuracy only occurs along a line, its effect diminishes as the mesh is refined, resulting in global second order accuracy.

The accuracy of the scheme can be demonstrated by computing solutions on grids of various refinement and comparing the solutions. Using the test case from section 3.3.1 (a 10 percent thick sine-squared bump in subsonic flow), one obtains the solutions shown in figure 4.7. A 49×17 computational grid with one embedded region is depicted in figure 4.7a with its corresponding computed Mach number contours in figure 4.7b. Solutions were also computed on 25×9 and 97×33 grids, each with one level of embedded grid. The L_2-norm of the total pressure loss (the error indicator introduced in section 3.3.1) for each of these cases is plotted versus the average Δx on the global grid in part c of the figure. Although second order accuracy was not obtained (as can be seen by the difference between the

computed errors and the line with a slope of 2), better than first order accuracy has been demonstrated; the difference between the analytically derived and demonstrated is probably due to smoothing which is not included in the former.

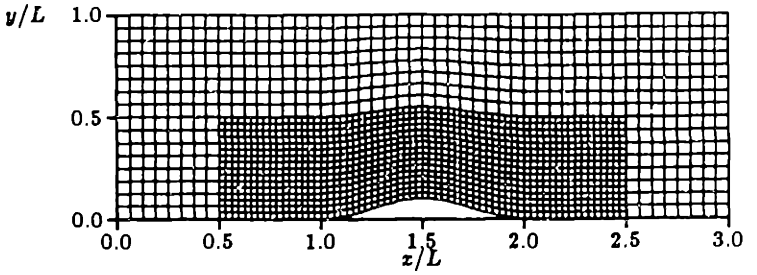
One can also note from figure 4.7b that the Mach number contours pass smoothly through the embedded mesh interface, indicating that the solution and its first derivative are continuous there.

4.3.2 Conservation

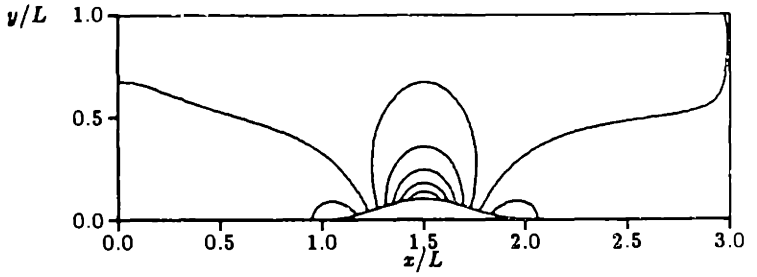
The development of the interface formulation was guided by the desire for the scheme to be conservative (at least for the convective terms). As in section 3.3.3, the conservative property can be examined by summing the total corrections over the entire domain. Using the nomenclature of figure 4.4, the sum of the convective changes in cells B and $C1 \ldots C4$ does not depend on the interface (faces $2-5$ and $5-9$) as would be the case if cell C were not divided. This is both because the eastern flux in cell B is performed using a three-point integration and because the values at the interface (nodes 2, 5, and 9) are not updated until the flux balance in cell B has been performed.

Unfortunately the smoothing operator in cell B is not conservative. This can be seen by summing the smoothing contributions to its various nodes

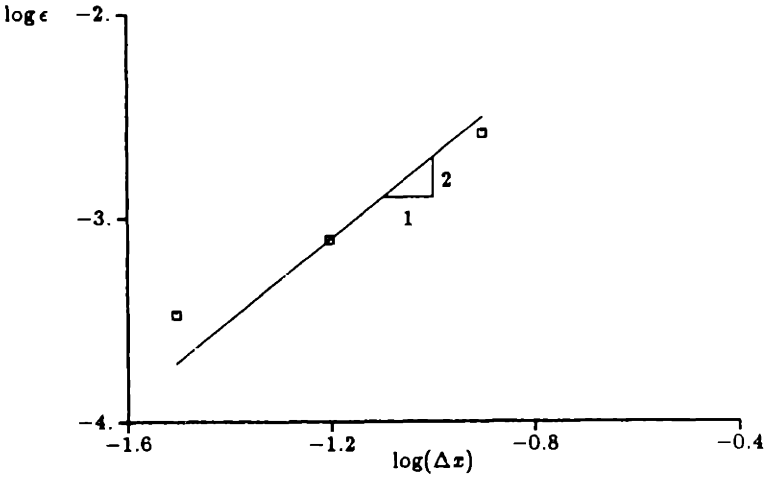
$$\begin{aligned}
\sum_{\text{all nodes}} \frac{\text{smooth}}{\mu} &= (\mathbf{U}_8 + \mathbf{U}_9 + \mathbf{U}_2 - 3\mathbf{U}_1) \\
&+ \left(\frac{\mathbf{U}_1 + \mathbf{U}_2}{2} + \frac{\mathbf{U}_1 + \mathbf{U}_2 + \mathbf{U}_8 + \mathbf{U}_9}{4} + \mathbf{U}_5 - 3\mathbf{U}_2\right) \\
&+ \left(\mathbf{U}_2 + \frac{\mathbf{U}_1 + \mathbf{U}_2}{2} + 2\frac{\mathbf{U}_1 + \mathbf{U}_2 + \mathbf{U}_8 + \mathbf{U}_9}{4}\right. \\
&\quad \left. + \frac{\mathbf{U}_8 + \mathbf{U}_9}{2} + \mathbf{U}_9 - 3\mathbf{U}_5\right) \\
&+ \left(\frac{\mathbf{U}_8 + \mathbf{U}_9}{2} + \frac{\mathbf{U}_1 + \mathbf{U}_2 + \mathbf{U}_8 + \mathbf{U}_9}{4} + \mathbf{U}_5 - 3\mathbf{U}_9\right)
\end{aligned} \tag{4.19}$$



a: computational grid, 49 × 17 with embedded region



b: Mach number contours on 49 × 17 grid, $\Delta M = 0.05$



c: accuracy versus grid resolution

Figure 4.7: Demonstrated accuracy on subsonic test case with one embedded region

$$+ \quad (\mathbf{U}_1 + \mathbf{U}_2 + \mathbf{U}_9 - 3\mathbf{U}_8)$$

where the smoothing coefficient has been taken to have the constant value μ. This can be collapsed to

$$\sum_{\text{all nodes}} \frac{\text{smooth}}{\mu} = 2\mathbf{U}_2 - \mathbf{U}_5 + 2\mathbf{U}_9 \tag{4.20}$$

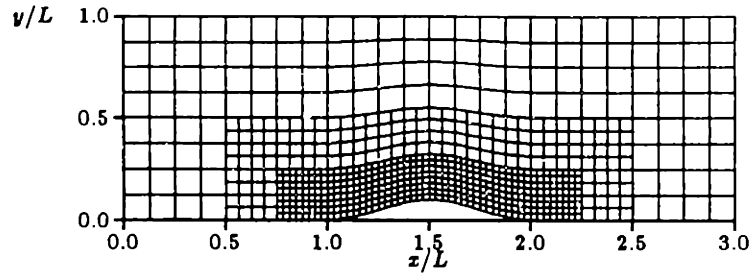
which does not vanish, and hence the smoothing formulation is not conservative.

The conservation property has been demonstrated using the subsonic sine-squared bump problem with the doubly embedded grid shown in figure 4.8a, yielding the solution shown in figure 4.8b. Again the smoothness of the Mach number contours indicate that the solution and its first derivative are continuous across the interfaces. In figure 4.8c, the mass flux through the channel is plotted versus axial position. This plot shows the mass flow to be very nearly constant with no appreciable errors being generated at the interfaces, demonstrating a conservative interface.

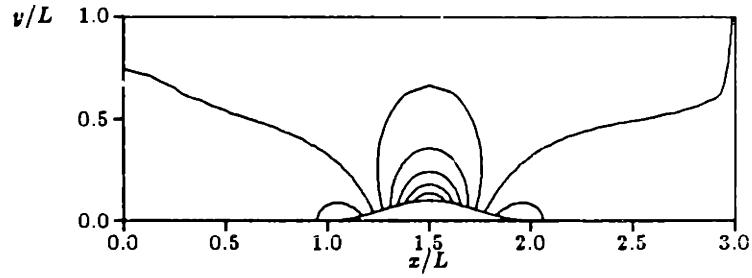
One word of caution is required however. As noted above, the smoothing is definitely not conservative at the interface even though the convective fluxes are. In the case of figure 4.8, the interfaces were placed in regions where very little smoothing was used. If on the other hand one were to place the interface near a region of large smoothing, for example very near and parallel to a shock, the scheme would not be conservative, sometimes even causing divergence of the integration scheme.

4.3.3 Stability

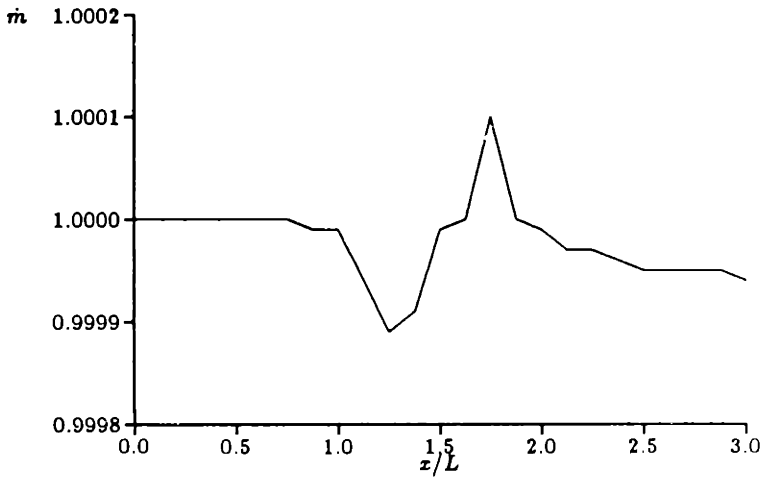
Although the energy stability analysis method described in section 3.3.2 is capable on being extended to embedded mesh solutions, the procedure is significantly complicated by the presence of the interface nodes. The difficulty arises because the operator identities listed in table 3.2 have to be extended to account for end effects which then propagate through the



a: computational grid, 25 × 9 with 2 embedded regions



b: Mach number contours, $\Delta M = 0.05$



c: mass flux rate versus axial position

Figure 4.8: Demonstrated conservation for the subsonic test case with two embedded regions

inner product manipulations. Even without considering end effects, the stability analysis for the one-dimensional Ni's scheme with one level of the multiple-grid accelerator required the manipulation of over 600 separate inner products; if end effects were included over 2000 terms would result, making the analysis virtually untenable.

Fortunately, analytical determination of the stability bounds for the interface procedure can be circumvented because in practice the scheme as described above has proven to be stable.

4.3.4 Wave propagation

One of the most important considerations in the generation of the embedded mesh interface is that waves enter and exit the embedded region promptly. This interface condition enhances both the scheme's stability and convergence properties.

To examine the current interface treatment, consider the propagation of a simple wave into and out of an embedded region as shown in figures 4.9 through 4.12. In each case the wave is propagated via the one-dimensional simple wave equation

$$u_t \pm u_x = 0 \tag{4.21}$$

where the sign determines if the wave is travelling to the left or right. All solutions are performed with the time step parameter $\lambda = \Delta t/\Delta x = 0.7$.

In figure 4.9, the wave begins just outside the embedded region as shown on the first line. The fluxes all balance on the level 1 grid, yielding no changes as shown on the next line. (Note that because level 1 is the finest grid, there are no transport operators at this level.) The distribution formulae similarly yields no corrections so that after the level 1 update, the wave shape is unaltered. The level 0 flux balance does produce a change for the cell just outside the embedded interface as shown; the transport operation in the coarse level 0 cell yields no change. The distribution on level 0 yields the

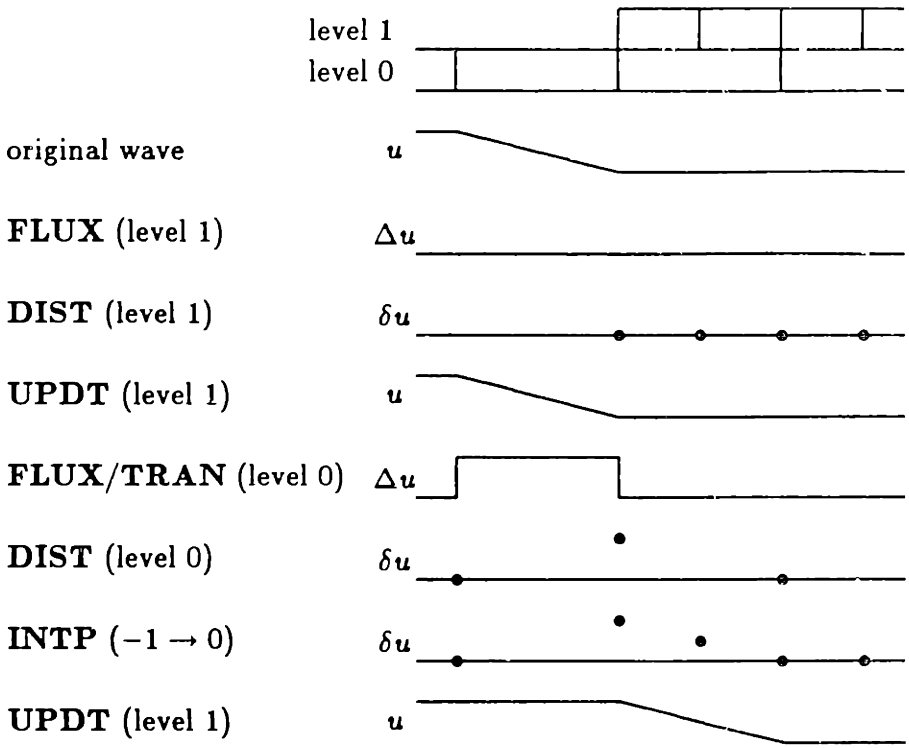


Figure 4.9: Wave entering embedded region, $u_t + u_x = 0$

corrections only at the level 0 nodes as shown. Interpolation is then used to compute corrections on all level 1 nodes and the solution is updated. Note that in this case the wave has entered the embedded region without being altered, as desired.

Waves exiting the embedded region must also pass through the interface promptly. Shown in figures 4.10 through 4.12 are the propagation of waves which start in different embedded mesh cells. Note that in each case the wave exits the embedded region without undue distortion.

The effect of using the multiple-grid accelerator to couple the embedded and global grids and their solutions is demonstrated in figure 4.13a, which is a pseudo-time plot of the propagation of a simple wave through an embedded grid. The speed at which the wave front propagates can clearly be seen to

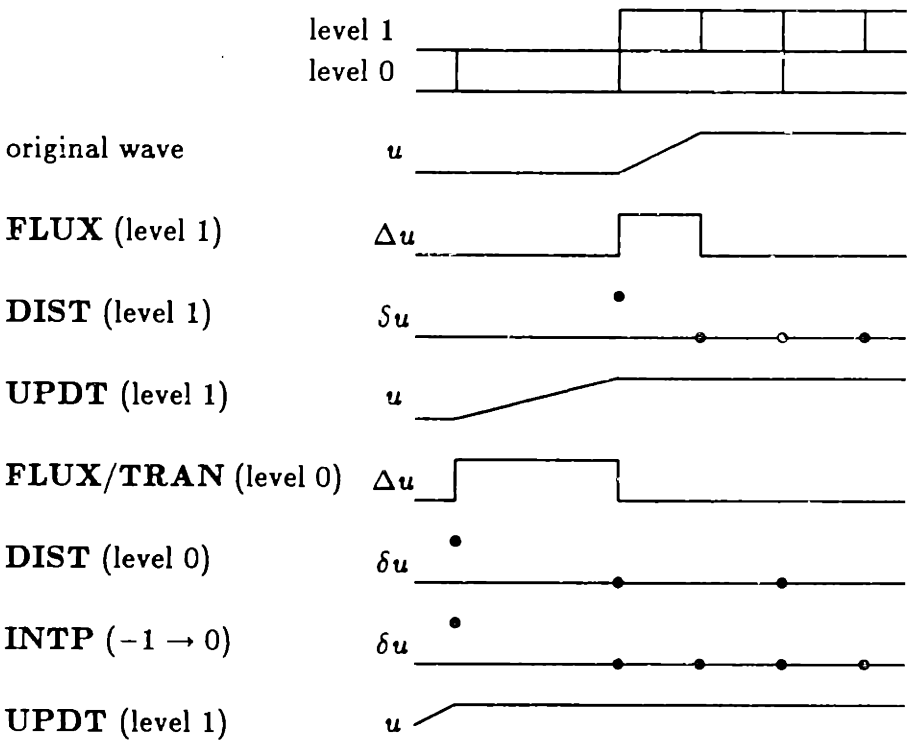


Figure 4.10: Wave exiting embedded region (case A), $u_t - u_x = 0$

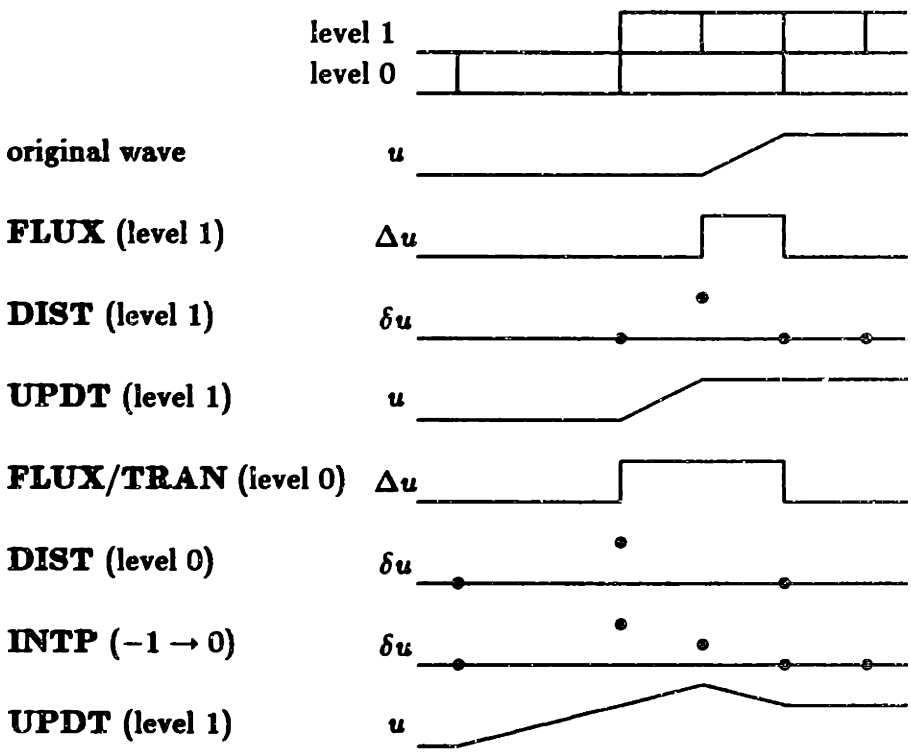


Figure 4.11: Wave exiting embedded region (case B), $u_t - u_x = 0$

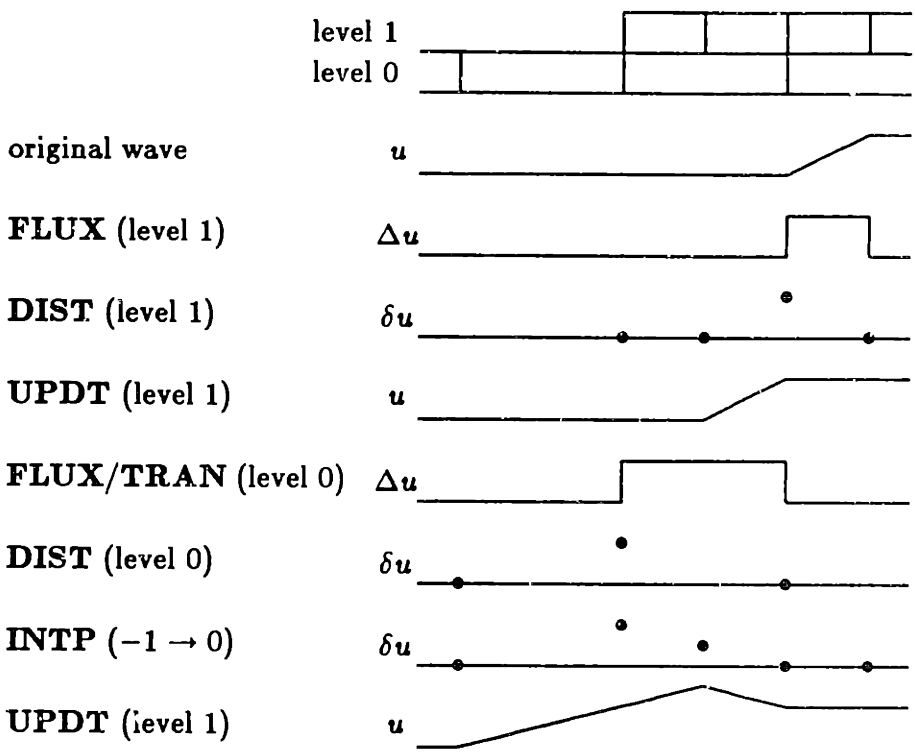
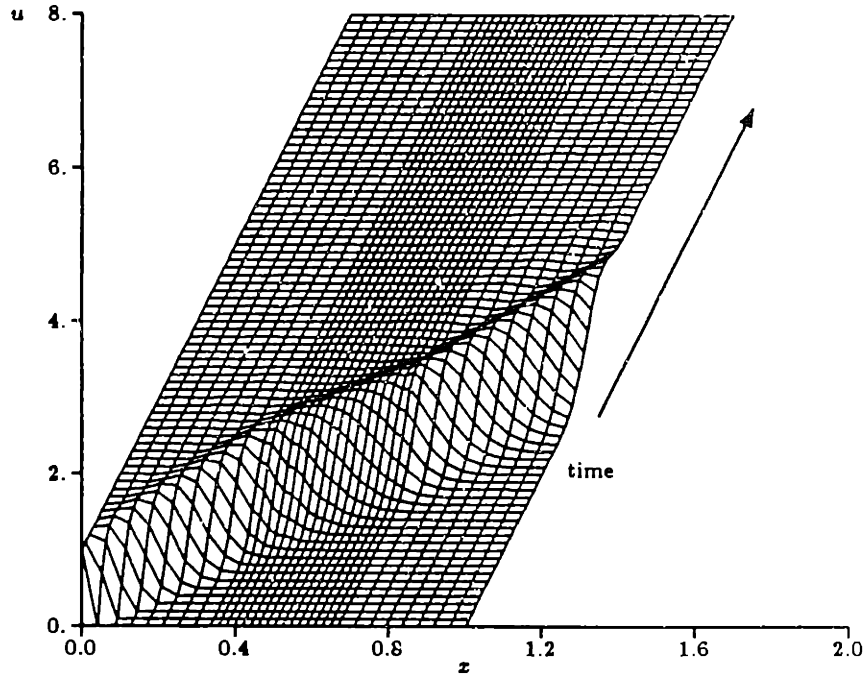


Figure 4.12: Wave exiting embedded region (case C), $u_t - u_x = 0$

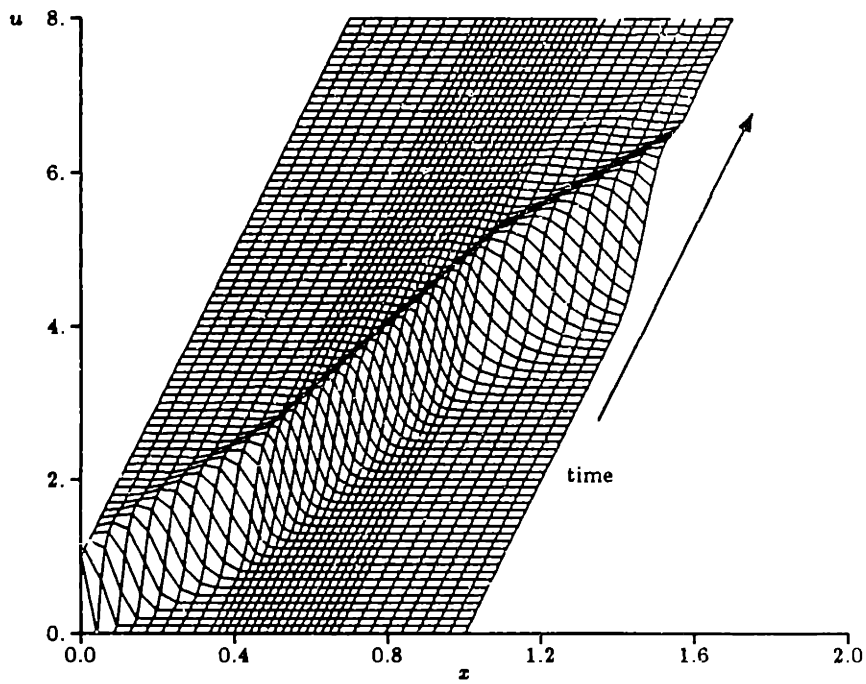
increase through the embedded region and then decelerate after exiting that region. This increase in propagation speed is attributed to the combined effect of propagation on both the fine and coarse grids.

Shown in figure 4.13b is the propagation of the same wave through a simple global grid with non-uniform spacing (the spacing is chosen to be the same as the composite grid). This simulation roughly approximates a coupling scheme in which the embedded and global grid integrations are coupled only through boundary conditions at the interface. As can be seen, the speed of propagation of the wave front decreases as it passes through the small computational cells due to the smaller times steps allowed by the CFL-condition.

The convergence to steady state of solutions computed on embedded grids is vastly enhanced by the interface treatment and multiple-grid coupling procedure. Because waves propagate through embedded regions at coarse grid speeds (actually slightly faster), the convergence rate on the composite grid is comparable to the convergence rates achieved on the coarse grid alone as shown in figure 4.14. The convergence histories for computations on these grids is shown in 4.14 along with the convergence history for the global grid without embedded regions. Note that the embedded grid convergence rates are better than that of all global grid rates (as expected). This effect, which was noted first by Usab[84] for his transonic airfoil calculations, is significantly better than the convergence rates obtained by techniques which simply couple the global and embedded regions at the interface[11]. In the latter technique, wave propagation is restricted to the embedded (fine) grid speed, without the benefit of the multiple-grid accelerator in the embedded regions.



a: embedded grid coupled with multiple-grid



b: non-uniform global domain

Figure 4.13: Demonstrated wave propagation for $u_t + au_x = 0$, $\lambda = 0.7$

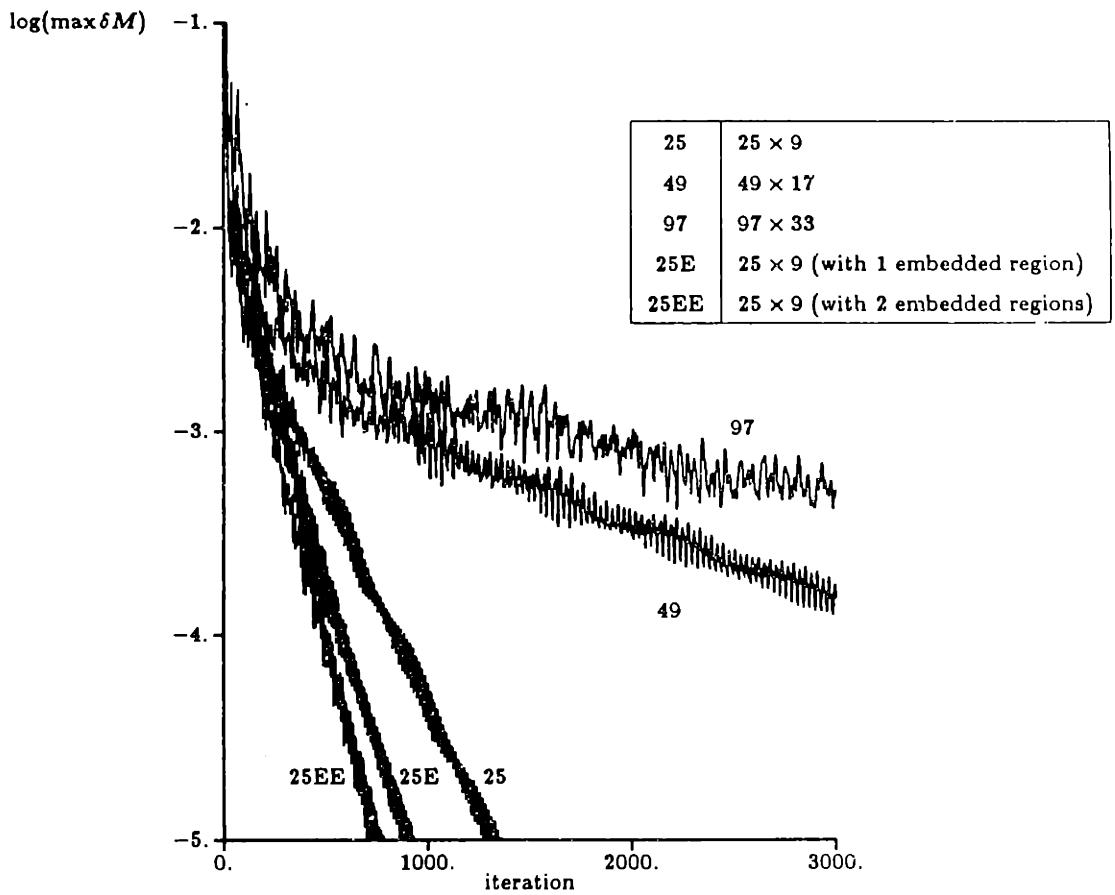


Figure 4.14: Convergence histories for subsonic sine-squared bump case

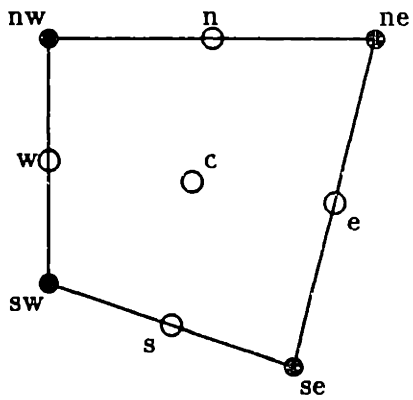


Figure 4.15: General cell. Filled circles denote required nodes, open circles denote optional nodes.

4.4 Summary of Embedded Mesh Procedure

The remainder of this chapter contains a summary of the integration scheme developed in chapters 3 and 4, including the special considerations which are needed in order to properly compute in the vicinity of the embedded mesh interface. Virtually all of the material has been discussed in the preceding pages — it is repeated here only for completeness.

The computational cell on which the computation is based is shown in figure 4.15, where the filled nodes indicate nodes which are always present and those depicted as open circles may or may not exist. The only exception to this rule is that if the center node (c) exists, then the four side nodes (s, e, n, and w) must exist.

The information which is stored at the nodes includes:

x and y the values of the independent variables

U the values of the dependent variables. For the Euler equations, there are 5 storage at each node, 4 for the dependent variables and 1 for the smoothing coefficient

d**U** a storage location in which is accumulated corrections to the dependent variables

In addition, each node can be marked with one of the following node descriptors:

boundary refers to nodes which are on a domain boundary (and not those which are on an embedded mesh interface, unless of course the node is on both)

outside refers to nodes which are at the edge of an embedded region on the next finer level. For example, a level 0 cell can have one or more nodes marked *outside* if a neighboring level 0 cell has been divided.

inside fine refers to nodes which are at the edge of an embedded region created by dividing a cell at the next coarser level. For example if a level 0 cell is divided, the four newly created level 1 cells will have some of their nodes marked *inside fine*.

inside coarse refers to nodes which at the edge of an embedded region on the next finer level. For example, if a level 0 cell is divided, some of its node are marked *inside coarse*.

The overall flowchart of the integration scheme is shown in figure 4.16. Notice that most of the loop is repeated on all successively coarser grids. Each of the procedures indicated is described in the following sections.

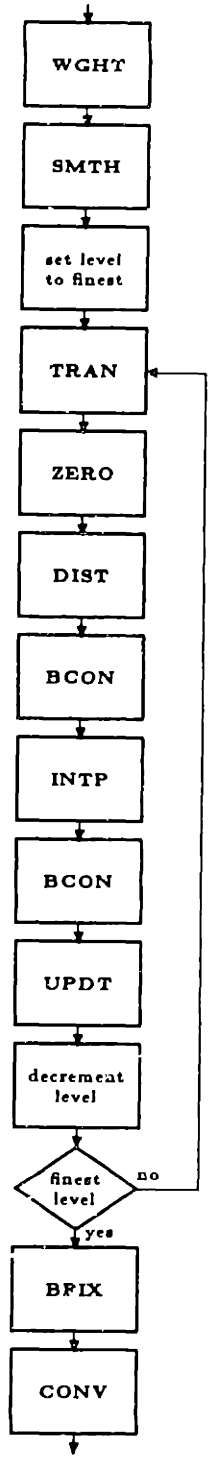


Figure 4.16: Overall flow of embedded grid integration scheme

4.4.1 WGHT

This subroutine computes the smoothing coefficient at every node in the domain. It is the first routine called for every integration step so that it can use $d\mathbf{U}$ as a scratchpad. The procedure that it uses is:

- for every node

$$\begin{aligned} d\mathbf{U}_1 &= 0 \\ d\mathbf{U}_2 &= 0.000001 \\ d\mathbf{U}_3 &= 0 \end{aligned}$$

- for every fine cell
  - create values for nodes which do not exist (this actually only needs to be done for cells which are on a boundary and/or just outside an embedded region)

$$\begin{aligned} \mathbf{U}_s &= (\mathbf{U}_{sw} + \mathbf{U}_{se})/2 \\ \mathbf{U}_e &= (\mathbf{U}_{se} + \mathbf{U}_{ne})/2 \\ \mathbf{U}_n &= (\mathbf{U}_{ne} + \mathbf{U}_{nw})/2 \\ \mathbf{U}_w &= (\mathbf{U}_{nw} + \mathbf{U}_{sw})/2 \\ \mathbf{U}_c &= (\mathbf{U}_{sw} + \mathbf{U}_{se} + \mathbf{U}_{ne} + \mathbf{U}_{nw})/4 \end{aligned}$$

  - for each corner node in this cell (sw is used as an example)
    - if the node is not marked

$$\begin{aligned} d\mathbf{U}_{1,sw} &= d\mathbf{U}_{1,sw} + \mathbf{U}_{4,se} + \mathbf{U}_{4,ne} + \mathbf{U}_{4,nw} - 3\mathbf{U}_{4,sw} \\ d\mathbf{U}_{2,sw} &= d\mathbf{U}_{2,sw} + \mathbf{U}_{4,se} + \mathbf{U}_{4,ne} + \mathbf{U}_{4,nw} + 3\mathbf{U}_{4,sw} \\ d\mathbf{U}_{3,sw} &= d\mathbf{U}_{3,sw} + \mathbf{U}_{5,se} + \mathbf{U}_{5,ne} + \mathbf{U}_{5,nw} - 3\mathbf{U}_{5,sw} \end{aligned}$$

* if the node is just outside and not on a boundary

$$\begin{aligned} d\mathbf{U}_{1,sw} &= d\mathbf{U}_{1,sw} + \mathbf{U}_{4,s} + \mathbf{U}_{4,c} + \mathbf{U}_{4,w} - 3\mathbf{U}_{4,sw} \\ d\mathbf{U}_{2,sw} &= d\mathbf{U}_{2,sw} + \mathbf{U}_{4,s} + \mathbf{U}_{4,c} + \mathbf{U}_{4,w} + 3\mathbf{U}_{4,sw} \\ d\mathbf{U}_{3,sw} &= d\mathbf{U}_{3,sw} + \mathbf{U}_{5,s} + \mathbf{U}_{5,c} + \mathbf{U}_{5,w} - 3\mathbf{U}_{5,sw} \end{aligned}$$

* if the node is at corner of domain

$$\begin{aligned} d\mathbf{U}_{1,sw} &= d\mathbf{U}_{1,sw} + 0 \\ d\mathbf{U}_{2,sw} &= d\mathbf{U}_{2,sw} + 1 \\ d\mathbf{U}_{3,sw} &= d\mathbf{U}_{3,sw} + 0 \end{aligned}$$

* if the western edge is a boundary and the node is not just outside

$$\begin{aligned} d\mathbf{U}_{1,sw} &= d\mathbf{U}_{1,sw} + 4(\mathbf{U}_{4,nw} - \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{2,sw} &= d\mathbf{U}_{2,sw} + 6(\mathbf{U}_{4,nw} + \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{3,sw} &= d\mathbf{U}_{3,sw} + 4(\mathbf{U}_{5,nw} - \mathbf{U}_{5,sw}) \end{aligned}$$

* if the western edge is a boundary and the node is just outside

$$\begin{aligned} d\mathbf{U}_{1,sw} &= d\mathbf{U}_{1,sw} + 4(\mathbf{U}_{4,w} - \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{2,sw} &= d\mathbf{U}_{2,sw} + 6(\mathbf{U}_{4,w} + \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{3,sw} &= d\mathbf{U}_{3,sw} + 4(\mathbf{U}_{5,w} - \mathbf{U}_{5,sw}) \end{aligned}$$

* if the southern edge is a boundary and the node is not just outside

$$\begin{aligned} d\mathbf{U}_{1,sw} &= d\mathbf{U}_{1,sw} + 4(\mathbf{U}_{4,se} - \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{2,sw} &= d\mathbf{U}_{2,sw} + 6(\mathbf{U}_{4,se} + \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{3,sw} &= d\mathbf{U}_{3,sw} + 4(\mathbf{U}_{5,se} - \mathbf{U}_{5,sw}) \end{aligned}$$

* if the southern edge is a boundary and the node is just outside

$$\begin{aligned} d\mathbf{U}_{1,sw} &= d\mathbf{U}_{1,sw} + 4(\mathbf{U}_{4,s} - \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{2,sw} &= d\mathbf{U}_{2,sw} + 6(\mathbf{U}_{4,s} + \mathbf{U}_{4,sw}) \\ d\mathbf{U}_{3,sw} &= d\mathbf{U}_{3,sw} + 4(\mathbf{U}_{5,s} - \mathbf{U}_{5,sw}) \end{aligned}$$

- for each side node in this cell (s is used as an example)

$$\begin{aligned} d\mathbf{U}_{1,s} &= d\mathbf{U}_{1,s} + \mathbf{U}_{4,sw} + \mathbf{U}_{4,e} + 2\mathbf{U}_{4,c} \\ &\quad + \mathbf{U}_{4,w} + \mathbf{U}_{4,sw} - 6\mathbf{U}_{4,s} \\ d\mathbf{U}_{2,s} &= d\mathbf{U}_{2,s} + \mathbf{U}_{4,sw} + \mathbf{U}_{4,e} + 2\mathbf{U}_{4,c} \\ &\quad + \mathbf{U}_{4,w} + \mathbf{U}_{4,sw} + 6\mathbf{U}_{4,s} \\ d\mathbf{U}_{3,s} &= d\mathbf{U}_{3,s} + \mathbf{U}_{5,sw} + \mathbf{U}_{5,e} + 2\mathbf{U}_{5,c} \\ &\quad + \mathbf{U}_{5,w} + \mathbf{U}_{5,sw} - 6\mathbf{U}_{5,s} \end{aligned}$$

- for every node

$$\begin{aligned} \psi &= \gamma\left(1 + \delta\frac{|d\mathbf{U}_1|}{d\mathbf{U}_2}\right) \\ \mathbf{U}_5 &= \mathbf{U}_5 + \alpha(\psi - d\mathbf{U}_5 + \beta d\mathbf{U}_3) \end{aligned}$$

where α, β, γ, and δ are input smoothing coefficients

- for every node

  - initialize the corrections

$$\begin{aligned} d\mathbf{U}_1 &= 0 \\ d\mathbf{U}_2 &= 0 \\ d\mathbf{U}_3 &= 0 \\ d\mathbf{U}_4 &= 0 \end{aligned}$$

  - store $\mathbf{U}_2$ for use in the convergence check

4.4.2 SMTH

This subroutine computes the smoothing for every node in the domain, storing it in the correction array, $d\mathbf{U}$. It is called before any other calculations so that the effect of the smoothing is properly integrated into the embedded mesh procedure. In this section, any reference to $d\mathbf{U}$ without a numeric subscript it is to be understood to apply to $\mathbf{U}_1$, $\mathbf{U}_2$, $\mathbf{U}_3$, and $\mathbf{U}_4$. In addition, the function $\bar{g}$ is defined by

$$\bar{g}(se, sw) = (\mathbf{U}_{se} - \mathbf{U}_{sw})\, f(\mathbf{U}_{5,se}, \mathbf{U}_{5,sw})$$

where by definition

$$f(\mu_a, \mu_b) \equiv \max(\mu_a, \mu_b)$$

The procedure used here is:

- for every fine cell
  - create values for nodes which do not exist (this actually only needs to be done for cells which are on a boundary and/or just outside an embedded region)

$$\begin{aligned}
\mathbf{U}_s &= (\mathbf{U}_{sw} + \mathbf{U}_{se})/2 \\
\mathbf{U}_e &= (\mathbf{U}_{se} + \mathbf{U}_{ne})/2 \\
\mathbf{U}_n &= (\mathbf{U}_{ne} + \mathbf{U}_{nw})/2 \\
\mathbf{U}_w &= (\mathbf{U}_{nw} + \mathbf{U}_{sw})/2 \\
\mathbf{U}_c &= (\mathbf{U}_{sw} + \mathbf{U}_{se} + \mathbf{U}_{ne} + \mathbf{U}_{nw})/4
\end{aligned}$$

  - for each corner node in this cell (sw is used as an example)
    - if the node is not marked

$$d\mathbf{U}_{sw} = d\mathbf{U}_{sw} + (\bar{g}(\mathbf{U}_{se}, \mathbf{U}_{sw}) + \bar{g}(\mathbf{U}_{ne}, \mathbf{U}_{sw}) + \bar{g}(\mathbf{U}_{nw}, \mathbf{U}_{sw}))/8$$

* if the node is just outside and not on a boundary

$$d\mathbf{U}_{sw} = d\mathbf{U}_{sw} + (\bar{g}(\mathbf{U}_s, \mathbf{U}_{sw}) + \bar{g}(\mathbf{U}_c, \mathbf{U}_{sw}) + \bar{g}(\mathbf{U}_w, \mathbf{U}_{sw}))/8$$

* if the western edge is a boundary and the node is not just outside

$$d\mathbf{U}_{sw} = d\mathbf{U}_{sw} + (\bar{g}(\mathbf{U}_{nw}, \mathbf{U}_{sw}))/2$$

* if the western edge is a boundary and the node is just outside

$$d\mathbf{U}_{sw} = d\mathbf{U}_{sw} + (\bar{g}(\mathbf{U}_w, \mathbf{U}_{sw}))/2$$

* if the southern edge is a boundary and the node is not just outside

$$d\mathbf{U}_{sw} = d\mathbf{U}_{sw} + (\bar{g}(\mathbf{U}_{se}, \mathbf{U}_{sw}))/2$$

* if the southern edge is a boundary and the node is just outside

$$d\mathbf{U}_{sw} = d\mathbf{U}_{sw} + (\bar{g}(\mathbf{U}_s, \mathbf{U}_{sw}))/2$$

– for each side node in this cell (s is used as an example)

$$\begin{aligned} d\mathbf{U}_s &= d\mathbf{U}_s + (\bar{g}(\mathbf{U}_{se}, \mathbf{U}_{\cdot}) + \bar{g}(\mathbf{U}_e, \mathbf{U}_s) \\ &\quad +2\bar{g}(\mathbf{U}_c, \mathbf{U}_s) + \bar{g}(\mathbf{U}_w, \mathbf{U}_s) + \bar{g}(\mathbf{U}_{sw}, \mathbf{U}_s))/8 \end{aligned}$$

4.4.3 TRAN

This subroutine transports changes from the fine grid to the centers of coarser grids using the sum of simple injection and inward distribution. The procedure used here is:

- for all coarse cells on this level

$$\begin{aligned} x_\xi &= (x_{ne} + x_{se} - x_{nw} - x_{sw})/2 \\ y_\xi &= (y_{ne} + y_{se} - y_{nw} - y_{sw})/2 \end{aligned}$$

$$
\begin{aligned}
x_\eta &= (x_{ne} - x_{se} + x_{nw} - x_{sw})/2 \\
y_\eta &= (y_{ne} - y_{se} + y_{nw} - y_{sw})/2 \\
d_\xi &= \sqrt{x_\xi^2 + y_\xi^2} \\
d_\eta &= \sqrt{x_\eta^2 + y_\eta^2}
\end{aligned}
$$

$$
\begin{aligned}
a &= \sqrt{\gamma(\gamma-1)\left(\frac{\mathbf{U}_{4,c}}{\mathbf{U}_{1,c}} - \frac{\mathbf{U}_{2,c}^2 + \mathbf{U}_{3,c}^2}{2\mathbf{U}_{1,c}^2}\right)} \\
uc_\xi &= \left|\frac{\mathbf{U}_{2,c}}{\mathbf{U}_{1,c}} y_\xi - \frac{\mathbf{U}_{3,c}}{\mathbf{U}_{1,c}} x_\xi\right| + a d_\xi \\
uc_\eta &= \left|\frac{\mathbf{U}_{2,c}}{\mathbf{U}_{1,c}} y_\eta - \frac{\mathbf{U}_{3,c}}{\mathbf{U}_{1,c}} x_\eta\right| + a d_\eta \\
\frac{\Delta t}{A} &= \frac{\mathrm{CFL}}{\max(uc_\xi, uc_\eta)}
\end{aligned}
$$

$$
\begin{aligned}
d\mathbf{U}_c = d\mathbf{U}_c + \frac{\Delta t}{4A}\Bigg\{ &+ \left[\mathbf{I} + \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(x_{ne} - x_{sw}) - \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(y_{ne} - y_{sw})\right] d\mathbf{U}_{nw} \\
&+ \left[\mathbf{I} + \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(x_{se} - x_{nw}) - \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(y_{se} - y_{nw})\right] d\mathbf{U}_{ne} \\
&+ \left[\mathbf{I} + \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(x_{sw} - x_{ne}) - \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(y_{sw} - y_{ne})\right] d\mathbf{U}_{se} \\
&+ \left[\mathbf{I} + \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(x_{nw} - x_{se}) - \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(y_{nw} - y_{se})\right] d\mathbf{U}_{sw}\Bigg\}
\end{aligned}
$$

where $\mathbf{I}$ is the identity matrix and $\frac{\partial \mathbf{F}}{\partial \mathbf{U}}$ and $\frac{\partial \mathbf{G}}{\partial \mathbf{U}}$ are the Jacobian matrices evaluated with $\mathbf{U}_c$.

4.4.4 ZERO

This subroutine initializes the corrections for all corner nodes which were updated on the previous grid. This is accomplished by the procedure:

- for all coarse cells on this level
  - for each corner node which is not just inside coarse, set

$$d\mathbf{U} = 0$$

4.4.5 DIST

This subroutine, which consumes the majority of processing time, performs flux balances on fine cells and distributes the changes to the nodes. For coarse cells, the change stored at the center of the cell is used for the distribution. The processing is given by:

- for all cells on this level
  - compute the average value of the dependent variables and the change for this cell
    - for fine cells

$$\begin{aligned} d\mathbf{U}_c &= 0 \\ \Delta\mathbf{U} &= 0 \end{aligned}$$

      - for each side (side s will be used as an example)

$$\phi = \begin{cases} 0 & \text{if the side is on a solid boundary} \\ 1 & \text{if the side is not on a solid boundary} \end{cases}$$

      - if the side node does not exist

$$\begin{aligned} d\mathbf{U}_c &= d\mathbf{U}_c + \mathbf{U}_{cw} + \mathbf{U}_{ce} \\ \Delta\mathbf{U} &= \Delta\mathbf{U} + \phi((\mathbf{F}_{sw} + \mathbf{F}_{se})(y_{sw} - y_{se}) \\ &\quad +(\mathbf{G}_{sw} + \mathbf{G}_{se})(x_{sw} - x_{se})) \end{aligned}$$

      - if the side node exists

$$\begin{aligned} d\mathbf{U}_c &= d\mathbf{U}_c + (\mathbf{U}_{sw} + 2\mathbf{U}_s + \mathbf{U}_{se})/2 \\ \Delta\mathbf{U} &= \Delta\mathbf{U} + \phi((\mathbf{F}_{sw} + 2\mathbf{F}_s + \mathbf{F}_{se})(y_{sw} - y_{se})/2 \\ &\quad +(\mathbf{G}_{sw} + 2\mathbf{G}_s + \mathbf{G}_{se})(x_{sw} - x_{se})/2) \end{aligned}$$

      where $\mathbf{F}$ and $\mathbf{G}$ are the flux vectors, computed directly from $\mathbf{U}$. Note: ϕ does not operate on the pressure terms.

· normalize the average cell values

$$\mathbf{U}_c = \mathbf{U}_c/8$$

* for coarse cells (use the values currently stored at the center node)

– compute the cell's geometric and time step parameters

$$\begin{aligned}
x_\xi &= (x_{ne} + x_{se} - x_{nw} - x_{sw})/2 \\
y_\xi &= (y_{ne} + y_{se} - y_{nw} - y_{sw})/2 \\
x_\eta &= (x_{ne} - x_{se} + x_{nw} - x_{sw})/2 \\
y_\eta &= (y_{ne} - y_{se} + y_{nw} - y_{sw})/2 \\
d_\xi &= \sqrt{x_\xi^2 + y_\xi^2} \\
d_\eta &= \sqrt{x_\eta^2 + y_\eta^2}
\end{aligned}$$

$$\begin{aligned}
a &= \sqrt{\gamma(\gamma-1)\left(\frac{\mathbf{U}_{4,c}}{\mathbf{U}_{1,c}} - \frac{\mathbf{U}_{2,c}^2 + \mathbf{U}_{3,c}^2}{2\mathbf{U}_{1,c}^2}\right)} \\
uc_\xi &= \left|\frac{\mathbf{U}_{2,c}}{\mathbf{U}_{1,c}} y_\xi - \frac{\mathbf{U}_{3,c}}{\mathbf{U}_{1,c}} x_\xi\right| + a d_\xi \\
uc_\eta &= \left|\frac{\mathbf{U}_{2,c}}{\mathbf{U}_{1,c}} y_\eta - \frac{\mathbf{U}_{3,c}}{\mathbf{U}_{1,c}} x_\eta\right| + a d_\eta \\
\frac{\Delta t}{A} &= \frac{\mathrm{CFL}}{\max(uc_\xi, uc_\eta)}
\end{aligned}$$

– for fine cells, modify the cell change

$$\Delta\mathbf{U} = \frac{\Delta t}{2A}\Delta\mathbf{U}$$

– distribute the changes to the corner nodes (sw is used as an example).

$$\psi = \begin{cases} 0.25 & \text{node in domain interior} \\ 0.50 & \text{node on solid boundary (side)} \\ 1.00 & \text{node at corner of domain} \end{cases}$$

$$d\mathbf{U}_{sw} = d\mathbf{U}_{sw} + \psi\frac{\Delta t}{2A}\left(\mathbf{I} + \frac{\Delta t}{A}\left(+ \frac{\partial \mathbf{F}}{\partial \mathbf{U}}(y_{se} - y_{nw}) - \frac{\partial \mathbf{G}}{\partial \mathbf{U}}(x_{se} - x_{nx})\right)\right)\Delta\mathbf{U}$$

- for fine cells
  - for each side node (if it exists) (s is used as an example).

$$\psi = \begin{cases} .25 & \text{node in domain interior} \\ .5 & \text{node on solid boundary (side)} \end{cases}$$

$$d\mathbf{U}_s = d\mathbf{U}_s + \psi\frac{\Delta t}{2A}\left(\mathbf{I} + \frac{\Delta t}{A}\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}y_\xi - \frac{\partial \mathbf{G}}{\partial \mathbf{U}}x_\xi\right)\right)\Delta\mathbf{U}$$

4.4.6 BCON

This subroutine modifies the corrections at boundaries such that the boundary conditions are satisfied, using the following procedure:

- for all boundary nodes
  - if the node is fully contained in the current grid level

$$d\mathbf{U} = \chi(d\mathbf{U})$$

where χ is a function which modifies the corrections such that the boundary conditions are satisfied. See section 3.2 for more details.

4.4.7 INTP

This subroutine interpolates the corrections form the current grid to all finer grid levels, using the procedure:

- for all levels from the current to the finest

- for all coarse cells
  - for all center nodes

$$dU_c = (dU_{sw} + dU_{se} + dU_{ne} + dU_{nw})/4$$

  - for all side nodes which are not just inside coarse (s is used as an example).

$$dU_s = (dU_{sw} + dU_{se})/2$$

- for all coarse cell on the next finer level
  - for all side nodes which are just inside coarse (s is used as an example)

$$dU_s = (dU_{sw} + dU_{se})/2$$

4.4.8 UPDT

This subroutine updates the dependent variables according to the rules:

- for all cells on the current and finer levels
  - for each of the cell's nodes (sw is used as an example)
    - if the node is not just inside fine
      - mark the node for updating
    - if the node is just inside coarse
      - mark the node for updating
- for all nodes which are marked

$$U = U + dU$$

4.4.9 BFIX

This subroutine ensures that the flow is aligned with the wall at all solid surface nodes. This procedure is only needed to eliminate the accumulated round-off errors. The procedure is given by:

- for all solid surface boundary nodes
  - rotate the velocity vector to align it with the wall

4.4.10 CONV

This subroutine computes convergence statistics:

- integrate the surface pressure coefficients along the solid surfaces and calculate the lift and drag coefficients
- for every node, compute

$$\Delta = |\mathbf{U}_2 - (\mathbf{U}_2)_{\text{from before the time step}}|$$

- compute the convergence statistics over the whole domain

  $||\Delta||_\infty$ (maximum)

  $||\Delta||_1$ (average)

  $||\Delta||_2$ (RMS)

Chapter 5

Grid and Data Structures

The previous chapter described the integration method used to solve the steady state Euler equations on arbitrarily shaped embedded regions. In this chapter, the data structure needed to describe those embedded meshes is discussed.

The chapter begins with a review of some of the types of data structures which could be employed. This is followed by a description of the data structure which is used in the current program. The remainder of the chapter is concerned with the various data structure handling procedures which are used to create, maintain, and change the data structure.

5.1 Data Structures

Traditionally, numerical simulations of partial differential equations are conducted on a regular net of computational nodes which are indexed (i, j). However recently, there has been a considerable number of studies which have employed unstructured grids for a variety of reasons.

On the one hand, unstructured triangular grids have been explored by Löhner[57,56] and Jameson[49] because of the geometric flexibility which they afford for problems with complicated geometric shapes, especially in three dimensions. On the other hand, an unstructured grid is used in this work because of the flexibility which it affords in the generation of adaptively generated embedded patches. In either case, a more complicated data structure than the traditional (i, j) indexing needs to be employed.

To simplify matters, the grids used in the current study are not completely unstructured, and hence are best described as *semi-structured*. The structure of the grid is specified through the following rules:

- All fine cells are quadrilaterals, although some may have additional nodes at the midpoints of one or more faces.
- All coarse cells must be composed of exactly four (children) cells on the next finer grid level.
- Parent and children cells must be oriented in similar compass directions.
- A cell must be oriented such that its south face
  - coincides with at most the north face of only one cell on the same grid level, and
  - does not coincide with either the east, south, or west face of any other cell on the same grid level

Similar restrictions apply to the other faces of the cell as well. This is only a restriction because of some assumptions made in the development of some of the data structure handling routines described below.

- Embedded regions must be wholly contained in the next coarser grid. This leads to the requirement that embedded mesh interfaces on different grid levels cannot be coincident.

Having such structure in the grid simplifies the data structure development.

Usab[84] has described in detail different possibilities for data structure designs for grids similar to the one described above. He identifies three possible choices:

- *cell-wise* — In this case, the structure of the grid is described on a cell-by-cell basis. Such a data structure offers the maximum flexibility in domain shape under the restrictions above. The major disadvantage of such a data structure is that it tends to require more memory for its implementation than the other possibilities. For a single grid level, each interior node is referenced by four cells and hence four *pointer* words are required for each node.

- *line-wise* — In this case, lines of cells are described by a single data structure entry, resulting in a structure which is not as flexible as the one above. The major advantage of this structure type is that is requires about half the storage as the one above; except for nodes at the corners of the lines, each node is referenced by two lines of cells one each side of it.

- *block-wise* — Here blocks of cells are described in a single data structure entry in a two-dimensional array. This structure, which is less flexible than either of the above two, has the advantage that it in general only requires one *pointer* for each node.

In choosing the data structure used in the current program, high flexibility was the prime consideration; minimizing storage was of lesser importance since the adaptively refined grids contain significantly fewer nodes than is typically required of globally refined grids.

Also, recall that all the operations in the flow integrator discussed in chapters 3 and 4 are either performed on a cell-by-cell basis or are performed on all computational nodes. Therefore in the current program, a cell-wise data structure is used because it is the most general one which also closely matches the data needs of the flow integrator.

The following sections contain a complete description of the actual implementation of the data structure used as well as a brief discussion of some of the data structure handling techniques which are required for grid adaptation.

5.2 Data Structure Employed

The data structure used here is very similar to the one developed by Usab[84]. It consists of five array types: nodes, cells, boundaries, levels, and bodies. The interconnections of four of these are shown in figure 5.1, where the arrows indicate entries in one array type which *point* to entries in another. The other array type (body) is not directly linked to the others through pointers. Notice that there are no embedded mesh interface arrays in the current data structure.

In the following sections, each of the array types is described. Each section begins with a statement of the purpose of the array type. This is then followed by a detailed description of the actual arrays and the specific information which they contain.

It should be noted that the data structure contains information about the current grid and solution, independent of the flow model and integration scheme being used. The only information which is required is the number

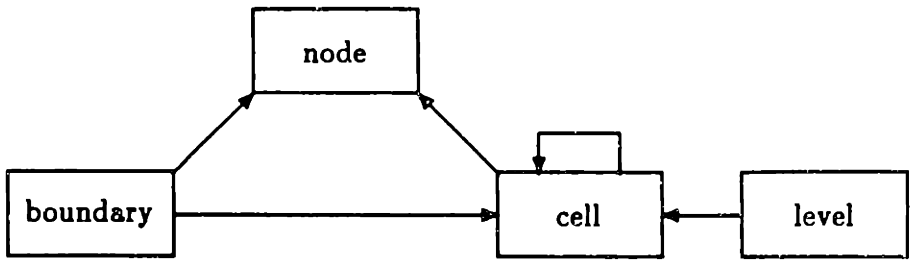


Figure 5.1: Interconnection of data structure arrays

of dependent variables and not their specific definitions.

5.2.1 Node arrays

The node arrays contain the independent and dependent variables at each computational node. The nodes are numbered arbitrarily and hence can be stored in the node array in any order.

The independent variables are stored in:

```
GEOMG2(j,i)      i=1, ..., number_of_nodes
                 j=1, ..., number_of_space_dimensions
```

where i denotes the node number and j denotes the specific independent variable. In two dimensions, the j=1 entry is the x-coordinate and j=2 denotes the y-coordinate.

The dependent variables are stored in a similar array:

```
DPENG2(j,i)      i=1, ..., number_of_nodes
                 j=1, ..., number_of_dependent_variables
```

where again i is the node number and j the dependent variable number. For the integrator described in chapters 3 and 4, there are 5 dependent variables (including the smoothing coefficient).

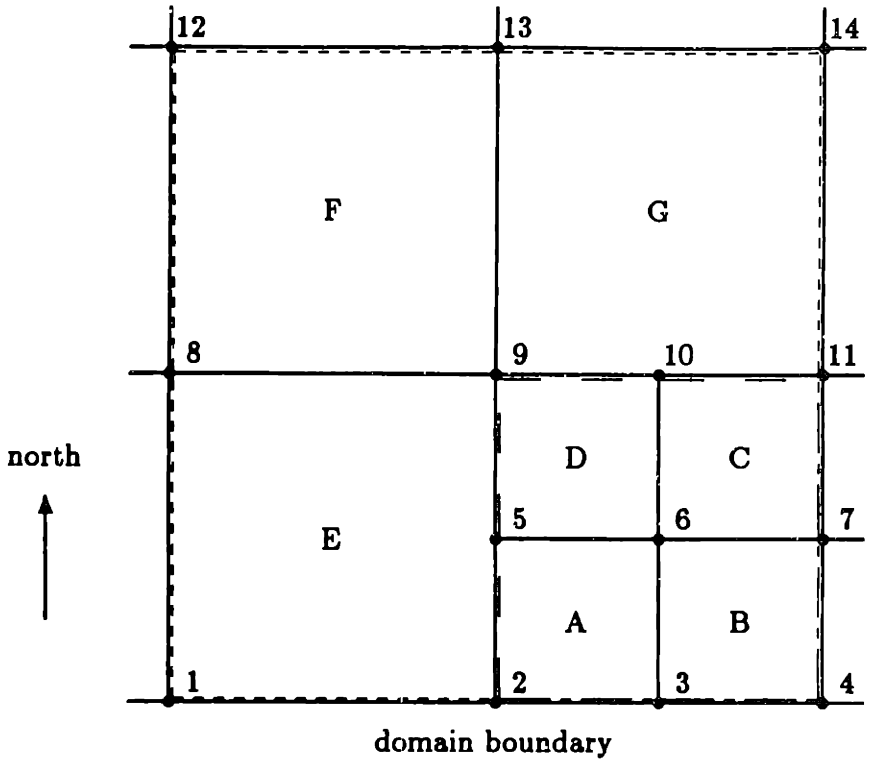


Figure 5.2: Sample grid

5.2.2 Cell arrays

The cell arrays contain connectivity information which links the nodes to form a grid. In addition, special information relating to the relative position of a cell to the boundaries of the domain and embedded mesh interfaces is contained in these arrays. Finally, the cell arrays contain pointers to neighboring cells so that the grid can be traversed.

A portion of a sample grid is shown in figure 5.2, where the cells are lettered and the nodes are numbered. Notice that the line at the bottom of the figure corresponds to a domain boundary. In addition to the fine cells, there are two coarse cells shown: cell ABCD is denoted by a dashed box and ABCDEFG is denoted by the dotted box.

In describing the relationships of cells to domain boundaries and embedded mesh interfaces, it is convenient to define the following four terms:

boundary — refers to cells which are on a domain boundary. For example cell A is on a boundary with its southwest and southeast nodes.

just outside — refers to cells which are just outside an embedded region. For example cell E is just outside an embedded region at its southeast and northeast nodes.

just inside (coarse) — refers to coarse cells which are just inside an embedded region. For example cell ABCD is just inside (coarse) at its southwest, northwest, and northeast nodes.

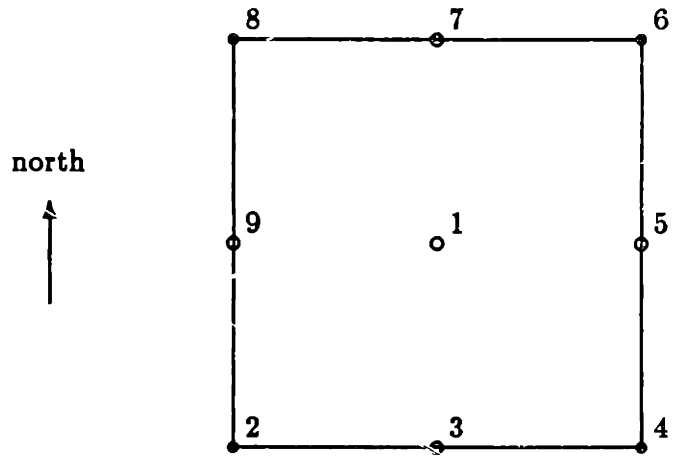
just inside (fine) — refers to fine cells which are just inside an embedded region. For example cell C is just inside (fine) at its northwest and northeast nodes.

The cells are stored such that cells on a given grid level are stored consecutively. Within a grid level, fine cells which are not just outside and which are not on a boundary are stored first, followed by the remaining fine cells, and finally the coarse cells. Within each group however, the cells are numbered arbitrarily.

The basic cell array is given by:

```
ICELG2(j,i)      i=1, ..., number_of_cells
                 j=1, ..., 10
```

The first nine rows (`j=1, ..., 9`) are pointers to entries in the node arrays, according to the numbering scheme given in figure 5.3. For example, if cell G has node 14 at its northeast corner, then `ICELG2(6,G)=14`. Any node which does not exist is entered as a 0; for example `ICELG2(3,E)=0` due to the absence of a south node in cell E.



Filled circles denote nodes which are always present
Open circles denote nodes which may or may not be present

Figure 5.3: Numbering scheme for nodes within a cell

The entries in the tenth row contain information about the cell's relationship to domain boundaries and embedded mesh interfaces. Using the definitions above, this information is encoded using the scheme shown in figure 5.4. This results in:

```
ICELG2(10,ABCDEFG)=b'... 0000 0000 0000 0011'
ICELG2(10,ABCD   )=b'... 0000 1101 0000 0011'
ICELG2(10,A      )=b'... 1001 0000 0000 0011'
ICELG2(10,B      )=b'... 0000 0000 0000 0011'
ICELG2(10,C      )=b'... 1100 0000 0000 0000'
ICELG2(10,D      )=b'... 1101 0000 0000 0000'
ICELG2(10,E      )=b'... 0000 0000 0110 0011'
ICELG2(10,F      )=b'... 0000 0000 0010 0000'
ICELG2(10,G      )=b'... 0000 0000 0011 0000'
```

where `b'... xxxx xxxx xxxx xxxx'` denotes a binary representation of the

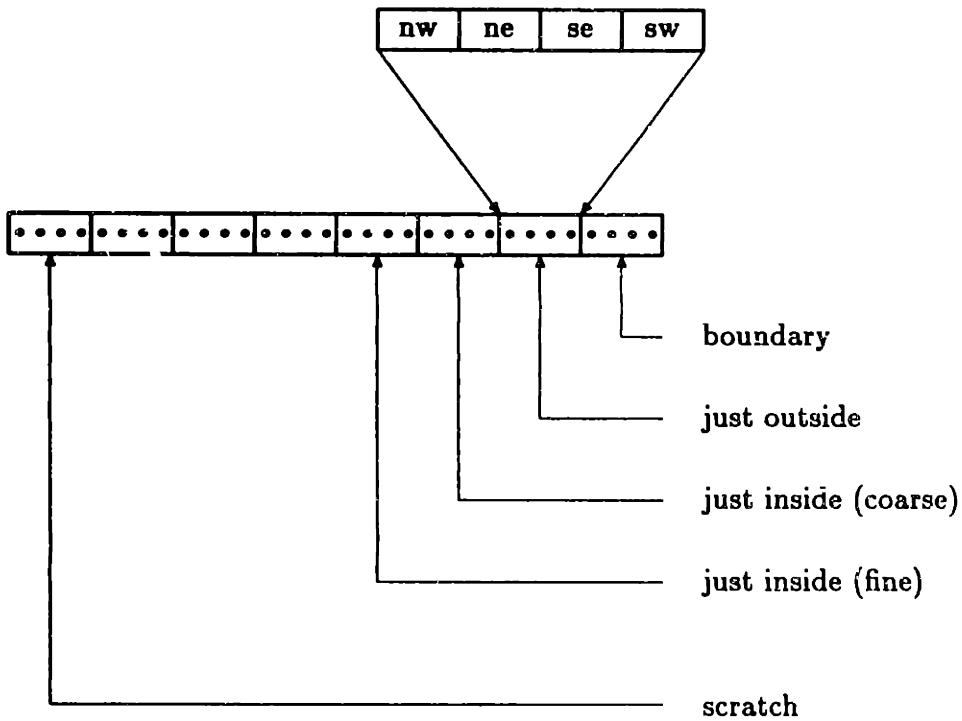


Figure 5.4: Encoding of special cell information

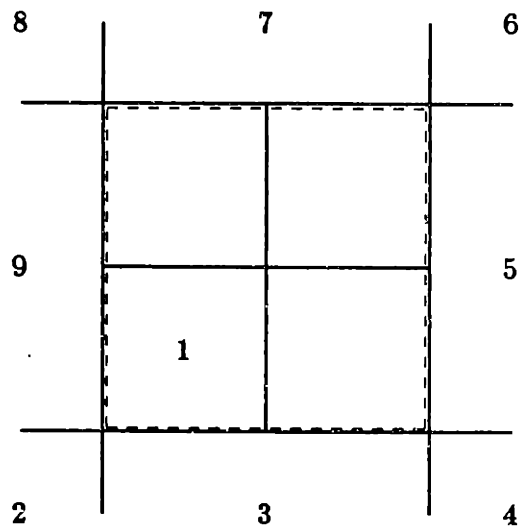


Figure 5.5: Definition of neighboring cells

rightmost 16 bits of each word.

The other cell array contains neighbor information in the form:

```
NBORG2(j,i)      i=1, ..., number_of_cells
                 j=1, ..., 9
```

Here the j index refers to the neighbors as defined in figure 5.5. As above, 0 is entered is a cell does not have a neighbor in the given relative location.

Since neighbor information is only required by some operations, the neighbor table is written out to a disk file when not needed and its storage locations are released for other purposes. (With the integrator described in chapters 3 and 4, the correction arrays occupy the released storage.)

5.2.3 Boundary arrays

In order to apply boundary conditions, one needs to know the nodes on the boundaries, the cells on the boundary, and the type of boundary.

In the current implementation, the boundary information is contained in two arrays, the first of which is given by:

```
IBNDG2(j,i)      i=1, ..., number_of_boundary_nodes
                 j=1, ..., 6
```

Here the j=1 entry points to the boundary node and j=2 and j=3 point to the cells. The fourth row, j=4, describes the boundary orientation, that is if the boundary is on the south side of the domain, etc. The j=5 entry describes the boundary condition type, which is an integer value which is passed to the flow integrator; its meaning is unknown to the data handling routines. The final row, j=6, is the level number of the coarser of the two adjoining cells.

For example, if boundary array entry m is for node 2, then the boundary table would contain the entries:

```
ICELG2(1,m)=2
ICELG2(2,m)=E
ICELG2(3,m)=A
ICELG2(4,m)=3 (indicating a southern boundary)
ICELG2(5,m)=bc_type (solver dependent)
ICELG2(6,m)=0
```

where it is assumed the cell E is on level 0 and cell A is on level 1.

In addition, boundary slope information and initial condition information is stored in:

```
BONDG2(j,i)      i=1, ..., number_of_boundary_nodes
                 j=1, ..., 9
```

The first four rows contain slope information for the segments for the boundary face just to the left and just to the right of the boundary node. This slope information is computed based upon splines as described in section 5.3.1.1.

The remaining five rows contain initial values of the dependent variables at the boundary nodes. (This information is convenient in some boundary condition formulations.)

5.2.4 Body arrays

Grid refinement results in the creation of new computational nodes at the faces of the divided cells. As described in section 5.3.3, these new nodes are generally placed at the midpoints of the faces of the divided cell. But as discussed in section 5.3.1, this is not adequate for boundary nodes. The body array, which contains information used to remedy this situation, has the form:

```
BODYG2(j,i)     i=1, ..., number_of_body_definition_points
                j=1, ..., 11
```

The first two rows of this array contain the independent variables which describe the boundaries. The following five rows contain the original values of the dependent variables at these points, and the final four rows contain slope information derived from spline fits of the nodes locations.

5.2.5 Level arrays

The final array of the data structure contains information as to which cells are associated with which multiple-grid levels. By definition, level 0 refers to the initial global grid level; coarser levels have negative level numbers and finer grids (embedded grids) have positive level numbers. The level pointer array takes the form:

```
ILVLG2(j,i)     i=-10, ..., 0, ..., 10
                j=1, ..., 6
```

where it is assumed that there are no more than 10 grid levels finer and coarser than the global level.

The first two rows point to the first and last fine cells on a given level which are not just outside an embedded region and which are not on boundaries. The third and fourth rows point to the first and last of the remaining fine cells. Finally, the last two rows point to the first and last coarse cells on the given level. If there are no cells in any category on any level, the pointer to the last cell is set to a value less than the first cell.

5.3 Procedures

The requirement that the grid structure, and hence data structure, be modified automatically during grid refinement results in a number of data structure handling routines which are described in the following sections. In all, there are more than 30 such routines in the MITOSIS program, only a few of which are discussed here.

5.3.1 Body definition

As mentioned above, the division of cells results in the creation of new nodes. Generally these nodes are placed at the midpoints of the cell's faces. Unfortunately this is not adequate for nodes along domain boundaries. For example, if a cylinder is described by eight nodes on the base grid (and hence is approximated by an octagon), refinement should place new nodes such that they describe the cylinder rather than more finely describe the octagon.

There are two techniques used here to ensure that the boundary closely approximates the original body definition. Hence, there needs to be some method of describing the boundaries at some level finer than the original global grid level.

This is done in the current program by actually defining the domain boundaries for a finer grid than the global grid. However, if a grid is to be created which is finer than the body definition, the new boundary node

locations are computed from spline fits which are described below.

5.3.1.1 Surface splines

The body coordinates which are defined above are spline fit with a pair of parametric cubic splines, where the parameter is taken to be the body point number. This then yields two functions

$$x = x(n) \quad \text{and} \quad y = y(n) \tag{5.1}$$

Following the notation of Dahlquist and Björck[28, pp132–134], define for the spline of $x = x(n)$ $0 \leq n \leq m$ the terms

$$\left.\begin{aligned} h_i &\equiv n_i - n_{i-1} \\ d_i &\equiv \tfrac{x_i - x_{i-1}}{h_i} \end{aligned}\right\} \quad i = 1, \ldots, m \tag{5.2}$$

The conditions that the zeroth, first, and second derivatives be continuous at the $m-1$ interior nodes leads to the series of equations

$$h_{i+1}k_{i-1} + 2(h_i + h_{i+1})k_i + h_i k_{i+1} = 3(h_i d_{i+1} + h_{i+1} d_i) \quad i = 1, \ldots, m-1 \tag{5.3}$$

where the k_i are the first derivatives dx/dn at the spline nodes. Boundary conditions are provided by assuming that the second derivative vanishes at the end nodes ($i = 0$ and $i = m$), giving

$$2k_0 + k_1 = 3d_1 \tag{5.4a}$$

$$k_{m-1} + 2k_m = 3d_m \tag{5.4b}$$

Thus the above forms a tridiagonal set of equations for k_i which is efficiently solved using the Thomas inversion technique[74, pp 345–349].

For the present problem, the above matrix is actually even simpler because the parameter against which the splines are computed is an integer. Thus $n_i - n_{i-1} \equiv 1$, resulting in the simpler forms

$$k_{i-1} + 4k_i + k_{i+1} = 3(y_{i+1} - y_{i-1}) \tag{5.5}$$

and

$$2k_0 + k_1 = 3(y_1 - y_0) \quad (5.6a)$$

$$k_{m-1} + 2k_m = 3(y_m - y_{m-1}) \quad (5.6b)$$

There are two sets of tridiagonal equations to be solved in the body representation in the current work, one each for the x- and y-coordinate (as a function of node number), but the two sets are completely independent.

Body geometries with slope discontinuities such as at sharp leading and trailing edges are treated by computing separate splines on each side of the discontinuity. For example, a bi-circular arc airfoil is composed of separate splines for the upper and lower surfaces. The implication here is that at each body node there may be two surface slopes, one for the segment to the left and another for the segment to the right. Hence at each body node, four spline coefficients are computed and stored (dx/dn and dy/dn for the left segment, and dx/dn and dy/dn for the right segment). At most nodes however, the two dx/dn's are identical as are the two dy/dn's.

5.3.2 Grid generation

The grid generation technique used in the current study is performed in a three-step process:

1. Define the node locations on the boundaries of the computational domain,

2. Generate an initial guess for the interior node locations using a simple algebraic generation technique, and

3. Smooth the locations of the interior grid nodes by using a partial differential equation method similar to that proposed by Thompson[81].

The assignment of boundary point distributions for each edge of the computational domain is accomplished in two ways in the current work. For

actual boundaries in the domain (that is actual bodies), the node distribution is taken directly from the distribution of points which is given by the case input file. Because these body points are fit with a cubic spline as described above, they should be smoothly distributed so as to avoid difficulties with the spline-fitting technique.

The body nodes which are on artificial domain boundaries (such as far-field boundaries) are generated automatically by the program based upon the input far-field boundary shape. For example in the O-type mesh examples, the outer boundary is a circle centered at the origin with a radius given by the user. The far-field body nodes are generated to be equally spaced along that outer boundary.

The initial locations of the grid points are generated using a simple algebraic technique. Here, straight lines are drawn between the southern and northern boundaries of the domain. The interior nodes are then distributed along these lines so as to approximate the spacings on the western and eastern boundaries. For O-type airfoil meshes, there are no given spacings on the western and eastern boundaries and so the nodes are spaced so that the spacing grows exponentially away from the airfoil.

The grid smoothing process which is employed is an elliptical grid generation scheme similar to that described by Thompson[81]. In this method, the interior grid points obey the Laplace's equations

$$\xi_{xx} + \xi_{yy} = 0 \tag{5.7a}$$

$$\eta_{xx} + \eta_{yy} = 0 \tag{5.7b}$$

where (ξ, η) are the coordinates of the nodes in computational space and (x, y) are the node locations in physical coordinates. Given the boundary node distributions, these equations form a closed set and can be solved.

The node locations are defined by the intersections of contours of ξ and η in space. Typically, evenly spaced contours are used to generate the grid.

Grids generated with (5.7) have the nice property that grid lines of the same family cannot cross due to the maximum principle for Laplace's equation.

Unfortunately solving (5.7) for ξ and η as a function of x and y requires the use of a computational grid in (x, y) space. Since such a grid is the object of the grid generation process, solving (5.7) is clearly not reasonable. To circumvent this problem, Thompson suggested inverting equations (5.7) to yield

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = 0 \tag{5.8a}$$

$$\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = 0 \tag{5.8b}$$

where

$$\alpha = x_\eta^2 + y_\eta^2 \tag{5.9a}$$

$$\beta = x_\xi x_\eta + y_\xi y_\eta \tag{5.9b}$$

$$\gamma = x_\xi^2 + y_\xi^2 \tag{5.9c}$$

Equation (5.8) can now be solved on a uniformly spaced (ξ, η) grid with Dirichlet boundary conditions given by the specified boundary node distributions.

Applying the above to an O-type domain about a NACA-0012 airfoil results in the grid shown in figure 5.6, which is a 33×17 grid extending to ten chord lengths from the airfoil. The computational cells are nearly orthogonal and their size varies smoothly throughout the domain.

The same technique can be applied to the generation of an H-type mesh about the same airfoil as shown in figure 5.7. Here the grid contains 33×33 nodes with an outer boundary which extends ten chord lengths from the airfoil in all directions. Again the cells are nearly orthogonal and their sizes vary smoothly. However this mesh is clearly unacceptable due to the sparseness of the grid in the vicinity of the airfoil. This sparseness is a direct consequence of the properties of Laplace's equation (5.7).

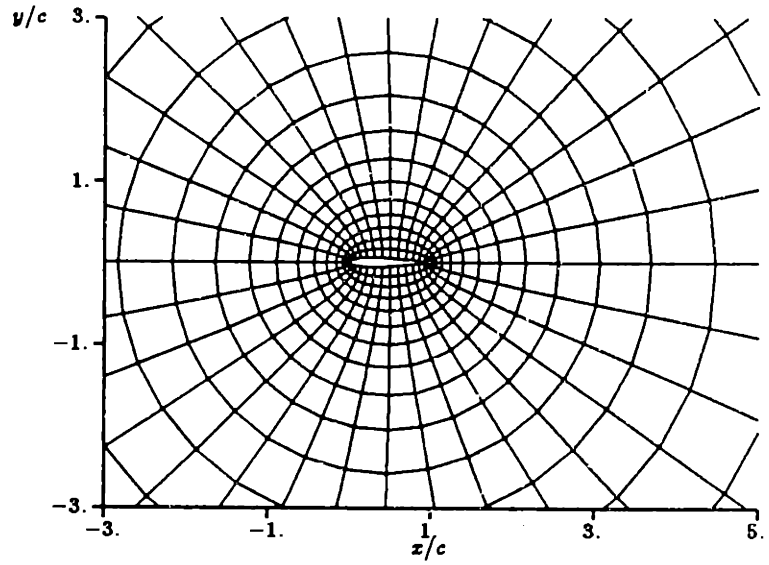
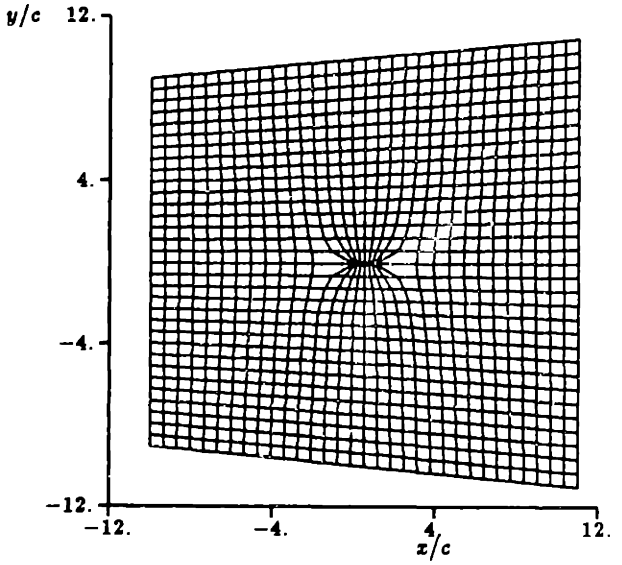


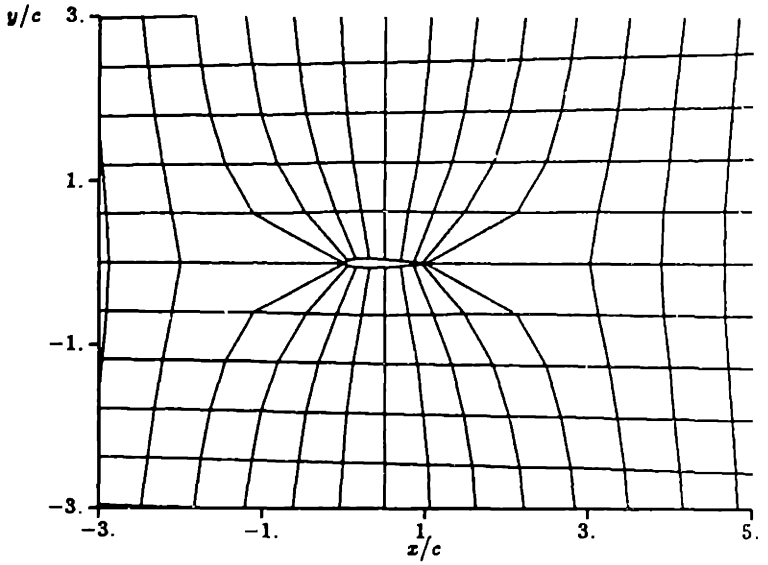
Figure 5.6: O-type mesh in the vicinity of a NACA-0012 airfoil

In order to get a more reasonable spacing near the airfoil, there are two basic approaches. In the first, Laplace's equation (5.7) is converted into Poisson's equation by adding the forcing functions P and Q to the right hand sides. These functions are tailored to yield the desired resolution and grid shape in the vicinity of the airfoil as discussed in [77]. Typically, this is accomplished by determining values for P and Q at each surface node which yield the desired grid there. P and Q are then decayed exponentially out into the field.

There are however two main problems with this approach. Since P and Q are not in general positive, the maximum principle which guarantees smooth, one-to-one grids does not hold. Therefore, there is no assurance that the resulting grid will not fold over onto itself. The other problem is a more practical one, namely that the computation of P and Q have to be severely under-relaxed in order to achieve convergence of the grid generation. This



a: entire domain



b: near airfoil

Figure 5.7: Bad H-type mesh about a NACA-0012 airfoil

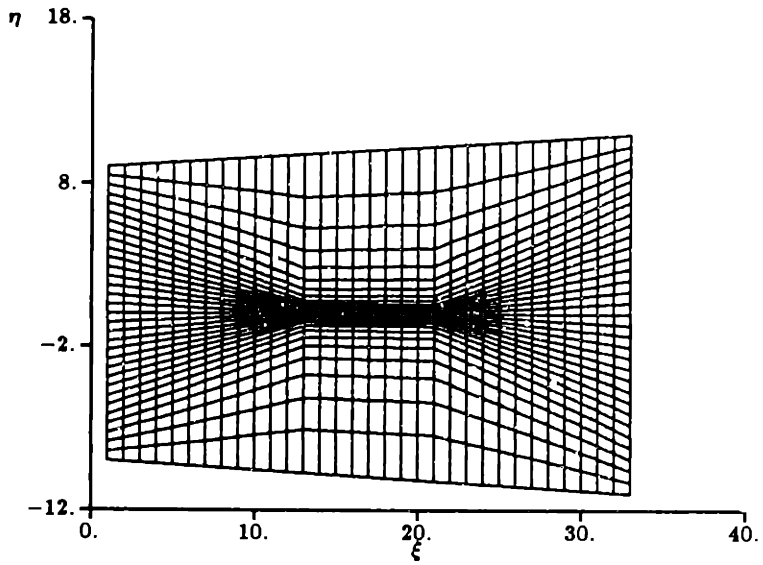


Figure 5.8: Desirable grid about zero-thickness, non-cambered airfoil

turns out to significantly increase the time required to set up such a grid.

The other method is conceptually much simpler and in fact easier to implement. Instead of generating the grid by the intersection of contours of evenly spaced ξ and η, one can simply choose other points in (ξ, η) space to specify the grid. This technique was first explored by Drela and Giles[30].

Conceptually this technique can be explained in the following manner. Initially consider the H-type mesh around an isolated, zero-thickness, non-cambered airfoil. Solving (5.8) and then using contours with constant increments in ξ and η results in a rectangular grid of lines very similar to that shown in figure 5.7. As mentioned above, the very large cells in the neighborhood of the airfoil are unacceptable.

A more reasonable grid for this case is shown in figure 5.8, where there is considerable clustering in the vicinity of the airfoil. Thus an appropriate grid for the real airfoil can be created by beginning with the grid in figure 5.8

and then "inflating" the slit airfoil to its correct thickness and camber.

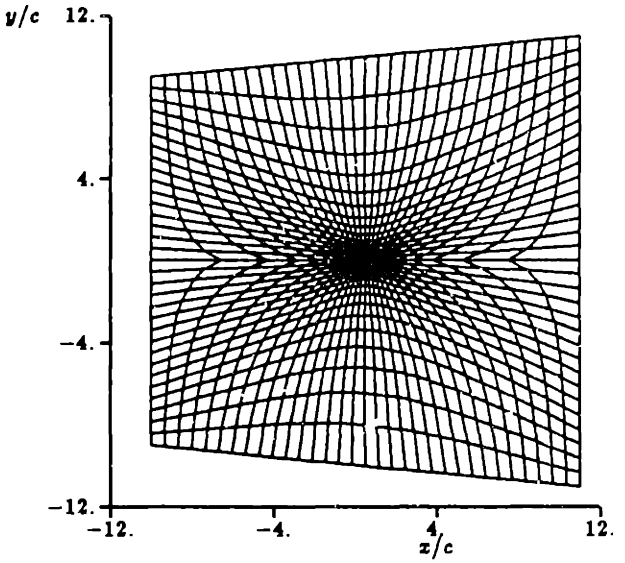
This is accomplished by considering the grid point locations in figure 5.8 as the (ξ, η) coordinates to be used when solving equation (5.8). The grid which results from such a process is shown in figure 5.9, where the grid is much more reasonable in the vicinity of the airfoil. One slightly negative side effect of this procedure is that because (5.8) is no longer solved on a uniform (ξ, η) grid, the grid generation technique is only first order accurate; in practice this is not important since the flow integration technique can handle minor irregularities in the grid.

In practice, the solution of (5.8) is accomplished using a point-Jacobi relaxation scheme. This rather crude technique is necessary in the current work because of the unstructured nature of the grids. This obviates the use of an implicit technique such as successive line over-relaxation (SLOR) or alternating direction implicit (ADI). In order to circumvent the slowness of point-Jacobi, the present scheme employs a combination of coarse/fine sequencing and partial-over-relaxation to decrease the spectral radius on each grid.

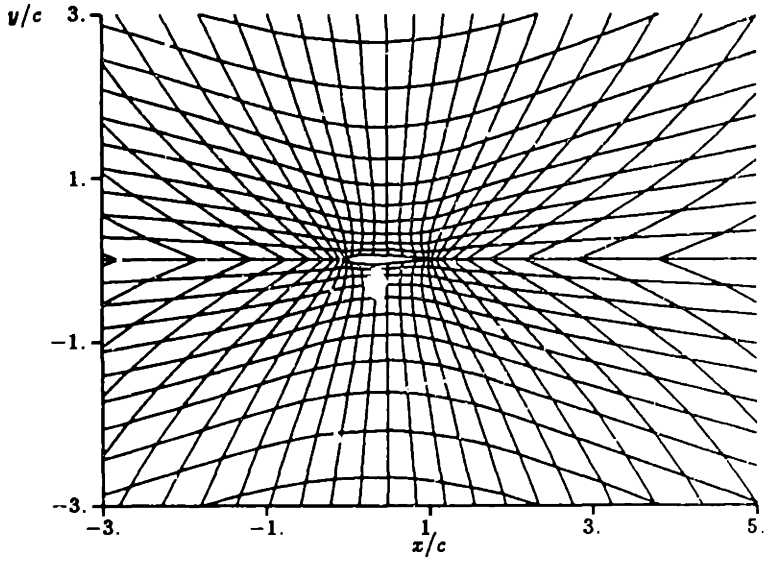
5.3.3 Cell division

The cell division process is a complicated one, carried out in a series of steps. Starting with the number of the cell to divide, the process is:

- Check to ensure that there is enough room in the data structure for five new nodes, four new cells, and two new boundary conditions.
- Determine the node numbers for the nodes associated with the cell to be divided.
- Abort the division process if:
  - the cell is already divided, or



a: entire domain



b: near airfoil

Figure 5.9: Good H-type mesh about a NACA-0012 airfoil

- the cell has been marked for deletion, or
- the cell is just inside an embedded region

- Make room in the cell table at the appropriate level if currently there is insufficient room for the four new cells.
- Determine the cell numbers of the neighbors of the cells to be divided as well as the neighbors' children.
- Create a node at the centroid of the cell with dependent variable values equal to the average of the values at the corner nodes.
- Create nodes at the midpoints of the faces if such nodes do not currently exist (they could exist if the neighbor was previously divided). The dependent variable values are the averages of the adjacent node values.
- Change the pointers in the cell which is being divided to point to the newly created nodes.
- Create the four new cells.
- Set up the pointers in the four new cells to point to the appropriate nodes.
- Inform the neighbor cells of the new nodes.
- For cells which are along a boundary:
  - Move the newly created boundary node to lie on the boundary as discussed above.
  - Create a new boundary array entry for the boundary node and set up the appropriate pointers.

- Change the neighbor pointers in all the neighboring cells to point to the newly created cells.

- Update all the information contained in `ICELG2(10,n)` for the cell which is divided, the new cells, and all their neighbors pertaining to a cell's location relative to embedded mesh interfaces. The logic to do this is rather complicated and the reader should refer to the listing of subroutine `G2DIVD` in appendix B.

5.3.4 Cell fusion

As for the cell division process, fusion of a cell also is complicated. Again the input is the cell to fuse (that is the number of the coarse cell which becomes a fine cell as a result of the fusion). The process is given by:

- Determine the node numbers for nodes associated with the cell to be divided.

- Abort the division process if:

  - the cell is not divided, or

  - the cell has been marked for deletion

- Determine the cell numbers of the cells to be deleted (the children of the cell to be fused), the neighbors of the cells to be divided as well as the neighbors' children.

- Abort the division process if:

  - any of the children cells are themselves divided, or

  - neighbors of any of the children cells are divided

- Mark the center node for deletion

- Mark side nodes for deletion if they are not needed by the neighboring cells of the cell to be fused.

- Mark for deletion the four children cell of the cell which is being fused.

- Change the pointers in the cell which is being fused so as not to point to the deleted nodes.

- Inform the neighbor cells of the nodes which have been deleted.

- For cells which are along a boundary:

  - Mark for deletion the boundary array entry for any boundary nodes which have been deleted.

  - Change boundary array entries which used to point to a deleted cell.

- Change the neighbor pointers in all the neighboring cells to point to the newly deleted cells.

- Update all the information contained in `ICELG2(10,n)` for the cell which is fused, the deleted cells, and all their neighbors pertaining to a cell's location relative to embedded mesh interfaces. Again the logic to do this is rather complicated and the reader should refer to the listing of subroutine `G2DIVD` in appendix B.

Notice that in the above discussion, nothing was actually deleted — entries were only *marked for delete*. This is done because the data structure realignment which is required for deleting an entry can be substantial. By marking for delete, a garbage collection process can be performed once after all cells are fused, thereby saving considerable time in the realignments.

5.3.5 Miscellaneous procedures

This section contains a description of two data structure procedures which are associated with the shape and size of the embedded regions rather than with the pointers, etc. The first of these, void and island detection, is concerned with ensuring that the embedded regions are in some sense continuous and that the edge of the embedded regions are reasonably smooth. The second, buffer zone creation, is used to enlarge embedded regions.

In both of these procedures, the data structure is scanned to determine the structure (shape) of the embedded regions, using mainly the information contained in `ICELG2(10,n)`. Any necessary adjustments are made using the division and fusion procedures described above.

Void and island detection

In general it is desirable to have the embedded regions be "smoothly" shaped, that is the edge of the embedded region should not have severe indentations. Such "smooth" embedded regions tend to have shorter embedded mesh interfaces then do "un-smooth" embedded regions. This is important in the current work since it was shown in chapter 4 that errors are incurred at the interfaces.

One method of determining the smoothness of an embedded region is to search for ***voids*** and ***islands*** in the embedded regions. Voids are defined as undivided cells which are surrounded by either embedded regions and/or boundaries; islands are divided cells which are surrounded by either undivided cells and/or boundaries. Elimination of such voids and islands, which are shown in figure 5.10, can thus be used to "smooth" the shape of embedded regions.

The detection and elimination of islands and voids in the current work is a two-step process. In the first, each cell is examined to determine if it is a void or an island; if so the cell is marked for either division or fusion. The exact rules which are applied are:

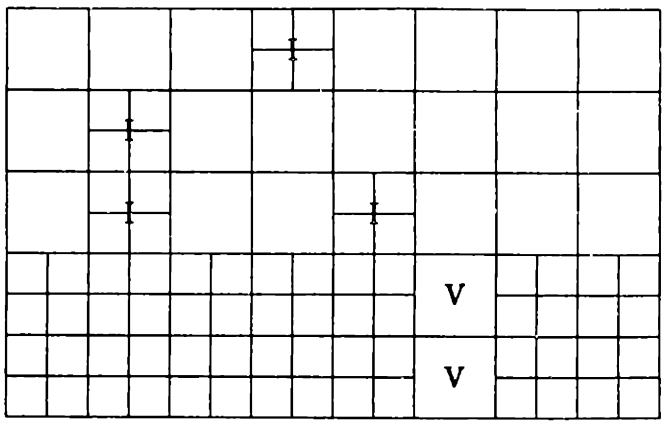


Figure 5.10: Portion of a grid with islands marked with I and voids marked with V

- if a fine cell has all four corner nodes lying either just outside an embedded region and/or on a boundary, then the cell is a void and it is marked for division.

- if a coarse cell has all four corner nodes either lying just inside an embedded region and/or on a boundary, then the cell is an island and it is marked for fusion. (Note that the term island also applies to a peninsula of width one.)

In the second pass, the actual cell divisions and fusions are performed using the procedures describe above.

Buffer zone creation

As will be discussed in chapter 8, occasionally adaptation yields embedded regions which are slightly too small for the physics of the problem and thus buffer zones need to be created around the embedded regions. A *buffer zone* is defined as a single line of cells which surround the current embedded regions.

The creation of such a buffer zone is performed in a two-step process.

As above, in the first step all fine cells are examined to determine if they are just outside an embedded region; those that are are marked for division. In the second pass the marked cells are divided using the algorithm described in section 5.3.3.

Chapter 6

Feature Detection and Grid Refinement

Previous chapters have dealt with the data structure necessary for describing irregularly shaped embedded grids and with the integration of the governing equations on those grids. This chapter describes the procedure used to sense flow features, that is regions of "rapid changes", in the evolving flow field and to generate the embedded regions automatically to encompass the detected features.

The chapter begins with a discussion of the types of features (geometric and flow) which occur, and their respective causes and properties. Following that, a feature detection strategy is developed for those features whose existence and location are not known *a priori*. The feature detector is composed of two main components, calculation of an appropriate refinement parameter that defines the changes occurring throughout the field and a threshold selection algorithm that distinguishes between the relative importance of the changes, both of which are described. The chapter concludes with a discussion of the algorithm that is used to create and/or modify the embedded regions adaptively.

The basic concepts employed in the feature detection algorithm developed here are applicable to any field description, whether numerically or

experimentally generated. The only restriction on their use is that the user be able to define an appropriate refinement parameter for the expected feature types.

6.1 Features

Features are defined as regions of the flow domain in which the scale of the relevant physics is smaller than the scales which characterize the flow domain elsewhere. These features can arise either directly in response to a singularity in the geometry of the domain, indirectly due to a mathematical singularity of the flow model, or a combination of the two. It should be noted that "singularity" is not to be taken in its strict mathematical sense.

Those features which are caused directly by geometric singularities are called *geometric features*. The geometric singularity may manifest itself as a slope discontinuity such as a sharp edge (for example at an airfoil leading or trailing edge) or as a corner (for example at the intersection of a cooling passage and a turbine airfoil surface).

Other geometric features can be generated by the computational grid which is imposed. For example, a blunt leading edge which is treated with an H-type mesh results in a leading edge singularity. In this case, the singularity is due to a slope discontinuity in one family of grid lines, even though the domain geometry is smooth and continuous.

In either case, the locations of all geometric features in a domain remain fixed and can be determined *a priori*.

A second type of feature is called a *flow feature* since it arises from mathematical singularities in the flow equations. Examples of such flow features which arise from the Euler equations include shock waves, expansion fans, slip lines, vortex cores, and vortex sheets. More detailed flow equations admit still more types of flow features — the Navier Stokes equations result in boundary and shear layers while the finite rate chemistry equations allow

flame fronts and other types of reaction zones.

Some flow features are attached to geometric singularities (such as an oblique shock emanating from a corner in supersonic flow), but others arise in smooth geometric regions (for example normal shocks standing on the upper surface of a transonic airfoil). Hence, in general the locations (and even existence) of flow features are not known *a priori* and must be sensed. The following sections describes a procedure which has been developed to detect and react to such flow features.

6.2 Flow Feature Detection

The flow feature detection algorithm developed here is based on the computation of a suitable *refinement parameter* at every node in the domain. The refinement parameter must be defined such that it is significantly different at flow features than elsewhere in the domain. The magnitude of refinement parameter which separates flow features from the background is determined automatically by a *threshold selection* algorithm. The following sections contain descriptions of refinement parameters (how to select and compute them) and the threshold selection algorithm.

6.2.1 Refinement parameter

In order to automatically detect flow features in a given domain, one must have an *a priori* notion of the types of flow features which may occur. For example, in steady inviscid transonic flow calculations about airfoil-like bodies, one might expect shock waves, slip lines, and expansion fans (but not boundary layers) to be present. A good refinement parameter is therefore one whose value is larger at the aforementioned features than in the smooth regions of the domain.

For the Euler equations there are many values which might be sensed for this purpose. For example, one may use any field quantity or one of

its derivatives; the field quantity can be a conservation variable such as the density or energy, derived quantities such as the entropy or vorticity, or a non-physical parameter such as the smoothing coefficient derived in Chapter 3.

Once the appropriate field quantity and order of the derivative are chosen, there are a number of methods by which the derivative can be measured in two or more dimensions. For example, a first derivative of say ϕ can be the magnitude of the gradient

$$|\nabla\phi| = \sqrt{\phi_x^2 + \phi_y^2} \tag{6.1}$$

or alternatively the derivatives in either or both the computational coordinate directions (ϕ_x or ϕ_y) as in Kallinderis[50]. The former is chosen here since it is independent of the arbitrarily aligned local coordinate directions. Similarly for a second derivative, the Laplacian

$$\nabla^2\phi = \phi_{xx} + \phi_{yy} \tag{6.2}$$

is used due to its invariance to local coordinate inclinations.

Once the method of computing the appropriate derivative is determined, one must choose whether the derivative is computed in physical space (a divided difference such as $\frac{\phi_1-\phi_2}{x_1-x_2}$) or in computational space (an undivided difference such as $\phi_1 - \phi_2$); the appropriate choice depends on the type of features which are expected.

One of the feature types which occurs frequently in transonic flow calculations is shocks, which the numerical scheme models as a discontinuity of fixed strength (for straight shocks), but with a width which is directly proportional to the local grid spacing. Hence, the undivided difference across a shock remains constant with continuing refinement whereas the divided difference increases due to the spatial difference in the denominator.

The difficulty with using divided differences for detecting discontinuities is readily apparent by considering a domain which has two shocks of ap-

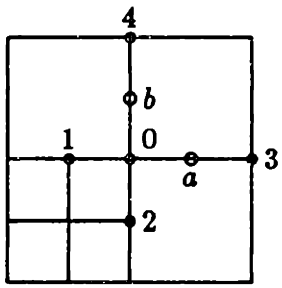


Figure 6.1: Node numbering for computation of derivatives at node 0

proximately equal strengths. Suppose that in the first adaptation cycle one of the shocks is refined whereas the other is not. On the second adaptation cycle, the previously refined shock will have divided differences which are twice as large as at the unrefined shock and hence will command even more adaptation; this is in opposition of the desired behavior which should tend to treat equal-strength shocks equally.

The computation of undivided differences is complicated by the presence of embedded regions. For example, consider the computation of the first difference on the mesh shown in figure 6.1. The gradient calculation at node 0 must be computed based upon values stored at the nodes 1 through 4 (which are denoted by the filled circles). To simplify the calculation, pseudo nodes are created as given by the open circles; the values there are determined by averaging the values at the nodes at either end of the appropriate face, or

$$\phi_a = \frac{\phi_0 + \phi_3}{2} \tag{6.3a}$$

$$\phi_b = \frac{\phi_0 + \phi_4}{2} \tag{6.3b}$$

With these pseudo-nodes, the gradient can be calculated directly by

$$\frac{\partial \phi}{\partial \xi} = \phi_a - \phi_1 \tag{6.4a}$$

$$\frac{\partial \phi}{\partial \eta} = \phi_b - \phi_2 \tag{6.4b}$$

$$\tilde{\nabla}\phi = \sqrt{\left(\frac{\partial \phi}{\partial \xi}\right)^2 + \left(\frac{\partial \phi}{\partial \eta}\right)^2} \tag{6.4c}$$

where $\tilde{\nabla}\phi$ denotes the magnitude of the undivided gradient. This difference in accumulated by a cell-wise sweep of the entire computational domain. Special care must be taken at boundaries to ensure that the magnitude of $\tilde{\nabla}\phi$ is comparable to the magnitude just away from the surfaces.

Similar computations are performed for the second difference

$$\frac{\partial^2 \phi}{\partial \xi^2} = \phi_a - 2\phi_0 + \phi_1 \tag{6.5a}$$

$$\frac{\partial^2 \phi}{\partial \eta^2} = \phi_b - 2\phi_0 + \phi_2 \tag{6.5b}$$

$$\tilde{\nabla}^2\phi = \left|\frac{\partial^2 \phi}{\partial \xi^2} + \frac{\partial^2 \phi}{\partial \eta^2}\right| \tag{6.5c}$$

Finally, the refinement parameter, $\mathcal{R}$, is defined as the ratio of the appropriate derivative from above (equation (6.4) or (6.5)) to the average derivative over the entire domain.

The mechanics of computing refinement parameters are relatively simple when compared to a determination of which refinement parameters are suitable for inviscid, transonic flows about airfoil-like bodies. One procedure (and the method used here) is to simply compute various refinement parameters on an actual flow domain. Examination of the behavior of the various parameters provides a basis for an appropriate choice.

The refinement parameters which were tested here consist of the density (ρ), pressure (p), velocity magnitude (q), and the total pressure (p_o) as well as their first and second undivided differences.

The flow domain for that study is from the basic test case described in detail in Chapter 8 (an RAE-2822 airfoil at $M_\infty = 0.75$ and $\alpha_\infty = 3.0°$). The refinement parameters are computed based upon a uniform 64×32 grid solution.

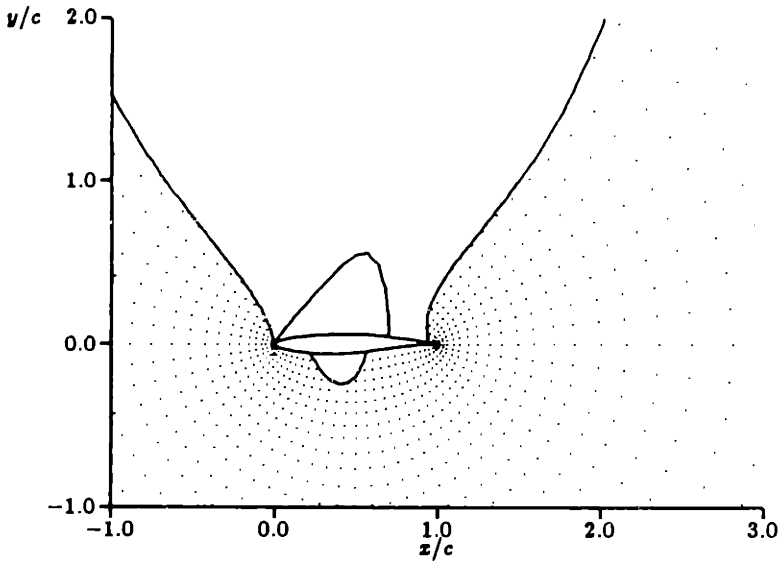


Figure 6.2: Contours of $\mathcal{R} = \rho$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

Each of the figures 6.2 through 6.13 contains contours of one of the appropriate refinement parameters in the vicinity of the airfoil. The contour interval is fixed at $\Delta\mathcal{R} = 0.25$ with a maximum contour level of $\mathcal{R} = 5.0$. In addition, the computational nodes which have a refinement parameter greater than the average ($\mathcal{R} \geq 1.0$) are denoted by dots. From such figures one can determine the relative importance that the particular choice of $\mathcal{R}$ places on each feature type.

The results of these plots are summarized in Table 6.1. If shock wakes are an expected dominant feature, then the use of p_o or one of its derivatives should be used whereas if stagnation zones and expansion fans are expected, then a more appropriate choice is either the first or second difference of the ρ, p, or q.

These results are valid only for the type of problem specified (inviscid transonic flow); for other flows (such as incompressible flows), other refine-

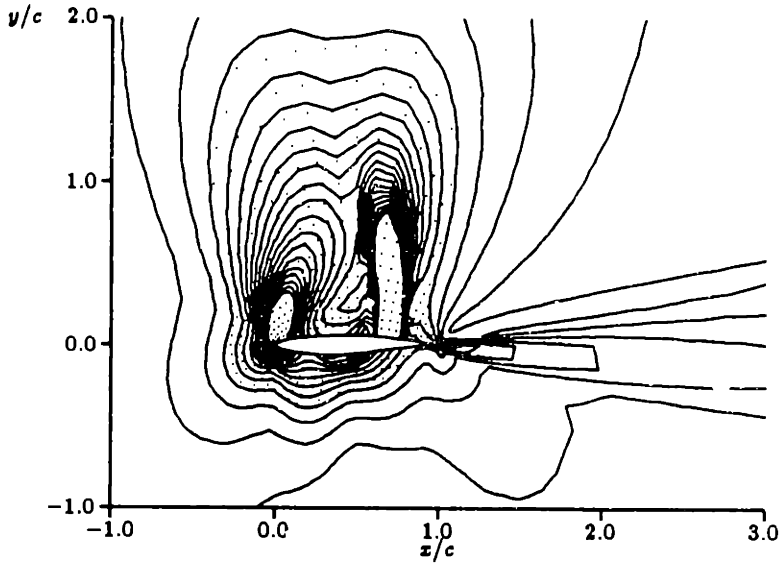


Figure 6.3: Contours of $\mathcal{R} = \tilde{\nabla}\rho$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

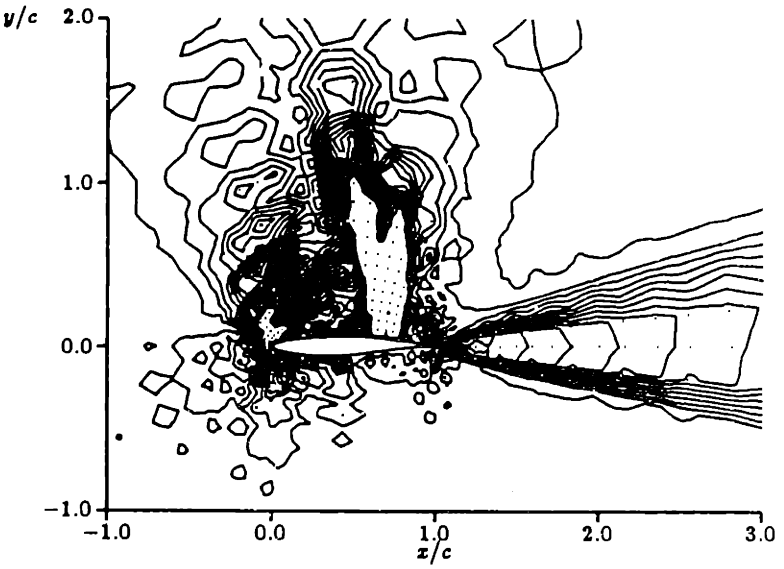


Figure 6.4: Contours of $\mathcal{R} = \tilde{\nabla}^2\rho$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

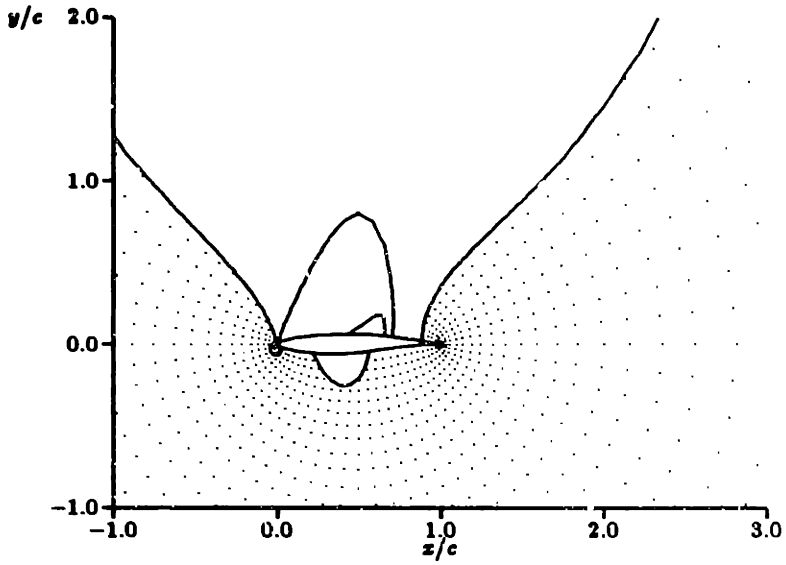


Figure 6.5: Contours of $\mathcal{R} = p$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are **dotted**

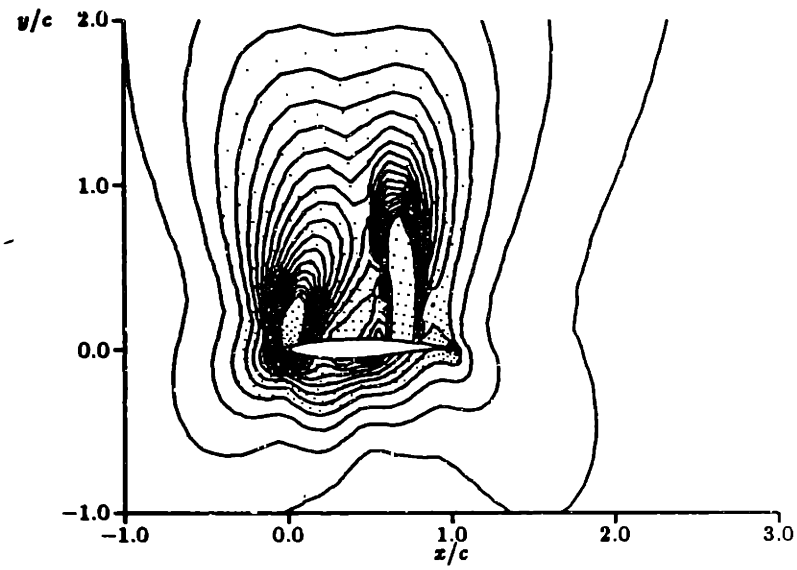


Figure 6.6: Contours of $\mathcal{R} = \tilde{\nabla}p$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are **dotted**

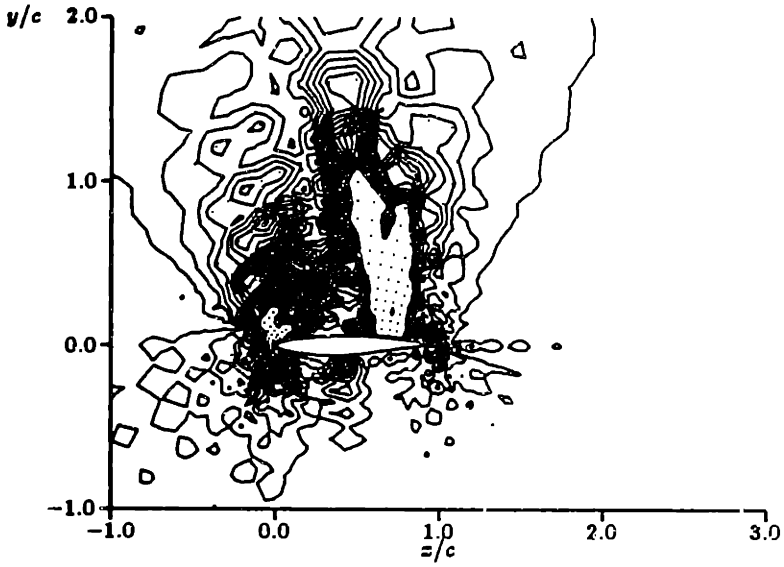


Figure 6.7: Contours of $\mathcal{R} = \tilde{\nabla}^2 p$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

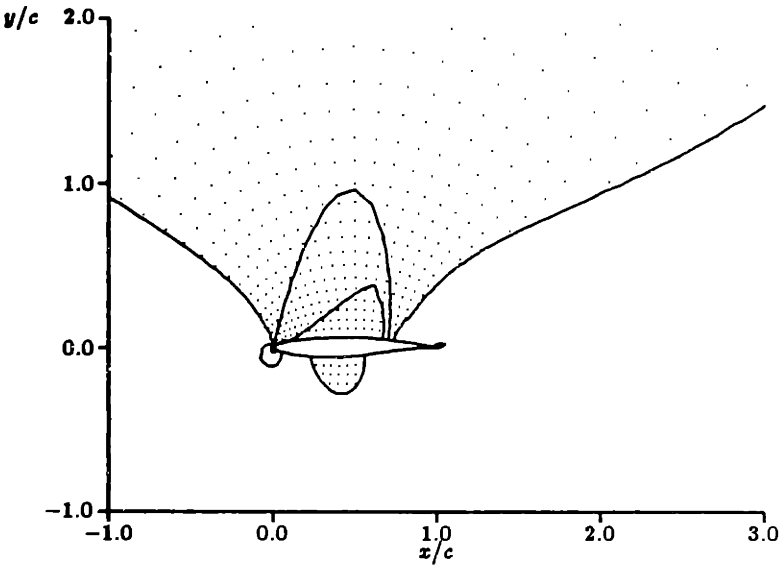


Figure 6.8: Contours of $\mathcal{R} = q$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

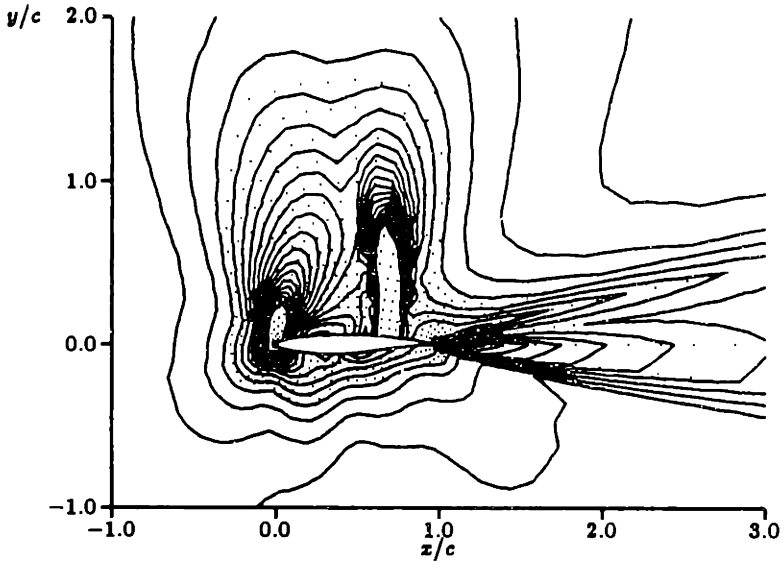


Figure 6.9: Contours of $\mathcal{R} = \tilde{\nabla} q$ with $\Delta \mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

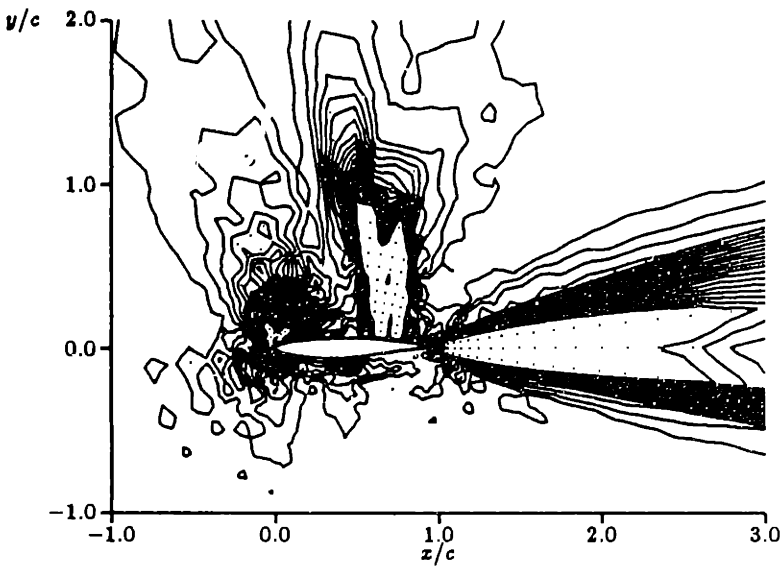


Figure 6.10: Contours of $\mathcal{R} = \tilde{\nabla}^2 q$ with $\Delta \mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

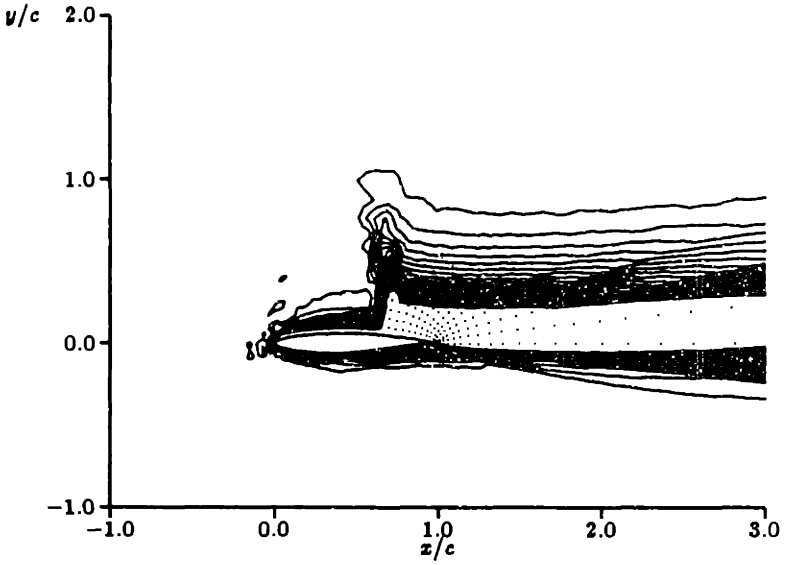


Figure 6.11: Contours of $\mathcal{R} = p_o$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

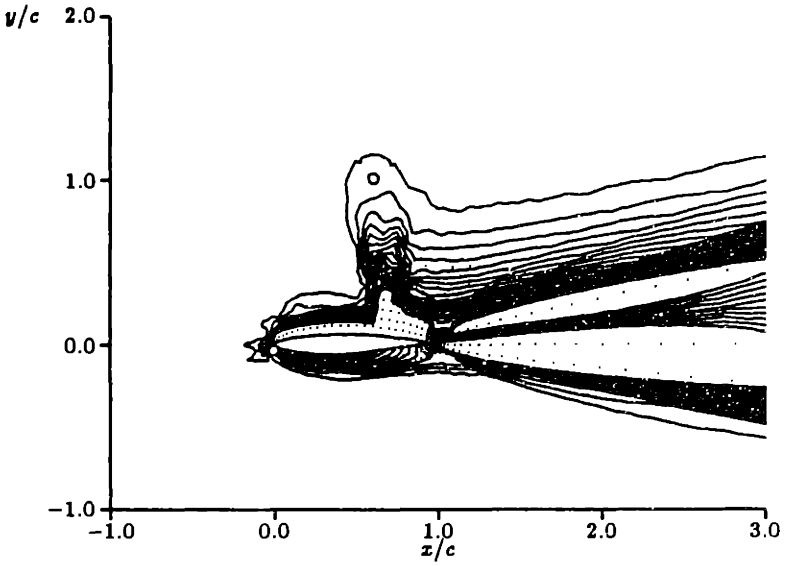


Figure 6.12: Contours of $\mathcal{R} = \tilde{\nabla} p_o$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

$\mathcal{R}$	Feature type				
	shock wave	slip line	expansion fan	stagnation zone	shock wake
$\|\rho\|$	0	0	0	1	0
$\tilde{\nabla}\rho$	2	1	2	1	0
$\tilde{\nabla}^2\rho$	2	2	1	1	0
$\|p\|$	0	0	0	2	0
$\tilde{\nabla}p$	2	0	2	2	0
$\tilde{\nabla}^2p$	2	2	1	2	0
$\|q\|$	0	0	0	0	0
$\tilde{\nabla}q$	2	2	2	2	0
$\tilde{\nabla}^2q$	2	2	1	2	0
$\|p_o\|$	1	0	0	0	2
$\tilde{\nabla}p_o$	2	2	0	0	2
$\tilde{\nabla}^2p_o$	2	2	0	0	2

Note: 0 $\Rightarrow$ the feature is not detected

1 $\Rightarrow$ the feature is somewhat detected

2 $\Rightarrow$ the feature is well detected

Table 6.1: Expected effectiveness of various refinement parameters for inviscid, transonic flows over airfoil-like bodies

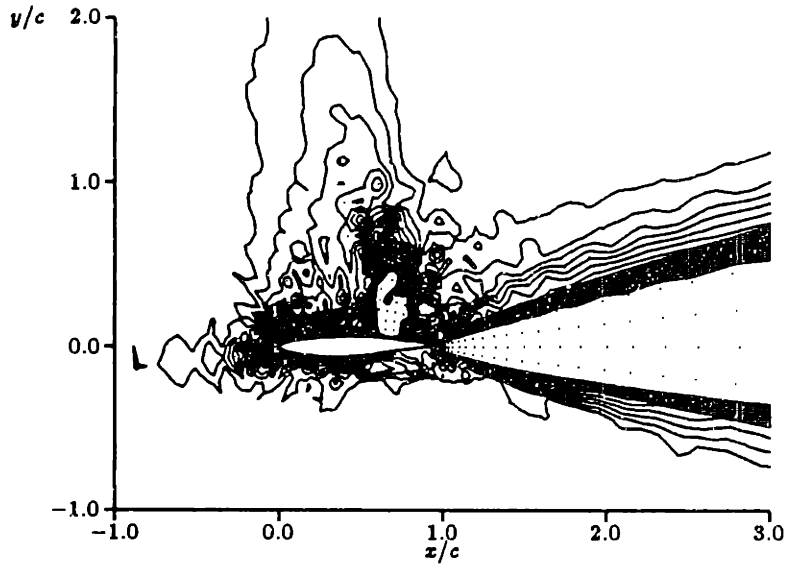


Figure 6.13: Contours of $\mathcal{R} = \tilde{\nabla}^2 p_o$ with $\Delta\mathcal{R} = 0.25$. Nodes with $\mathcal{R} \geq 1.0$ are dotted

ment parameters may need to be defined.

In order to circumvent the difficulties associated with choosing a suitable refinement parameter, others (most notably Berger[11]), have directly measured the truncation error of the integration scheme and used it as a refinement parameter. At first this seems to be the best refinement parameter that one could use since it is actually the truncation error which one attempts to reduce through adaptation. However, the truncation error has to be approximated because it cannot be measured directly. Berger approximates it by using Richardson extrapolation combined with knowledge of the order of the truncation error; unfortunately this is only known in the case of uniformly spaced grids which are not typical for problems of practical interest. Additionally, due to the difficulty associated with approximating the necessary derivatives, the predicted truncation error has to be smoothed before it can be used as a refinement parameter.

6.2.2 Threshold selection

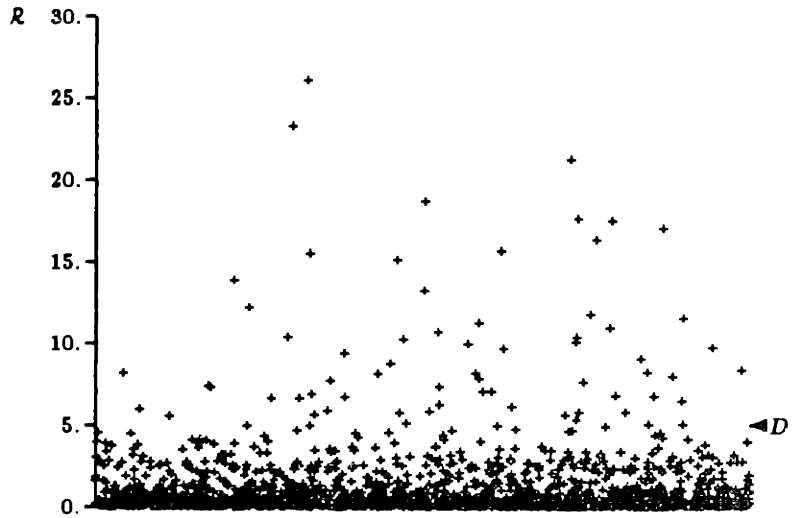
In addition to selecting and computing an appropriate refinement parameter ($\mathcal{R}$), it is necessary to determine a threshold value such that any node with a refinement parameter greater than the threshold is considered to be a part of a detected flow feature. The threshold value can either be determined *a priori* or can be computed based upon the current distribution of $\mathcal{R}$; the latter approach is used here.

The distribution of $\mathcal{R}$ (plotted versus an arbitrary abscissa) for a typical flow domain is shown in figure 6.14a, while the lower part 6.14b, is an elargement of the region near zero. For this case, $\mathcal{R} = \tilde{\nabla}\rho$ for the RAE-2822 airfoil at $M_\infty = 0.75$ and $\alpha_\infty = 3.0°$. The essential point to note from the figure is that there are a large number of nodes with very low values of $\mathcal{R}$, and progressively fewer are present as $\mathcal{R}$ increases. The same data is plotted as a histogram in figure 6.15. (The large bar at the right of the figure represents all of the nodes which have $\mathcal{R} \geq 5.0$).

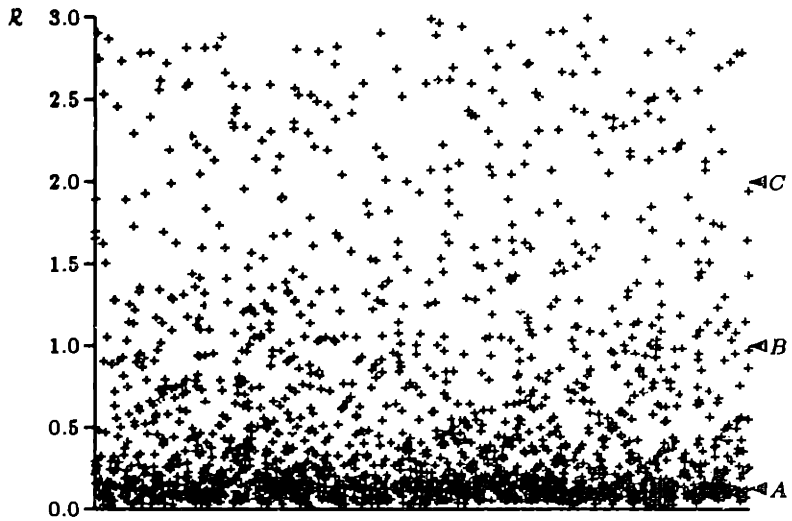
Various possible threshold levels (denoted A through D) are also shown in figure 6.14. The effect of choosing D as the threshold is that many nodes which have large $\mathcal{R}$ (signifying a feature) are excluded, resulting in a poor feature detector. Alternatively, choosing A results in almost all the nodes being detected, yielding a feature detector which does not adequately discriminate against background noise. Hence some value between these two (such as B or C) should be chosen.

The cumulative distribution function of this same data is presented in figure 6.16, where the ordinate represents the number of nodes which have refinement parameters greater than the abscissa value. This figure is much smoother than the histogram in figure 6.15 and can now be used to automatically determine an appropriate threshold, if one views the abscissa as possible values of the threshold.

Quite generally, there is a 'knee' in the cumulative distribution function



a: all nodes



b: blow-up near zero

Figure 6.14: Distribution of $\mathcal{R}$ for a typical flow domain

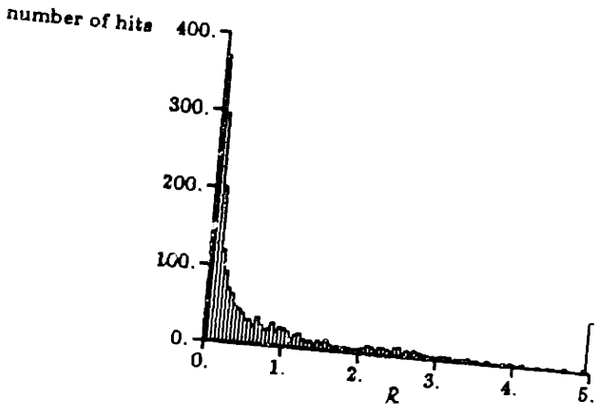


Figure 6.15: Histogram of $\mathcal{R}$ for a typical flow domain

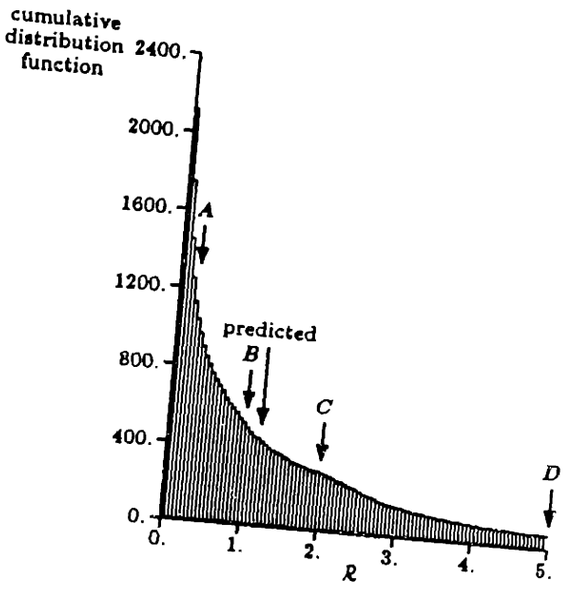


Figure 6.16: Cumulative distribution function for a typical flow domain

(in the vicinity of points B and C) which separates disparate behaviors. For relatively small $\mathcal{R} < 1$ (for example near A), a small decrease in the threshold value yields a large number of nodes which are newly detected as feature nodes, implying that the newly added nodes are actually part of the background noise. On the other hand, to the right of the knee (for example near D), a small decrease in the threshold value results in very few newly detected nodes, corresponding to the detection of actual flow features. Hence, if one can determine the location of the 'knee', then an algorithm for automatic selection of an appropriate threshold results.

Unfortunately, the 'knee' in the cumulative distribution function is not well defined, but its general location is known. Hence choosing a numerical value for the 'knee' is somewhat subjective. Fortunately however, as is shown in chapter 8, only a point in the neighborhood of the 'knee' need be selected as the threshold; the effect on the overall adaptive grid scheme is minimal.

For present purposes, the 'knee' is chosen arbitrarily as the point where $d(\frac{\text{cumulative hits}}{\text{total nodes}})/d(\frac{\mathcal{R}}{5})$ first increases above -1.0, corresponding to a line of slope -1 in figure 6.16. (The normalization parameter 5 is chosen based upon experience with a large number of computations.) It has been found that the slope of the cumulative distribution function can be computed robustly with finite differences if the function is smoothed first. The threshold resulting from this procedure is shown in figure 6.16.

In practice, this algorithm sometimes yields a threshold which is unacceptable because too many nodes are detected which are not at features. This situation can be remedied by constraining the computed threshold to exceed some specified value, such as 1.20 (20% above the average refinement parameter value). Alternatively, this situation can be remedied by constraining the threshold to be a value which limits the number of detected nodes, typically 25% of the nodes in the domain.

In addition to that threshold, which more properly may be called the

division threshold ($\mathcal{R}_d$), it is convenient to define a *fusion threshold* ($\mathcal{R}_f$). The latter is the value of the refinement parameter associated with nodes which are definitely not at flow features. For this work, the fusion threshold is set arbitrarily to 0.50, signifying that nodes with $\mathcal{R}$ less than half the average are not at features.

6.3 Grid Refinement

The basic grid refinement scheme consists of dividing cells whose refinement parameters exceed the divide threshold and of fusing (or un-adapting) those whose refinement parameters are smaller than the fusion threshold. This section contains the details of that scheme. This entire grid adaptation scheme is performed simultaneously on all grids — the global as well as previously embedded regions.

Initially, it is assumed that an appropriate refinement parameter $\mathcal{R}$ has been computed at every node and further that suitable division and fusion thresholds ($\mathcal{R}_d$ and $\mathcal{R}_f$ respectively) have been computed.

The first step of the grid refinement process involves increasing the computed refinement parameter in the neighborhood of geometric features such that the grid is automatically refined there. This is accomplished through the use of *refinement singularities*, which are defined as circles of fixed location and size which surround all pertinent geometric features. In practice, refinement singularities are defined at leading and trailing edge in H-mesh calculations and at other significant surface slope discontinuities; none are typically required for smooth airfoils and O-meshes.

The second step involves scanning through all computational cells and marking them for division or fusion according to the following rules:

- mark cell for division if $\mathcal{R} > \mathcal{R}_d$ for at least one of the cell's corner nodes

- mark cell for fusion if $\mathcal{R} < \mathcal{R}_f$ for all four of the cell's corner nodes
- do not mark the cell otherwise

Once all cells have been scanned, the third step involves the actual cell division and fusion processes as described in sections 5.3.3 and 5.3.4.

The fourth step involves the detection and elimination of *voids* and *islands*, that is single cells or lines of cells which either are not divided and are surrounded by divided cells, or which are divided and are surrounded by undivided cells, respectively. This step is required to ensure that the shape of the embedded region is smooth, thereby minimizing the length of the embedded mesh interface. Since errors are committed at such interfaces (see chapter 4), this minimization helps the overall solution accuracy.

If required, a fifth step involves the creation of a *buffer zone*, that is the division of all cells which are just outside an embedded region. The buffer zone is created in a straightforward way by visiting all cells and marking those which are just outside an embedded region; the marked cells are then divided and finally the void and island elimination step is repeated.

Chapter 7

Adaptation Control Using an Expert System

Early attempts to construct an adaptation algorithm which was both robust and efficient yielded many failures, mainly due to the fact that adaptation is currently not firmly rooted in theory, but instead is somewhat heuristic. To build an adaptive system which is both efficient and robust required a mechanism by which experience could be easily built into the program, and thus an expert system has been employed.

This chapter begins by describing expert systems, both in terms of their components and their operation. Through a case study, local compressible flow analysis, the operation of an expert system can be contrasted with the operation of a conventional system, making the advantages and disadvantages of each apparent. This then leads to the development of a general hybrid system architecture which subsequently forms the framework for the MITOSIS adaptive grid program. The chapter concludes with a description of the "hooks" necessary to incorporate the grid adaptation procedure into the hybrid system approach.

The material here has been collected from a number of books and articles on the subect of expert systems and artificial intelligence. Amongst the most useful are the books by Newell and Simon[65], Winston[88], and Goodall[40].

Survey articles on expert systems written specifically for those in the field of aeronautics include those by Kutler[51] and Cheeseman[23].

7.1 Expert Systems — Background

Expert systems are computer programs which solve problems that are normally associated with human experts by applying a *reasoning* mechanism to a body of *knowledge* in the expert's domain. These human experts can be from a variety of disciplines including business, medicine, science, and engineering.

The term *expert* is not to be taken too literally, since many systems perform either at a level below that of an expert or at a level higher than any one human expert. For this reason, many modern systems are instead being referred to as *knowledge systems*[42]. Within this thesis however, this fine distinction will not be made and the term expert system will be used, independent of the level of its expertise.

Expert systems comprise but one field of study in the broader discipline of artificial intelligence which has been the subject of significant research efforts over the past thirty years. Other fields of research within artificial intelligence include:

Robotics — associated with analyzing visual and/or tactile inputs in order to allow robots to perform assigned tasks in a constantly changing environment. One specific area of research within this field is vision which involves the determination of the number of and identity of objects in a scene.

Natural language processing — associated with taking natural language (for example, English) sentences and capturing the concepts and facts conveyed by them. The inputs and outputs can be either written or spoken.

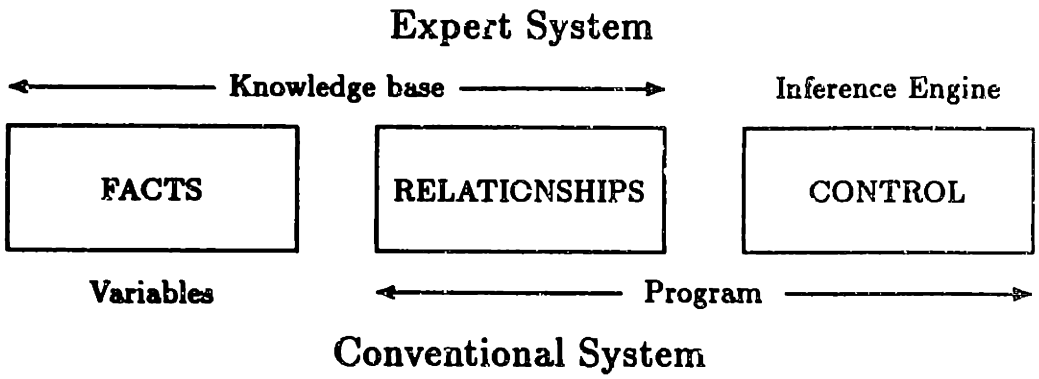


Figure 7.1: Organization of conventional and expert systems

Learning — associated with analyzing experiences and either extracting from them relevant knowledge or rejecting those which are inconsistent or irrelevant. Knowledge which is extracted could then be either used directly by experts or as input to an expert system.

The working definition of expert systems given above, although somewhat terse, conveys the essential difference between expert and conventional systems, as shown in figure 7.1. Here a conventional system is taken to be one in which programming statements typically written in a procedural language such as FORTRAN or C are translated rather straightforwardly into the machine instructions of a von Neumann machine.

In any program, there are three basic components:

Facts — descriptions of the state of a physical or abstract object or situation of interest. For example, the radius of a circle or an employee's name.

Relationships — statements concerning the interconnections of facts. For example, the circumference of a circle is related to its radius by the

algebraic relation $c = 2\pi r$.

Control — statements concerning the order with which the various relationships should be applied to certain facts in order to accomplish a task, typically the creation of new facts.

In conventional systems, programs are broken into two major components: the *variables* which contain the facts and the *program* which contains both the relationships and control. Typically relationships are coded as assignment statements while control is coded implicitly through statement ordering and explicitly as GOTOs, DO-loops, and subroutine calls. The important point here is that the relationships and control are intertwined.

In expert systems on the other hand, the rules are combined with the facts in the *knowledge base* and the control is in a separate component known as the *inference engine*. In this way, the domain dependent information (the facts and the rules) is segregated from the domain independent information (the control), leading to many inherent advantages, as will be discussed below.

Before continuing into the actual workings of expert systems, a word of caution should be included. Expert systems are not magic, but simply a different programming strategy. There isn't anything that can be done with an expert system that can not be done with traditional systems, and vice versa. Expert systems do not make complex problems simple, but instead provide a means for dealing with the complexity. The major difference between the two type of systems is the efficiency with which certain types of problems can be coded and computed.

7.2 Expert System Components

In the previous section, it was pointed out that all computing systems contain three major components: facts, relationships, and control. In this

section each of these will be discussed as they relate to an expert system. The discussions are intended to be relatively complete and as a result treat some expert system features which are not included in MITOSIS' expert system.

7.2.1 Facts

The first part of an expert system's knowledge base consists of the representation of *facts*, which in their simplest form can be expressed as attribute:value pairs. The *attribute* refers to the name of the general characteristic or property conveyed by the fact whereas the *value* contains the numerical or symbolic value for the fact at the current time. Examples of attribute:value pairs are `MACH:0.80` which has a numerical value and `AIRFOILTYPE:'NACA 0012'` which has a symbolic value.

In many ways, attribute:value pairs are very similar to variables in conventional programs, where the variable name serves the same purpose as the attribute's name. An important distinction however between attribute:value pairs and conventional system variables is that `UNKNOWN` is a valid value in most expert systems, that is that the system knows that the value is not known at the present time. The operation of many expert systems rely heavily on allowing values to be unknown, and hence sought as part of the solutions.

Another important distinction is that many expert systems can deal with uncertain facts, that is pieces of information which are not definitely known to be either true or false. In many expert system applications, specific facts cannot be stated conclusively based upon some observation, but instead belief in the fact is gradually accumulated as many pieces of inconclusive evidence are amassed.

There are many ways of expressing certainty, with the most common shown on the scale in figure 7.2. Here a *certainty factor* is defined in the range

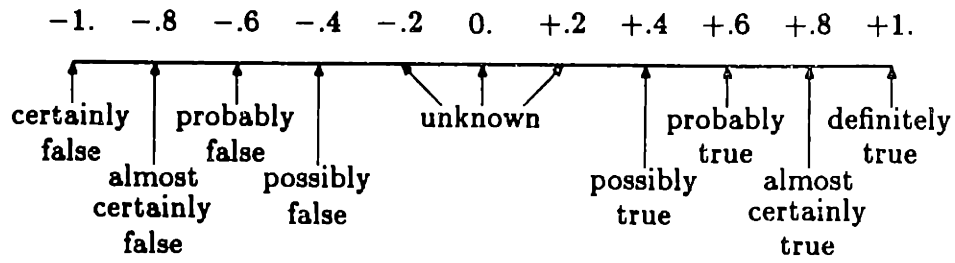


Figure 7.2: Range of permissible certainty factors

from +1 to −1, with positive certainty factors representing true statements and negative certainty factors representing false statements. Facts whose values are unknown are assigned a certainty factor of zero.

It is important to point out here that certainty factors are not probabilities, but rather express the confidence level or certainty with which a fact is known.

In many expert system applications, the attribute:value representation of facts is not sufficient and is therefore expanded into object:attribute:value triplets. Here *objects* represent either physical objects such as an `airfoil` or conceptual entities such as `free-stream-conditions`. The *attributes* then describe the various characteristics or properties of the object; for example, an `airfoil`'s attributes could include its `name`, `thickness`, `camber`, or even its `design-lift-coefficient`. A value would then be applied to each attribute for a given `airfoil`. For example, one could have the triplets `airfoil:thickness:0.12` or `airfoil:name:NACA 0012` which represent but two of the branches in the description of an object `airfoil`, whose full description might be given as

```
airfoil:  name:                    'NACA 0012'
          thickness:               0.12
          camber:                  0.0
          design-lift-coefficient: 0.0
```

Clearly, going from simple attribute:value pairs to object:attribute:value triplets aids in the organization of facts within an expert system's knowledge base. But the real power of the more general triplets lies in the ability to treat more than one airfoil at a time. The object:attribute part of the triplets contain a generic description of the object, that is an `airfoil` has a `name`, `thickness`, etc.; specific *instances* of airfoils are distinguished from each other by different sets of values, for example

```
airfoil:  name:       'NACA 0012'
          thickness:  0.12
          camber:     0.0
```

and

```
airfoil:  name:       'RAE 2822'
          thickness:  UNKNOWN
          camber:     UNKNOWN
```

represent two different airfoils. This allows relationships (described below) to be written by referring to an object's attributes, and as such is not tied to a specific instance of the fact but can be applied to all instances of the object.

Before continuing on to the relationships, two other representations of fact, *frames* and *semantic networks*, will be discussed briefly. These are included here for completeness even though they will not be used in this thesis.

Frames are a generalization of the object:attribute:value triplets in which the values are replaced by *slots*, each of which can contain multiple pieces of information such as a current value, a default value, and the name of a procedure which can be "called" in order to fill in a value. Thus frames offer an easy method by which procedures can be linked into an expert system. For attributes for which there is a well-defined, efficient procedure for the determination of a value, this can be a great asset. The chief disadvantage of frames is that their complexity makes the development of the expert system

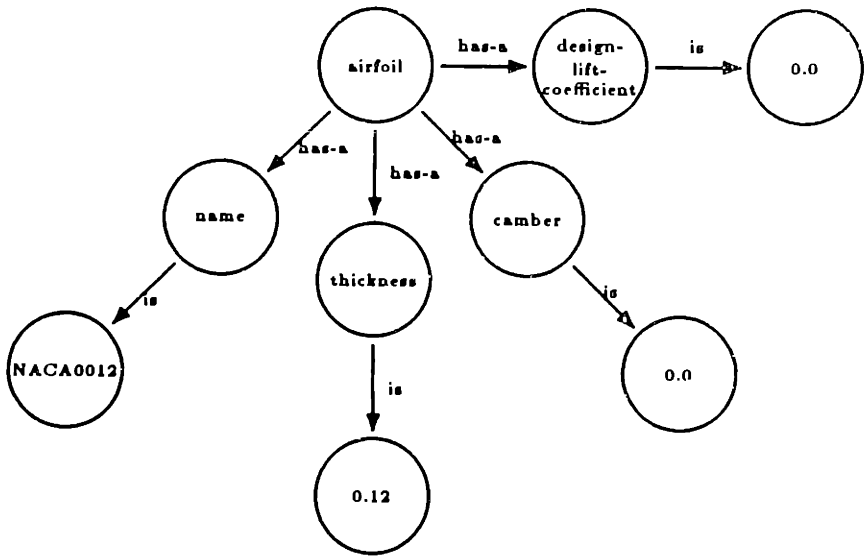


Figure 7.3: Simple semantic network

using them much more difficult than systems using object:attribute:value triplets.

Semantic networks are another generalization of the object:attribute:value triplets. Here the facts are represented by a network of *nodes* which represent objects or descriptors and *links* which relate objects and descriptors. The airfoil object:attribute:value triplet for a NACA 0012 airfoil described above is expressed in the semantic network shown in figure 7.3. Notice that the links are labeled; `has-a` links identify nodes which are properties or attributes of another node (in this case `airfoil`), and `is` links identify specific values for a given node.

If semantic networks were limited to the simple network hierarchy shown in figure 7.3, they would have little if any advantage over object:attribute:value triplets. Their power lies in their ability to express a complex hierarchy of facts as shown in figure 7.4. In this way a semantic net really becomes a full representation of the domain knowledge since it contains both facts and the relationships between facts. Many researchers feel that this network

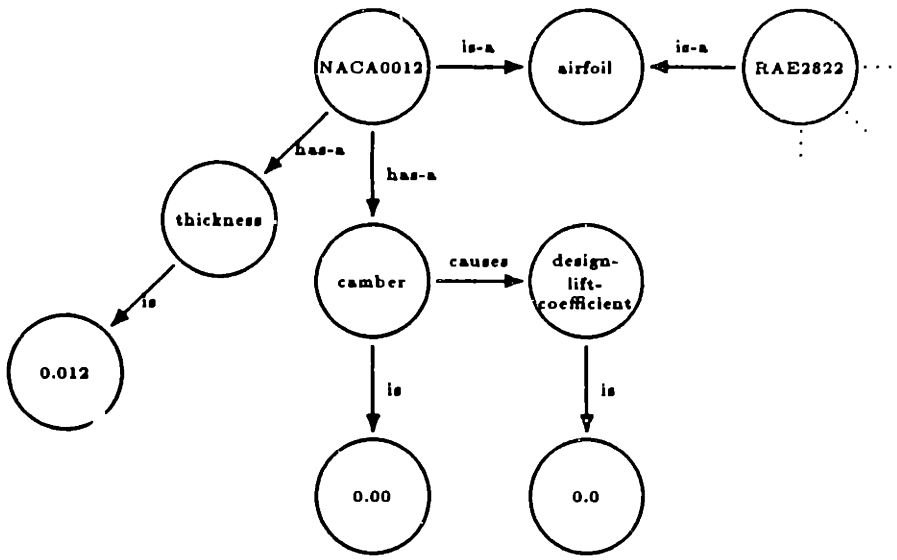


Figure 7.4: More general semantic network

approach to storing facts very closely mimics the way in which humans store knowledge[25]. Unfortunately semantic networks add greatly to the complexity of an expert system, again making the development of an expert system for a given application much more difficult than an equivalent system using object:attribute:value triplets.

7.2.2 Relationships

The second part of an expert system's knowledge base consists of the representation of *relationships* between facts, typically stated as cause-and-effect statements, known as *rules*, which have the general form:

```
Rule:
        if:         premise clause 1
        andif:      premise clause 2
                    ⋮
        then:       action clause 1
        andthen:    action clause 2
                    ⋮
```

(Note that if the facts are represented by a semantic network, then explicit rules of the type described here are not needed since the relational information is encoded directly in the network's links.)

In general, *premise clauses* are simple relational operators of the form

```
fact-1  .comparator.  fact-r
```

which evaluate to either TRUE, FALSE, or UNKNOWN (or to certainty factor for systems allowing uncertainty). Here the two facts `fact-1` and `fact-r` can each be either an object:attribute pair name for facts represented as object:attribute:value triplets, an attribute name for facts represented as attribute:value pairs, a symbolic or numeric constant, or the value UNKNOWN. The comparator typically comes from the algebraic set

```
.EQ.  equal to
.NE.  not equal to
.LT.  less than
.LE.  less than or equal to
.GE.  greater than or equal to
.GT.  greater than
```

Examples of valid premise clauses for facts described by object:attribute:value triplets are

```
airfoil:name  .EQ.  'NACA 0012'
```

and

```
airfoil:thickness  .LE.  limits:maximum-thickness
```

where in the latter example there is an object called `limits` which has amongst its attributes `maximum-thickness`. For facts represented by attribute:value pairs, premise clauses take the simpler form

```
thickness  .LE.  maximum-thickness
```

Considerable complication arises when multiple instances of objects are allowed as in object:attribute:value systems. For example, in the premise clause

```
airfoil:thickness  .LE.  limits:maximum-thickness
```

it is possible that there is more than one airfoil, each with a thickness less than or equal to the maximum allowable thickness. Hence one can only state that the premise clause is true, false, or unknown when a particular instance of `airfoil` is *bound* to the rule. In fact since `limits:maximum-thickness` could also refer to more than one set of limits, the rule must also be bound to a specific instance of `limits` before it can be evaluated.

In general the premise of any rule can contain reference to a particular class of object, for example `airfoil`, many times. In some cases the various references to the object have to bound *consistently*, that is they have to be bound to the same instance of the object. For example, if one wanted to express "if there is an airfoil whose name is not 'NACA 0012' and whose thickness is less than of equal to 0.12", it might be written as

```
if     airfoil(a):thickness  .LE.  .12
andif  airfoil(a):name       .NE.  'NACA 0012'
```

where the subscript a denotes consistent bindings for the two references to `airfoil`. On the other hand, the premise "if two airfoils have the same thicknesses" would require inconsistent bindings, written as

```
if     airfoil(a):thickness  .EQ.  airfoil(b):thickness
andif  airfoil(a):name       .NE.  airfoil(b):name
```

where the second clause is required to assure that `airfoil(a)` and `airfoil(b)` are not bound to the same instance of `airfoil`.

Given a set of bindings for a rule premise, the entire premise can then be evaluated. Since the entire premise is written as the *conjunction* of premise clauses (clauses connected by *and*), any one premise clause which evaluates to FALSE thus causes the entire premise to fail. Unfortunately, sometimes

the domain knowledge requires that a rule have *disjunctive* clauses (clauses connected by *or*) which cannot be written directly in the above scheme. In these cases the rule must instead be written as a series of rules, one for each of the disjunctive clauses, which can sometimes results in inefficient expert system operation. As a result, some modern expert system languages allow more complicated forms of premises, including explicit statements of disjunction[16].

The *action* part of a rule consists of one or more action clauses, each of which either modifies facts in the knowledge base, performs a user-defined function, or performs input and/or output. In some expert systems, it is even possible to create new rules or disable old ones.

The modification of the facts in the knowledge base can be done in three ways: a new instance of some object can be created, typically with its values all being UNKNOWN; the values in an existing instance of an object can be modified; or an entire instance of some object can be deleted. In the latter two cases, the object is typically bound to the same instance of the object as specified in one of the condition clauses. Some expert systems do not allow modification of values, so in these cases the rule must contain at least two action clauses — one to create the new, updated instance and one to delete the old instance[16].

The standard functions which are supported by most expert systems include the arithmetic operations (addition, subtraction, multiplication, and division) as well as other common mathematical functions such as root extraction, exponentiation, and absolute values. User-defined functions are generally just implementation-language functions which are executed based upon facts supplied through the rules and which generate new facts which are stored in locations specified by the rule.

Typically when a rule's action clauses are *fired*, or executed, they are performed sequentially. Thus in some sense the ordering of action clauses

defines a mini-procedure, with the restriction being that branching and other logical operations are forbidden.

For systems which employ certainty factors, there is often a considerable amount of computation involved in the execution of the action clauses for the purpose of assigning certainties to new or modified facts. First of all, the certainty of each premise clause needs to be determined based upon the certainties or the facts involved. Then the certainties of the premise clauses need to be combined into a single certainty for the entire premise, usually by using the minimum certainty factor from the conjunctive clauses. This certainty is then attenuated by a factor prescribed in the rule such that there is an interim output certainty for the fact which is then combined with the fact's certainty from before the execution of the action clause. A thorough discussion of certainty processing is contained in Winston[88].

7.2.3 Control

The organization of expert systems is based upon the realization that expert problem solving is composed of a very small set of domain-independent inferencing strategies applied to specific, domain-dependent knowledge. In expert systems, the control component, called the *inference engine*, is responsible for applying these general inferencing strategies to the facts and relationships contained in the expert system's knowledge base.

The basic operation of an expert system's inference engine consists of the following steps

search — search through all rules and all possible object bindings, making a list of those which are appropriate to execute at the present time. If none qualify for the list, then the execution of the inference engine halts.

select — if the list of rule-binding pairs contains more than one entry, then select the most dominant entry from the list. This step is also known

as *conflict resolution.*

act — perform the actions in the action clauses of the most dominant rule on the list, resulting in a modified set of facts in the knowledge base. The process of executing the action clauses in a rule is also known as *firing* the rule.

repeat — go back to the search step

This method of propagating new facts through the knowledge base is based upon the logical principle of *modus ponens*, which basically states that if A is known to be true and if there is a rule which states "If A, then B", then it is valid to conclude that B is true.

The description of the search phase in the above discussion is somewhat vague since there are two different methods, known as *forward-chaining* and *backward-chaining*, for determining which rules are appropriate to execute. On the one hand, the forward-chaining the strategy starts with an initial set of facts and applies the rules whose premises are satisfied by the current facts such that subsequent facts are derived. This technique, also known as *data-driven* inferencing, is therefore frequently used when one wants to determine all consequent facts from some initial set of facts. In backward-chaining on the other hand, the strategy is to start with a *goal* (a fact for which one desires a value), and then search for rules which in their action clauses assert a value for the desired goal. This technique, also known as *goal-driven* inferencing, is frequently used when the initial set of facts yields many subsequent facts, only one of which is required to be found.

It is clear that neither the forward- nor backward-chaining strategy is best for all problems, although one can always force either one to do a task which is ideally suited for the other. Generally problems of synthesis, planning, and control are ideally suited for forward-chaining whereas diagnosis, identification, and classification are problems typically attacked with

backward-chainers. There are however some problems which are most appropriately treated by a combination of these two strategies, where in effect the problem is attacked from both ends, working toward the middle.

In the following sections, the operation of forward- and backward-chaining inference engines are described, along with a discussion of conflict resolution. For simplicity it is assumed that facts are represented by attribute:value pairs and thus each rule can only have one binding. In this way, the statement "if the rule has no false premises ..." makes sense; if an object:attribute:value representation were used, one would instead have to say "if the rule, bound to the current object, has no false premises ..." and there would have to be a loop which successively binds each appropriate object to the rule. This added complication does not alter the basic inferencing strategy and thus can be neglected in the following discussions.

It is important to note that the knowledge in the knowledge base (the facts and the relationships) should not depend upon the type of inferencing strategy used but instead should be statements about the domain of interest. Unfortunately if the domain knowledge does not lend itself to the inferencing strategy being used, one often has to create rules and facts which in essence contain control information; this practice should be avoided if at all possible.

7.2.3.1 Forward-chaining

The basic operation of forward-chaining inference engine is shown in figure 7.5. It basically consists of a search-select-act loop, with a provision for stopping. In its operation, it uses a list of rules which are currently available for execution, called the *rule stack*, the management of which is described below.

Each cycle begins by deleting all rules left in the rule stack from the previous cycle. This step ensures that the control is purely data-driven, that is the operation in each cycle depends only on the current facts.

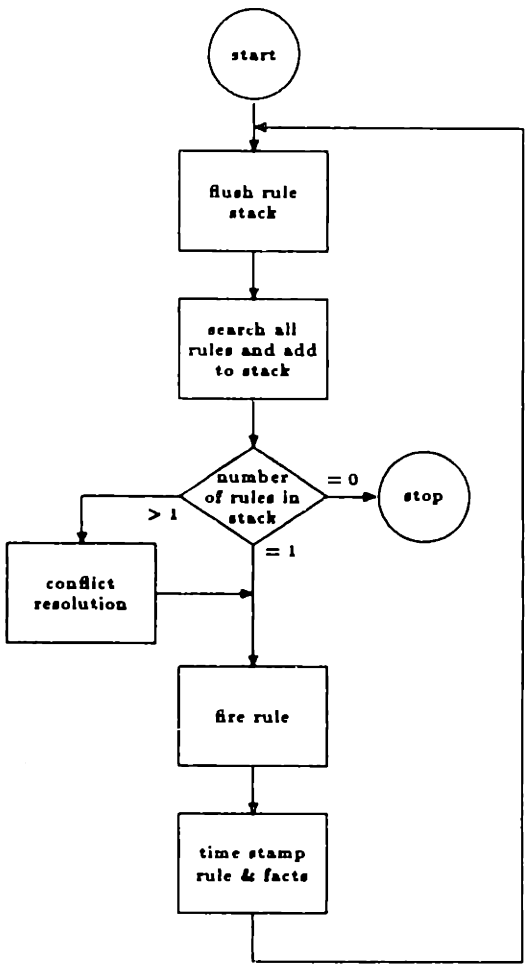


Figure 7.5: Flowchart of typical forward-chaining inference engine

In the next step, the search step, each rule is examined and those meeting the criteria:

- references no attributes with currently unknown values in either its premise or action clauses, except those attributes for which a value is asserted by the rule;
- has no false premise clauses; and
- asserts a value for at least one attribute which currently has an unknown value

are termed *triggered* and are added to the top of the rule stack. These criteria clearly demonstrate the use of UNKNOWN as a valid and essential value in the fact representation. The requirement that the rule has at least one unknown assertion is necessary to keep the inference engine from firing (executing) rules unnecessarily. This search step normally accounts for the vast majority of computer resources (CPU time), for every rule has to be examined (and in systems with object:attribute:value triplets, the bindings of all appropriate objects to each rule must be considered). The section below on acceleration techniques discusses various methods which have been devised to make the search more efficient.

The result of the search step is a rule stack which contains zero, one, or more rules. If there are zero, then there are no rules which can expand the current state of knowledge (as expressed by the facts) and therefore the inference engine stops. If on the other hand there is more than one rule in the stack, then there must be a mechanism for deciding which of these rules should be applied at the current time. A selection process, termed *conflict resolution*, is applied to the rules in the stack, in essence re-ordering them such that the most dominant rule floats to the top of the stack. Conflict resolution is described more fully in the section below.

The act step consists of *firing* the rule which means that the actions in the rule's action clause(s) are performed. Typically these actions involve giving a fact some specified value or the performance of simple algebraic functions (addition, root extraction, etc.). As will be seen later in this chapter, these actions need not be limited to simple tasks. In addition to firing the rule, the act step also *time-stamps* the rule and the newly asserted facts, the significance of which will be apparent in the discussion of conflict resolution strategies.

The operation of the forward-chainer can be viewed as a *tree search*, forward from the initial data. Whether the search is *depth-first* or *breadth-first* is controlled by the choice of conflict resolution strategy.

One of the most well known and commercially successful forward-chaining expert systems is XCON (originally called R1), written by John McDermott and the Digital Equipment Corporation[89]. XCON's domain of expertise is the custom configuration of VAX computer systems from the over 400 components available in the product line. Its initial set of facts consists of a customer's order, with its output (final set of facts) being a complete order including a layout of all components in their respective cabinets. The rules contain configuration heuristics, such as for determining if the components in the customer's original order are compatible, assigning components to power supplies and cabinets, and designing the appropriate cables. After three years of development, the system which contained about 500 rules was achieving better than 97 percent correct configurations (a much higher percentage than the humans who routinely performed the task before XCON become available).

The configuration problem treated by XCON naturally works by starting with some facts and building them up until the configuration is complete in much the same way as is done by a forward-chainer. XCON's forward-chaining is breadth-first, consistent with the stepwise configuration process

used by human experts. In contrast, the use of a backward-chainer would have been awkward since there are an enormously large number of possible final configurations and assuming one to backward-chain from would not have been an effective strategy.

7.2.3.2 Backward-chaining

In contrast to the relative simplicity of forward-chaining, the operation of a backward-chaining inference engine is much more complicated, as shown in figure 7.6. Like the forward-chainer, the backward-chainer consists of major loop which is repeated, but here only until a value has been determined for the desired goal. Unfortunately the looping operation of the backward-chainer can be obscured by the complex logic associated with *sub-goal creation* and *backtracking*.

To aid in the description of the operation of the backward-chainer, the sample goal tree shown in figure 7.7 will be used. Here, facts (or goals) are denoted by letters in circles and rules are represented by the numbered links between facts. A link which branches out, with the branches spanned by a small horizontal line denotes a rule with multiple premise clauses. For example, rule 1 states that

```
Rule 1:
        if:     b
        andif:  c
        andif:  d
        then:   a
```

A fact with multiple rules emanating from it signifies that both rules imply the fact, thus both rules 4 and 5 assert a value for d in their action clauses. Furthermore, the hierarchical structure of figure 7.7 indicates that the main goal to be pursued is a, which can be asserted given known values for either b, c, and d or from e.

The backward-chainer maintains two stacks for its operation — a *rule*

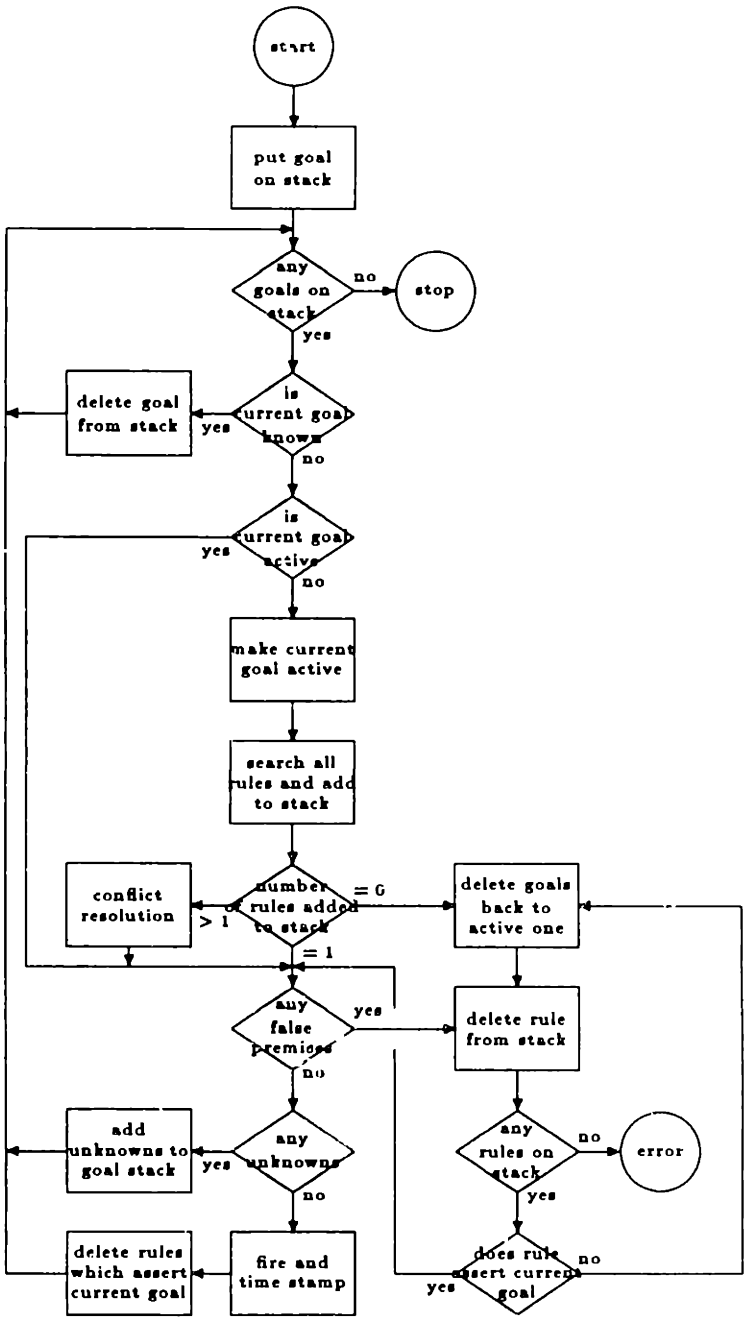


Figure 7.6: Flowchart of typical backward-chaining inference engine

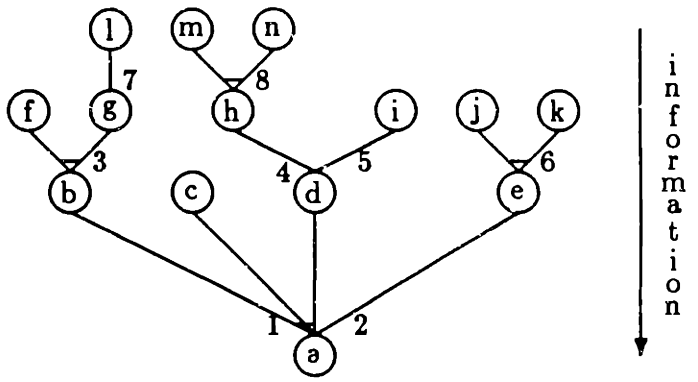


Figure 7.7: Sample goal tree

stack and a ***goal stack***. The rule stack contains the names of the rules which result from the search step, basically those which assert the current goal. The goal stack contains a list of the goals to be pursued, with the ***main goal*** (a in figure 7.7) being at the bottom of the stack. Goals can be either ***active*** or ***inactive*** depending on whether they are in the line from the currently pursued goal (the one on the top of the stack) to the main goal. For example in figure 7.7, if the current goal is h, then a, d, and h would be active goals and any other goal on the stack would be inactive.

The operation of the backward-chainer begins by putting the user-desired goal on an empty goal stack and calling it temporarily inactive. Thus the user-desired goal is always at the bottom of the goal stack.

The main loop begins by noting that if there are no goals on the goal stack, then the operation of the backward-chainer has been successful and processing terminates. If there is at least one goal on the goal stack, the backward-chainer then determines if the value of the goal on the top of the stack is known. If it is, that goal is deleted from the stack and processing continues once again at the top of the main loop. These two steps ensure proper operation in the event that the user-defined goal is known when the backward-chainer is invoked or in the event that the goal on the top of the stack was given a value during the firing of a rule.

The next step is to inquire if the *current goal* (the goal currently on the top of the stack) is active or inactive. If it is active, then the search step was performed previously and can be skipped in this cycle through the main loop. If it is inactive, it is first activated and then all the rules are searched, adding those which meet the following criteria to the rule stack:

- asserts a value for the current goal;
- has no false premises; and
- references no attributes with currently unknown values (either in premise or action clauses) which are also currently active goals.

This last criterion is necessary so that the backward-chainer does not get into an infinite loop situation with two rules, one of which states "if A, then B" and the other being its converse "If B, than A". As with the forward-chainer, for facts represented by object:attribute:value triplets, the search must be over all possible rule-object bindings. It should be pointed out here that these rules are added to a rule stack which has not been cleared before the search as was the case in the forward-chainer.

Following the search step, the backward-chainer determines the number of rules just added to the rule stack. If none were added, then this goal (fact) cannot be determined at this time and the backward-chainer must backtrack through the goal tree, as described below. If more than one rule was just added to the rule stack, then a conflict resolution strategy (described in the following section) is used to order the newly added rules, with the most dominant rule floating to the top of the stack.

Processing then continues by ensuring that the rule on the top of the stack does not currently have any false premises. This extra check is really only necessary for the case where the search step was skipped.

If there are no false premises, then the backward-chainer determines if any referenced attribute currently has an unknown value. If there are any

unknown facts, these are added to the goal stack (and marked inactive) and processing continues back at the top of the main loop.

Assuming that the rule on the top of the stack has neither false nor unknown premises, the rule is then fired and the rule and newly created facts are time-stamped for later possible use in conflict resolution.

The final step of the main loop deletes all rules from the top of the stack which assert the current goal. By design then, the rule in the top of the stack is always deleted but it takes along with it any other rule which was added to the stack as an alternative way of determining the current goal. For example in figure 7.7, both rules 4 and 5 would be added to the rule stack when the current goal was d in the search step above. If the conflict resolver favored rule 5, and if rule 5 were fired, then both rules 4 and 5 should be deleted from the rule stack since d would now be known.

The only other part of the backward-chaining inference engine is the backtracking mechanism. To understand how it works, first consider an example of where it would be necessary. In figure 7.7, assume that when a was the current goal, the conflict resolver favored rule 2 over rule 1 and further that rule 6 and j had been added to their appropriate stacks, such that the current stack were given by

Rule stack		Goal stack	
		current–	j (active)
current–	6		k (inactive)
	2		e (active)
	1		a (active)

Now suppose that no rules can be found which assert a value for j. This requires the backward-chainer to abandon its search along the 2–6 branch and backtrack to rule 1.

The back-tracker works in steps, deleting goals at one level at a time. It first deletes all goals back to the previous active one. In this example, j

and k would both be deleted since even if the backward-chainer could find a value for k, it would be of no use since both j and k need to be known for rule 6 to be able to be fired. The back-tracker then deletes the rule which failed from the rule stack. If this leaves the rule stack empty, then there is no way that the backward-chainer can assert a value for the main goal (a) and an error situation exists. If there are rules on the rule stack, then the rule on the top of the stack is examined to determine if it asserts the current goal and if it does, the backtracking is complete and processing continues back in the main loop. If the current rule does not assert a value for the current goal, then another level of the search tree is pruned by going back to the beginning of the backtracking mechanism. In the example above, the backtracking completes with the stacks

Rule stack	Goal stack
current– 1	current– a (active)

The backward-chaining inference engine just discussed is an example of a depth-first search, that is a search which chooses a branch at each fork and pursues that branch until either the fact at the leaf is either known or unattainable. This tends to generate a very focussed line of investigation, where in a sense an assumption is made at each fork of the goal tree and enough information is gathered to either support or discredit the assumption through the pursuit of one or more goals along the chosen branch. Alternatively, a breadth-first search would examine all branches at a level of the goal-tree hierarchy before going on to the next lower level.

Which of these two strategies is better depends on the shape of the goal tree. For a tree which has some very short branches and some very long branches, a breadth-first search might be better whereas a goal tree with branches of approximately the same length might be better attacked with a depth-first search. Another criterion which is often used to determine the type of search to use is based upon the reaction of the end-user. In many

systems, some fact values are determined by asking the end-user for a value and in such a case the focussed questioning of a depth-first search is often more palatable to the end-user.

One of the first true expert systems, called MYCIN[17], is an example of a backward-chaining expert system. MYCIN's domain of expertise is the diagnosis of bactermia (infections that involve bacteria in the blood) and meningitis (infections that involve inflammation of the membranes of the brain and spinal cord). MYCIN was developed at Stanford University in the mid–1970's more as a research system than as an actual consultation system to be used by physicians, although in its final form it proved to be better than the human experts to which it was compared.

MYCIN begins a consultation by acquiring some background facts about the patient and a general description of the symptoms that the patient is experiencing. Then in much the same way as is done by human experts, it conjectures diagnoses which it then uses backward-chaining to support or contradict.

One important property of MYCIN is that it used certainty factors in all its facts and rules. This is consistent with the field of medicine where knowledge is not black-and-white but rather somewhere in between, and even often unknown. Thus MYCIN's final diagnosis is often stated with qualifying phrases such as "there is strong suggestive evidence that ..." and "it may be possible that ...".

Another important property of MYCIN was in the interface with the user (physician). Many of the facts in the goal tree were obtained by asking the user (who could answer UNKNOWN), and thus MYCIN's developers found that the concentrated investigation consistent with a depth-first search to be essential. The user interface also contained an *explanation facility*, where the user could respond "why?" to one of MYCIN's questions, thereby eliciting from MYCIN an explanation of the line of reasoning that it was pursuing

which led to its question. Additionally after a diagnosis, the user could ask "how?" MYCIN arrived at its conclusion.

MYCIN's success has prompted the development of many other expert systems for diagnosis of both other medical and non-medical problems.

7.2.3.3 Conflict resolution

In both forward- and backward-chaining, the search step often produces a *conflict set*, that is a list of more than one rule which satisfies the search's criteria. As mentioned above, the mechanism by which the conflict set is ordered such that a dominant rule emerges is called conflict resolution.

Currently there isn't any one preferred means of ordering the conflict set but instead there is a whole assortment of ordering criteria from which the author of the expert system chooses. A few of the most popular ordering criteria are:

Specificity — In its purest form, specificity ordering states that if there are two rules A and B, and if A's premise is a subset of B's premise, then rule B should dominate on the grounds that it is more specialized to the current situation. This ordering, which is mostly used in forward-chaining systems, is somewhat complicated to determine as defined above and so in general the ordering is simply based on a specificity number which is a function of the number of and complexity of the premise clauses.

Rule recency — Here the conflict set is ordered based upon the time-stamps associated with each rule in the conflict set. The time-stamp, which is applied by the inference engine, is a number indicating the order in which the rules have been fired, with all rules which haven't yet been fired having a time-stamp of zero. The ordering can be such that either the most recently used or least recently used rule is favored,

depending on whether the inferencing should be focussed or broad-based.

Fact recency — The ordering process is essentially the same as rule recency, except the time-stamps of the facts are considered. Since most rules contain premises which refer to more than one fact, the composite time-stamp of the rule can be taken either as the minimum, the maximum, or even some average of the referenced facts' time-stamps.

Refraction — Refraction, which requires that a rule not fire more than once using the same input set of facts, is in some sense a combination of rule and fact recency. The implementation of refraction requires that the inference engine keep a list of all rule firings along with the facts that were involved. The conflict resolver then compares the entries in the conflict set with that list and orders them appropriately, where obviously the least recently used rule-fact combination is favored.

Expansibility — In this ordering, which is most often used in backward-chaining systems, the rule with the least number of unknown premises floats to the top of the conflict set. Thus the use of this conflict resolution strategy is an effective way of containing (not expanding) the number of goals added to the goal stack.

Certainty — For systems which can handle uncertainty, it is often desirable to favor the rules whose premises are most certainly known. As in fact recency, the certainty of the entire premise can be the minimum, the maximum, or some average certainty factor of the referenced facts.

Priority — Sometimes one wants to fire a rule whenever its premise is satisfied, regardless of the other rules in the conflict set. This is typically true in cases where some external input (interrupt) is meant to divert the attention of the inference engine, even if only temporarily. In most

expert systems, the vast majority of the rules will be given the nominal priority, with only a few rules which act as interrupts having a higher priority value.

Each of the ordering criteria described above would itself be insufficient to pick a dominant rule out of a conflict set. For example, the two recency orderings could not choose from amongst rules which had never fired and hence had time-stamps of zero. Therefore most conflict resolution strategies actually consist of multiple orderings, applied successively until one rule dominates over the others in the conflict set. In the OPS5 forward-chaining inference engine for example[16], the LEX conflict resolution strategy consists of successively applying refraction, data recency (most recent dominates), and specificity.

7.2.3.4 Acceleration techniques

As stated above, the vast majority of the time spent in most expert systems is in the search step, where all rules are examined to find those which satisfy the search requirements — either true premises in forward-chaining or goal assertion in backward-chaining. Many techniques have been developed to accelerate the search with three of these being the subject of this section.

One of the simplest acceleration techniques, *context limiting*, is based upon the observation that typically a task can be broken into subtasks called contexts. For example in XCON (described above), the contexts could be checking the order, assigning cabinets and power supplies, and designing the appropriate cables. While XCON is checking to be sure that the ordered components are compatible, it would be wasteful to consider rules which dealt with cable design.

To implement context limiting, each rule is assigned a context by the author of the expert system. The inference engine, which keeps track of the current context, begins with some starting context, say "initial", and thus

in its first search through the rules, it only considers those whose context is "initial", in essence limiting the computer resources needed for the search. Of course there needs to be a rule in each context whose responsibility it is to change to another context once the processing of the current context is complete.

A slightly more difficult acceleration technique to include in a forward-chaining inference engine is ***static rule ordering***. In forward-chaining with ***dynamic rule ordering*** as described above, the search step produces a conflict set which is ordered by the conflict resolver and the most dominant rule is fired. The remainder of the conflict set is discarded and the cycle begins again. Static rule ordering achieves its efficiency by not discarding the conflict set, but instead by successively firing all the rules in the conflict set, in effect making the search more breadth-first. Care must be taken here by the inference engine to ensure that the premise of each of the succeeding rules has not been falsified by a prior rule firing.

The final acceleration technique to be discussed here is the *Rete match* ***algorithm***, implemented in the forward-chaining OPS5 inference engine. A network, called the Rete net, is created to link each fact to all of its references in both premise and action clauses. Additionally, the network holds the results of any premise clause that can be evaluated, that is those premise clauses which references facts with known values.

When the inference engine first starts up, an exhaustive search of all rules, bound with all permissible objects, is completed, with the results of all premise clause evaluations being stored in the network. The conflict set then passes through the conflict resolution mechanism as usual and a dominant rule is selected for firing. During the firing of the rule, facts are assigned values as usual, but in addition the consequences of these new facts are propagated through the Rete net with the corresponding premise clause evaluations immediately resulting in additions to and/or deletions from the

conflict set. In this way the search step can be circumvented in all but the first cycle, with the only disadvantage being the storage and complexity required to implement the Rete network.

7.3 Model Expert Systems

This section describes two expert systems, one forward-chaining and one backward-chaining, which use a common knowledge base to attack a model fluid mechanical problem. Though the expertise here consists mainly of simple algebraic substitutions, these model systems are included to highlight the real strengths and weaknesses of expert systems through the comparison of their operation with the operation of conventional procedural systems.

The domain of expertise of these model expert systems is local compressible flow analysis, that is the aerodynamic and thermodynamic relationships which govern the flow at a point in a streamtube. Initially the conditions will be characterized by the attributes given in Table 7.1 with the relationships given in Table 7.2. It is assumed that the values and relationships are all written in a consistent set of units (or are non-dimensional).

The inference engines used in these two expert systems will have the following characteristics:

- facts are represented by attribute:value pairs
- certainty factor are not employed (except UNKNOWN)
- rules will allow only conjunctive premise clauses
- dynamic rule ordering
- conflict resolution based upon the following orderings (most major to least major)
  - fewest unknowns in premise first

attribute	description
a	speed of sound
A	cross-sectional area of streamtube
$\dot{m}$	mass flow rate
M	Mach number
p	static pressure
R	gas constant
T	static temperature
u	velocity
γ	ratio of specific heats
ρ	static density

Table 7.1: Initial set of attributes for the local compressible flow analysis model problem

relationship	description
$p = \rho RT$	state equation
$a^2 = \gamma RT$	speed of sound relation
$\dot{m} = \rho uA$	mass flow rate definition
$M = \frac{u}{a}$	Mach number definition

Table 7.2: Initial set of relationships for the local compressible flow analysis model problem

rule number	rule statement
1	$p = \rho RT$
2	$\rho = \frac{p}{RT}$
3	$T = \frac{p}{\rho R}$
4	$a = \sqrt{\gamma RT}$
5	$T = \frac{a^2}{\gamma R}$
6	$\dot{M} = \rho u A$
7	$\rho = \frac{\dot{m}}{uA}$
8	$u = \frac{\dot{m}}{\rho A}$
9	$A = \frac{\dot{m}}{\rho u}$
10	$M = \frac{u}{a}$
11	$u = Ma$
12	$a = \frac{u}{M}$

Table 7.3: Initial set of rules for the local compressible flow analysis model problem

- most premise clauses first
- least recently used rule first

Since there is no provision in the inference engine for algebraic manipulations, each of the relationships given in table 7.2 will have to be written as a series of rules, leading to the rule set given in table 7.3. The attributes γ and R were considered parameters whose values would be assigned *a priori* by the user and hence were not solved for in this set of rules although one could certainly do that if desired.

Note that the rules in table 7.3 are actually written in a short-hand form; a long-hand form of rule 1 might be:

```
Rule 1:  if:     p is UNKNOWN
         andif:  ρ is KNOWN
         andif:  R is KNOWN
         andif:  T is KNOWN
         then:   p ⇐ ρRT
```

Here the premise clauses ensure that the values on the right-hand side of the action clause are known and that the value to be asserted, p, is not known prior to firing the rule. The action clause states that the attribute p should be assigned a value which is computed by forming the product $\rho \cdot R \cdot T$. A rule of this form is well suited for a forward-chaining inference engine.

Unfortunately this rule would not function properly in a backward-chaining inference engine as can be seen by examining the following scenario. Suppose the current goal is p and that values are currently known for ρ and R but not for T. A backward-chaining inference engine should note that rule 1 asserts a value for p and should thus be considered for execution. However currently the value of T is UNKNOWN and so the backward-chainer should temporarily suspend its pursuit of p and should form a new goal (to pursue a value for T). With the rule as written above this would not occur because the search step would not even insert rule 1 into the conflict set because its fourth premise clause is false.

This apparent inconsistency can be resolved by examining the search steps of the forward- and backward-chainers in more detail. In the forward-chainer, the criteria which a rule must meet in order to be included in the conflict set automatically guarantee that the premises in rule 1 above are satisfied, thus making the premises redundant. Additionally if the premises are discarded, the rule would operate properly in a backward-chainer since the premise clause which requires T to be known is no longer included. In summary then, the appropriate form of the rule to implement $p = \rho RT$ is given by:

```
Rule 1:  if:
         then:  p ⇐ ρRT
```

The lack of premises in rule 1 is tied to the lack of restrictions in using the relationship $p = \rho RT$; physically the only restriction for using this rule is that the fluid is thermally perfect. If one wanted to extend the knowledge base here to possibly include non-thermally perfect gases, then there would be physical restrictions which should show up as premise clauses in the rules. As a result, rule 1 would then be written as the two rules:

```
Rule 1a:  if:    fluid is perfect
          then:  p ⇐ ρRT

Rule 1b:  if:    gas isn't perfect
          then:  p ⇐ ···
```

(Recall that the rule numbering is unimportant and is only included here for ease of reference.)

This then illustrates one of the real powers of expert systems, that is that the rules in the knowledge base should be concerned with representing the physics of the problem and not the control flow. Concerns about whether or not a rule applies to the current situation (except for physical restrictions embodied in premise clauses) are addressed by the controlling mechanism in the inference engine.

It is instructive now to examine the operation of the forward-chaining inference engine applied to the knowledge base given above in tables 7.1 and 7.3. Since the order of rule application is heavily dependent on the data, two different cases will be examined, the difference being the initial set of known facts.

In the first case, the attributes T, R, γ, M, $\dot{m}$, and A are assumed to initially have known values, yielding the locus of knowledge illustrated in figure 7.8. Facts are represented by circles containing the attribute name and rules (which are numbered) are represented by groups of vertical and

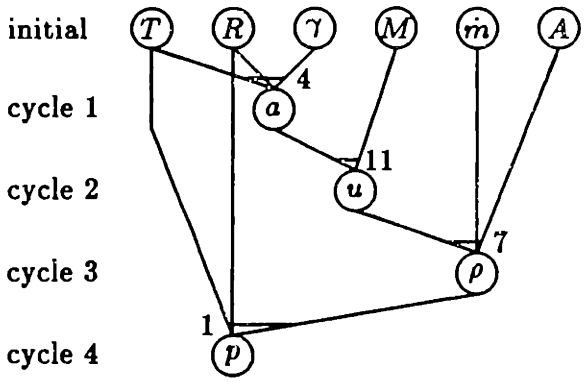


Figure 7.8: Locus of knowledge for first forward-chaining example

diagonal links between the circles. A small horizontal line spanning links indicates that the spanned links are all necessary to assert the common fact. For example the three links emanating from the top of a in the figure indicate that a can be asserted using rule 4 given known values of T, R, and γ.

The initial set of facts are shown across the top of the figure in a somewhat arbitrary order. As a result of the search step in the first cycle, only rule 4 was triggered (met the search criteria) to yield the value of a, the speed of sound. Thus after the first cycle, the parameters with known values are those shown on the top two rows of the figure. In like manner, the second cycle resulted in a value for u through the triggering and firing of rule 11. This figure clearly shows that in order to determine the value of ρ from the initially known attributes with the original set of rules, one must first compute values for a and then u.

Consider now the more complicated case which results from adding p to the initial set of knowns in the previous example. Here because the problem is overdetermined (3 unknowns and 4 physical laws), there is no unique locus

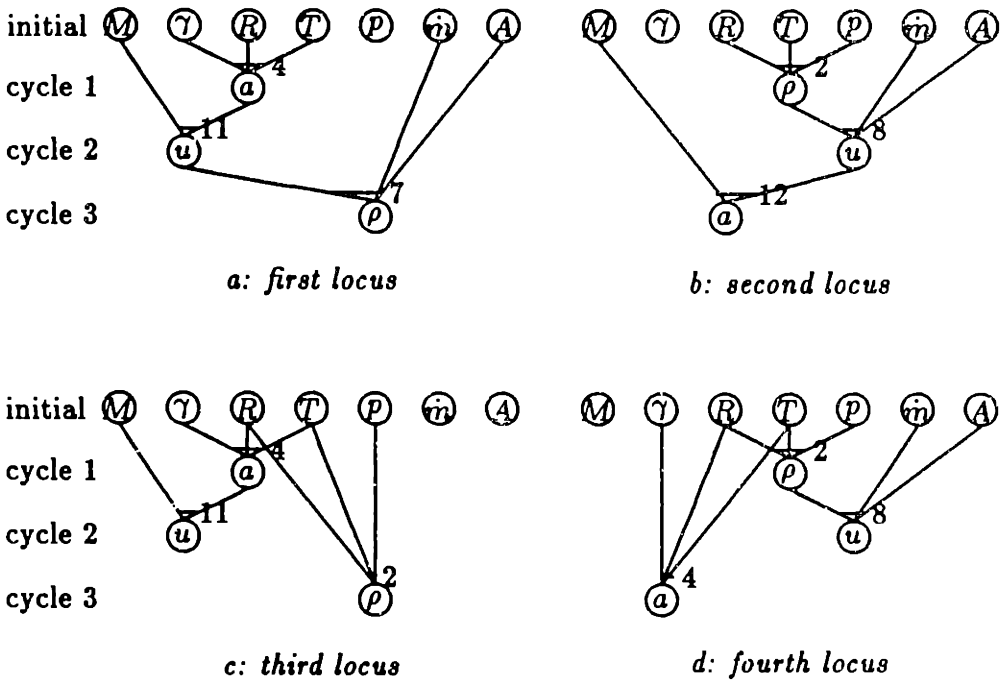


Figure 7.9: Loci of knowledge for second forward-chaining example

of knowledge but instead there are four completely independent loci as shown in figure 7.9. Also note that in part c of the figure there are actually three orders that the rules can be fired (4-11-2, 4-2-11, and 2-4-11), all of which yield exactly the same results. The particular locus of knowledge which the forward-chainer will pursue depends on the conflict resolution strategy which is chosen.

This non-uniqueness, which at first glance is disturbing, is generally not a problem as long as all the initial facts are consistent. For example in parts a and b of the figure, there are two completely independent ways to compute the density ρ (one using the definition of mass-flow rate and the other using the state equation), both of which yield the same result for a consistent set

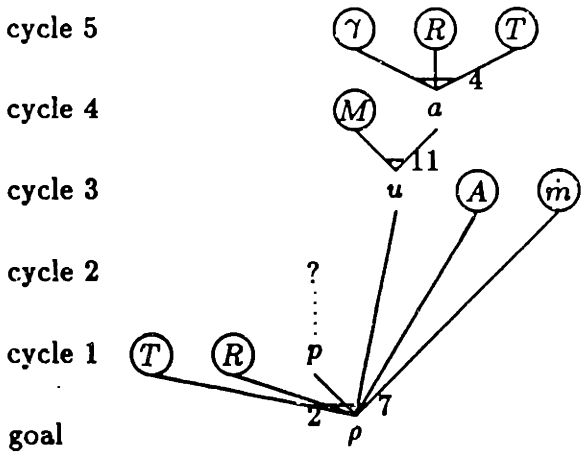


Figure 7.10: Locus of knowledge for first backward-chaining example

of initial facts.

In summary, both forward-chaining expert systems start with an initial set of facts and then use the rules to assert all possible consequent facts. The exact locus of knowledge is heavily dependent on the initial facts and to a somewhat lesser degree on the conflict resolution strategy chosen.

Consider now a backward-chaining inference engine applied to the local compressible flow knowledge base with the same initial set of known facts as in the first example above (T, R, γ, M, $\dot{m}$, and A). For an inference engine invocation with ρ as the goal, the locus of knowledge is given in figure 7.10. Here facts with known values are denoted by an attribute name in a circle, while those whose values are UNKNOWN are not circled. Rules are denoted in the same way as in previous knowledge loci.

For this case, the original goal (ρ) is shown on the bottom row of the figure. The search step in the first backward-chaining cycle finds two rules which assert a value for ρ, namely 2 and 7. Both these rules require values for three facts (for example, rule 2 requires T, R, and p), one of which is unknown (in the case of rule 2, p is unknown).

Assume that the conflict resolver chooses to try to apply rule 2 first.

Since p is needed but is currently unknown, it is added to the goal stack and becomes the new object of the search. Unfortunately the only rule which asserts a value for p is rule 1, but this rule cannot be used since one of its unknowns (ρ) is currently an active goal. Therefore the search step on the second cycle returns an empty conflict set signifying that a dead-end has been detected.

At this point, the backtracking algorithm proceeds back toward the root of the knowledge locus, searching for the last place where a choice was made as to which rule to pursue. In this case, there were two rules which assert a value for ρ and so the other (rule 7) will be pursued on the next cycle.

Rule 7 cannot be immediately fired because u is currently unknown, so u is added to the goal stack and becomes the new object of the search. There are two rules (11 and 8) which assert values for u, but the latter must be discarded since it requires a value for ρ which is an active goal. Therefore only rule 11 emerges from the search step of the fourth cycle. Again a needed value is unknown (in this case a) and so a fifth cycle commences, yielding only rule 4.

At this point all needed values for rule 4 are known and so the rule is fired, asserting a value for a. As a result, the values required for rule 11 are now known and hence it too can be fired. This process continues until the value of the original goal (ρ) is asserted and the backward-chainer stops.

It is interesting to note that even though p could not be determined above (and hence the inference engine had to backtrack), it is possible to now compute it using rule 1 from the newly computed value for ρ and the initial values given for R and T. Also, if p had been the original goal, then it is derivable as illustrated by the knowledge locus in figure 7.11. These two facts simply indicate that dead-ends in a backward-chainer do not necessarily imply that a value is unattainable, but rather that the value is unattainable using one particular path through the locus of knowledge.

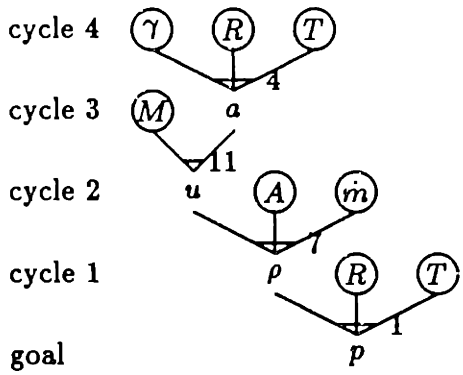


Figure 7.11: Locus of knowledge for second backward-chaining example

rule number	rule statement
13	$T_o = T\left[1 + \frac{\gamma-1}{2}M^2\right]$
14	$T = T_o\left[1 + \frac{\gamma-1}{2}M^2\right]^{-1}$
15	$M = \left[\frac{2}{\gamma-1}\left(\frac{T}{T_o} - 1\right)\right]^{\frac{1}{2}}$
16	$p_o = p\left(\frac{T_o}{T}\right)^{\frac{\gamma}{\gamma-1}}$
17	$p = p_o\left(\frac{T}{T_o}\right)^{\frac{\gamma}{\gamma-1}}$
18	$T_o = T\left(\frac{p_o}{p}\right)^{\frac{\gamma-1}{\gamma}}$
19	$T = T_o\left(\frac{p}{p_o}\right)^{\frac{\gamma-1}{\gamma}}$

Table 7.4: Rules added to the local compressible flow analysis model problem

One of the particularly nice features of expert systems is that because the knowledge is separated from the control, it is relatively easy to add new knowledge and hence new capability. For example in the local compressible flow example, stagnation pressure (p_o) and stagnation temperature (T_o) are easily added simply by defining two new attributes and by adding new rules. These new rules, which are listed in table 7.4, are derived from the energy equation and isentropic relations.

With this expanded set of rules, the initial set of facts from figure 7.8

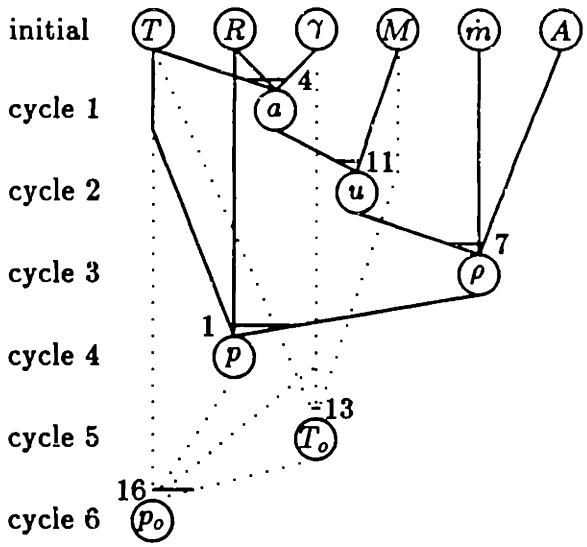


Figure 7.12: Locus of knowledge for forward-chaining example with extended knowledge base

now produces the locus of knowledge shown in figure 7.12, where the dotted lines simply denote the newly added rules. It should be noted here that the searches in cycles 1 through 4 all contained two rules in their conflict set, the rule which fire and rule 13; the firing of rule 13 was arbitrarily delayed until cycle 5 by the conflict resolution strategy.

7.4 Comparison of Expert and Traditional Systems

The previous sections described expert systems both in terms of their components and through the examination of their operation when applied in various ways to a model problem. In this section, the characteristics of expert systems are contrasted with those of traditional procedural systems in an attempt to elucidate the strengths and weaknesses of the two approaches.

The major differences between expert and procedural systems all grow out of the structural differences between the two. Recall that expert systems are separated into two parts: a knowledge base containing both facts (attributes) and relationships amongst the facts (rules), and a control strategy (inference engine) which models one of a few general problem solving strategies. On the other hand, procedural systems are broken differently into two parts: the variables (data) and the program which includes both relational and control information.

One implication of this structural difference is that because the rules in an expert system are separated from the control strategy, they can in the form of cause-and-effect statements directly represent physical laws and/or heuristics. As a result, the domain knowledge is explicitly stated in the rules, and modifications to the underlying physical model or extensions to the system's domain of expertise is more easily accomplished with an expert system than is possible with a traditional procedural system where the relationships are intermixed with control.

A side benefit here is that since the knowledge is explicitly represented in the rules, it is relatively straightforward to trace the locus of knowledge through the expert system. Sometimes this is even done automatically by including an explanation facility in the inference engine where the user can ask question such as "why is a value required for this attribute?" or "how was this conclusion reached?". The explanation facility answers these questions by tracing through the rule and/or goal stacks or by tracing through a list of rules which were fired.

Another important implication of the separation of relationships and control in an expert system is that the controller (inference engine) can directly mimic one of the problem solving strategies used most frequently by experts. Just as human experts sometimes attack a problem forward from the initial set of facts, using all appropriate physical laws and heuristics, so

too will a forward-chaining inference engine work forward, choosing the appropriate rules to develop all consequences of the initial data. On the other hand, problems in which human experts work backward through physical laws and heuristics in search of values for a particular goal are well suited to backward-chaining inference engines. Of course for efficiency, some domains require some combination of the two basic strategies (perhaps applied iteratively), resulting in inference engines which combine elements of both forward- and backward-chaining.

The important point here is that the inference engine, independent of its type, only applies those rules which are appropriate to the current situation, ignoring those in the knowledge base which are currently superfluous. Thus one can start with virtually any set of facts and a forward-chainer will adapt its application of the rules to match the given initial set. Also, a backward-chainer will adapt its search through the rules based upon the desired goal as well as the initially prescribed data. In summary, expert systems do not require *a priori* knowledge of which inputs and outputs are known.

An unfortunate consequence of this adaptiveness is that for tasks with known sets of input and outputs, expert systems are slow relative to procedural systems. This is due primarily to the search which has to be undertaken in each cycle on the inference engine. Additionally, tasks which can be described procedurally, that is tasks which follow a predefined recipe of operations and flow control based upon simple comparisons cannot be easily prescribed in an expert system since by design the rules can reference only facts and not other rules. Hence for applications which require intensive amounts of repetitive calculations which *can* be defined procedurally, the relative inefficiency of expert systems makes them unsuitable.

It should be noted again that experts and procedural systems are simply alternative strategies for solving problems. Either strategy can be used to solve any problem even though the other may be easier to implement or

may be more efficient. In general, problems which require a large amount of repetitive calculations which can be prescribed algorithmically are better suited for procedural systems; those for which the domain knowledge grows and/or those in which the types of inputs are not known *a priori* are better suited for expert systems.

7.5 A Hybrid Expert/Procedural System, EXPROC

In this section, the basic requirements of complex numerical processes, particularly those in Computational Fluid Dynamics (CFD), are examined in order to determine the relative suitability of expert and procedural systems. It turns out that neither one is completely satisfactory by itself, leading to the development of a hybrid system.

As used here, the term *complex numerical process* refers to a process which meets the following criteria:

- it is composed of a modest number of sub-processes (usually of order ten);
- its sub-processes are generally rather complicated, each requiring thousands, millions, or even more operations;
- its sub-processes are generally numeric in nature, that is they are mostly composed of simple arithmetic operations and simple logical tests;
- its sub-processes are relatively independent of each other, that is sub-processes can be viewed as black boxes with the only communication between them being through a prescribed set of inputs and outputs; and

- the order in which the sub-processes are executed is not fixed but instead changes depending on the results of the various sub-processes.

As an example of a complex numerical process, consider the design of an aircraft wing which is depicted in figure 7.13. At the center of the figure is a large pool of data, containing all geometric and aerodynamic properties which describe the wing and its operation. Data in the central pool might contain the shape of the wing (both planform and section), free-stream flight conditions, the spanwise loading distribution, and pressure distributions, amongst others. Note that values for many of the properties in the data pool may be unknown at early stages in the design process, with their values being set through the application of the various sub-processes.

Eleven of the sub-processes are shown surrounding the central data pool in figure 7.13. Included in this list are two- and three-dimensional analysis processes, processes to convert two-dimensional to three-dimensional data and vice versa, design modules, and graphics processors. Those that are listed are not meant to form an exhaustive set but rather be examples of typical sub-processes.

Consider now one of the sub-processes in detail, namely `2-D viscous analysis`. This sub-process is responsible for numerically integrating the Navier-Stokes equations (or some subset thereof) to predict the behavior of a two-dimensional section immersed in a viscous medium. Various methods have been proposed for this task, each involving the execution of billions of floating point operations as prescribed by some fixed algorithm.

As with all sub-processes, this one communicates with the central data pool via a two-way link. Here the data into the sub-process consists of local far-field boundary conditions and airfoil shape for a given section and the data out of the sub-process includes the pressure distribution, force coefficients, and the like. Additionally, this sub-process cannot communicate directly with any other sub-process but must instead deposit information

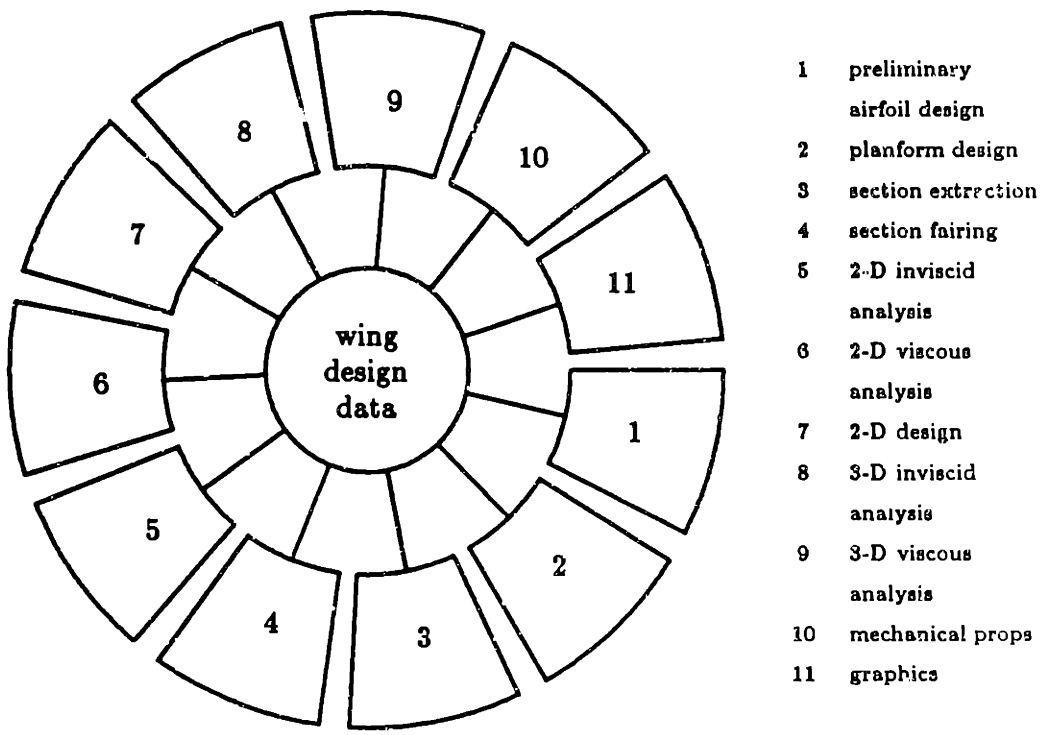


Figure 7.13: Components of a typical wing design program

into the central data pool for another sub-process to extract.

Though this communication stricture may seem overly restrictive, it leads to some substantial benefits in the design and development of the overall process. First, the function of each sub-process is completely independent of all sub-processes and thus can be developed and tested by itself. Second, because the inner workings of each sub-process is unknown to the other sub-processes, the algorithm can be changed as long as the output remain the same. In the wing design example, the viscous analysis sub-process could either integrate the full Navier-Stokes equations with some implicit scheme or could use an interacting boundary layer technique, as long as the required outputs were computed and in this way, the wing design process could grow as new techniques become available.

Consider now a scenario for the design of a wing for some aircraft. Typically the designer begins by laying-out the planform, that is the aspect ratio, taper ratio, etc., based upon some overall requirements for the wing. Next the preliminary airfoil shapes at various sections are defined according to some correlations based upon local flow properties and geometric constraints. The designer may then analyze the two-dimensional sections in order to update the spanwise loads needed in the preliminary airfoil design or may directly run a three-dimensional simulation after first fairing the sections.

The exact sequence of sub-processes which a designer uses cannot be prescribed *a priori* but instead is highly dependent on the data which is output from the prior sub-processes. Economical, political, and technological concerns all impact the choice of sub-processes which the designer chooses. For example, three-dimensional viscous analyses are currently prohibitively expensive and may only be used in the most severe of cases. Similarly, though two-dimensional analyses are relatively inexpensive, their use is somewhat limited by the assumptions made in their development.

Very often the choice of which sub-process to employ is heuristic in nature. For example, the rule "in general, the two-dimensional approximation is reasonably good outboard of the engines but inboard of 90 percent span on subsonic commercial transports, except when shocks are present" might lead a designer to rely heavily on the two-dimensional analysis for a particular design. But if the two-dimensional analysis indicates that a shock is present, then the designer may have to resort to the costlier three-dimensional techniques. Similarly, the designer may use a rule which states that "if the budget is tight or the design is needed immediately, then use approximate techniques".

The development of a complex numerical process such as the wing design example above can be accomplished in a variety of ways; here two competing techniques will be explored: procedural and expert systems.

First consider the development of a complex numerical process using a traditional procedural system. For the execution of the algorithmically-based sub-processes, the procedural system will be highly efficient. This is primarily due to the fact that procedural systems compile directly into machine instructions which more or less directly mimics the algorithm. For example, an algorithm to compute the inner product of two vectors is translated into machine instructions which loop over all elements of the input vectors, accumulating the sum of the products of the vectors' elements. There is very little overhead associated with the looping operation.

At the same time, the data-sensitivity necessary to properly determine the order of execution of the sub-processes is very hard to implement well in a procedural system. For a few sub-processes and a few possible conditions, this can be handled with some IF-tests and GOTOs. However as the number of sub-processes and possibilities increases, the logic quickly become unmanageable, often making extensions nearly impossible to implement. Many large systems have been rendered inoperative (at least temporarily)

by adding only one minor new feature somewhere else in the system because the reasons why and when certain sub-processes were executed were obscured by the logical structure of the controller.

Second, consider the development of a complex numerical process using an expert system. Here the controlling logic concerning the execution order of the various sub-processes is very easy to implement and extend through the use of expert system rules. The logic is easy to decipher because it is represented explicitly in the rules.

Unfortunately the execution of the algorithms in the sub-processes is considerably less efficient in an expert system than in a procedural system, mostly due to the search step in expert systems. For example, an inner product calculation by an expert system would require that the inference engine search for all element pairs in order to form the products which need to be summed rather than using a simple loop. In addition, the search step results in calculations which are nearly impossible to vectorize on modern computers, yielding even more inefficiency.

Clearly neither procedural nor expert systems alone are completely suitable for complex numerical processes and hence a new hybrid system is proposed here, hereinafter call EXPROC (for EXpert/PROCedural system). The idea to combine procedural and expert elements is not new; several other researchers have also used this approach, three of which are described here.

The first of these is the work of Tong[83,82] in which he uses an expert system to automate the design process for cooling fans. His system basically consists of two components: a conventional system to analyze a proposed geometry and an expert system to alter the geometry based upon the observed performance of the previous geometry and heuristic design rules gleaned from a human design expert. Successive iterations between these two components results in the evolution of an improved cooling fan.

The second example is the PAN AIR knowledge system by Conner and Purdon[26]. Here there are two major components: an expert system with which a user consults in order to set up a proper input file for the second component, the PAN AIR program which is written using conventional techniques.

The third example is a zonal grid generation system discussed briefly by Andrews[6]. Here an expert system is used to divide complex two-dimensional flow fields into four-sided, well-shaped zones; a grid generation program (which uses conventional programming techniques) then generates the final computational grid within each zone.

In each of these systems, the conventional and expert system components are rather loosely coupled, that is there is a limited interaction between the two components. Conner et al and Andrews apply the two components successively with the expert system performing the initial processing and then conventional components acting on the expert system's outputs. Even in Tong's work, where he alternately applies the two types of components, the components act somewhat independently.

Here, the combination of conventional and expert system components is more tightly coupled, as shown in figure 7.14. This new hybrid system architecture gains its power by combining the advantages of each type of component at a lower level.

EXPROC is composed of three parts (the central data pool, the procedures, and the expert system), each of which will be described in turn.

At the center of the figure is the data pool which contains all information which is to be shared amongst the procedures. Typically the data consists of some global parameters which describe the current problem as well as arrays which describe distributions. In the wing design example from above, the global parameters might include the free-stream conditions, the wing area and weight, and the aspect ratio; the wing shape, surface pressure

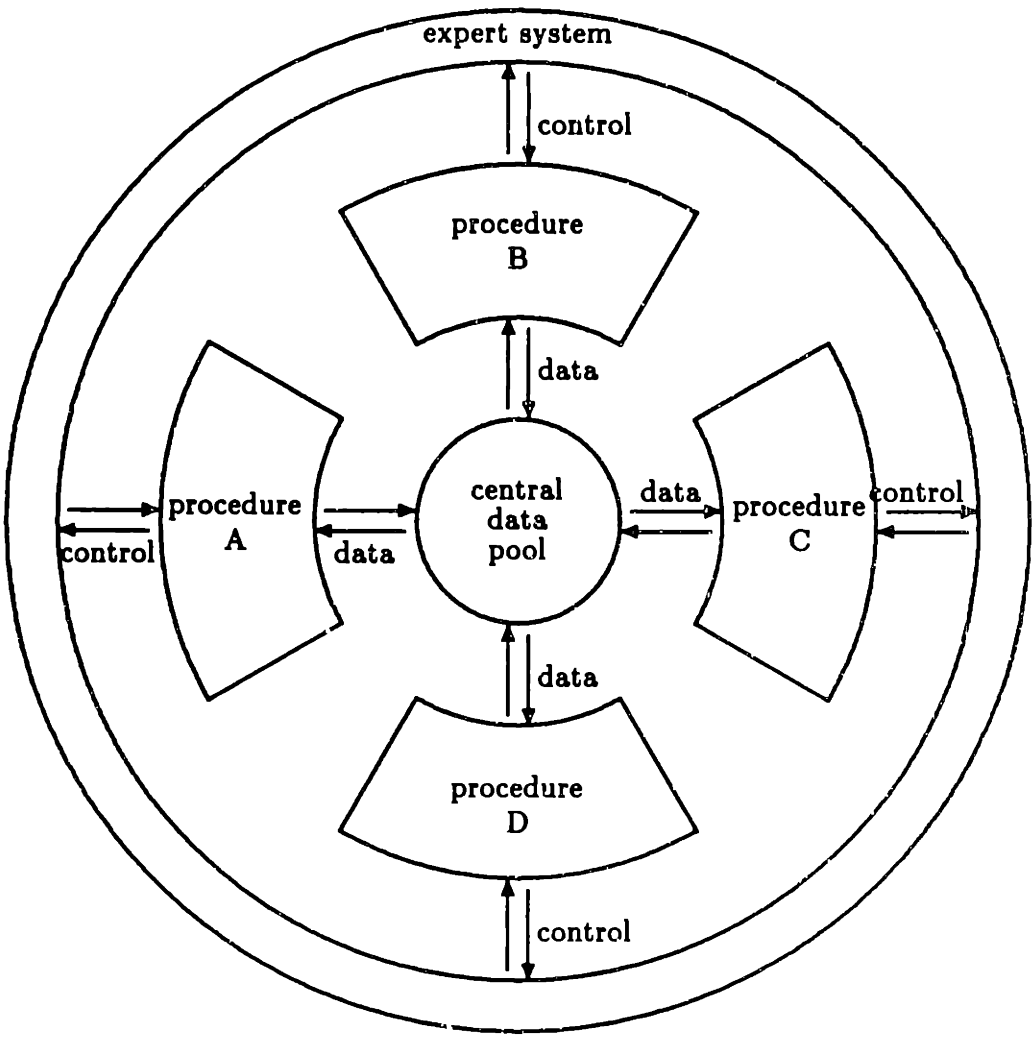


Figure 7.14: Organization of EXPROC

distributions, and spanwise loading and upwash distributions are examples of data which would be contained in arrays in the data pool.

Additionally, the data pool might contain some working arrays which may be used for inter-process communication. For example in the wing case, all the data may be stored in three-dimensional arrays. In order to perform a two-dimensional analysis, the `extract section` sub-process may need to be called first to set up the two-dimensional work arrays and upon completion of the analysis another process (`section fairing`) may be needed to insert the two-dimensional results into the three-dimensional arrays.

Surrounding the data pool is a small number of independent procedures which are generally both numerical and algorithmic in nature, making them ideal candidates for procedural system techniques. In the wing design example, these may include two- and three-dimensional flow analyses and graphics processors. Note that some of these procedures (sub-processes) may themselves be complex enough so as to be considered complex numerical processes which in turn can be handled effectively with EXPROC.

The final component of EXPROC is the expert system (shown in figure 7.14 as a ring encircling the procedures), the purpose of which is to control the flow of execution through the system. As such it passes control information into and receives status information out of the various procedures.

In EXPROC, the expert system represents facts as attributes:value pairs, where the values are either numeric or UNKNOWN; certainty factors are not supported. Rules contain any number of conjunctive premise clauses of the form

```
attribute-l  .comparator.  attribute-r
```

where the `comparator` can be any one of the algebraic relational operators. Rules also contain any number of action clauses which are executed sequentially when the rule is fired. Pre-defined actions include all the arith-

metic operators and many algebraic operators (root extraction, exponentiation, etc.). Also the action clauses can specify that one of the procedures (described above) be executed. In the wing design example, some rule's action clause might call the `two-dimensional inviscid analysis` procedure with some specified level of accuracy (specified by some attribute); on return the procedure might return a status flag to tell the expert system whether or not the calculation was successful.

Either a forward- or backward-chaining inference engine can be employed in EXPROC, where the forward-chainer can have either static or dynamic rule ordering. Although EXPROC does not contain an explanation facility, full flow tracing is available if desired so that the locus of knowledge can easily be traced by the user. Conflicts are resolved through four successive orderings, where the three most major can be chosen by the author of the knowledge base from:

- highest priority rule dominates;
- most true premises dominates;
- most recently fired rule dominates;
- least recently fired rule dominates;
- fewest unknowns first (backward-chaining only).

The fourth ordering (the final tie-breaker) is that the rules are selected in the order that they were defined. Additionally context limiting is employed so that only a subset of the rules need be examined in each search.

The EXPROC language reference guide, which is contained in Appendix D, includes a general overview of EXPROC and its language, a discussion of the language elements (attributes and contexts), and a complete description of the format and meaning of the various types of EXPROC statements. It

also contains a listing of the local compressible flow knowledge base used in the model problems discussed in the previous two sections.

Both the flexibility and efficiency of the hybrid EXPROC compare favorably with that of expert and procedural systems when applied to complex numerical processes. On the one hand, since the majority of computer resources (CPU time) are consumed by the execution of the sub-processes, the efficiency of EXPROC is essentially the same as the efficiency of procedural systems and much greater than that of an expert system. On the other hand, since the control of the execution order of the sub-processes is governed by rules in the expert system, the flexibility and data sensitivity of EXPROC is much the same as an expert system and much greater than is possible with a procedural system.

The structure of the hybrid system developed here is different from the hybrid systems of Tong, Conner and Purdon, and Andrews in one important respect — the level at which the expert and conventional components are coupled. In the other three systems, the components are coupled at a very high level, that is a brief conceptual block diagram of the data and processing flows through these hybrid systems would include a very small number of blocks which were conventionally programmed and others which included expert systems. The two types of components are of roughly equal stature in that each contains processing and control functions.

On the other hand, the hybrid system developed here is coupled at a much lower level; the conventional procedures are responsible for all CPU-intensive functions and the expert system serves a purely supervisory function, that is it controls the order of execution of the various processes based upon status information passed back from the processes to the expert system. This separation of processing and control is a major benefit of the new hybrid system approach.

7.6 Application of EXPROC to Grid Adaptation

Thus far, this chapter has been concerned with a general description of expert systems and subsequently the development of the hybrid system concept, EXPROC. The remainder of the chapter simply establishes the "hooks" necessary to combine the MITOSIS grid adaptation program into EXPROC.

The architecture of the adaptive grid program MITOSIS is shown in figure 7.15. As for the general hybrid system (figure 7.14), the central data pool forms the hub of the system, around which ten procedures which EXPROC calls directly are listed. These procedures, which only form a subset of all the procedures available to EXPROC, are described below.

To avoid confusion, more precise definitions of EXPROC and MITOSIS are presented. EXPROC refers to: the hybrid system approach to solving complex numerical processes which was developed in the previous section; an expert system (forward- and backward-chaining inference engines and a complier) which was written specifically to support programs which employ the hybrid system approach; and the language in which knowledge bases used by EXPROC are coded. In contrast, MITOSIS refers to a specific complex numerical process (two-dimensional adaptive grid calculations of the Euler flow equations) which uses the EXPROC hybrid system approach and the EXPROC expert system; MITOSIS' knowledge base is written in the EXPROC language.

This section begins with a description of MITOSIS' central data pool which is composed of three COMMON blocks: one for the independent and dependent variables and the associated data structure (G2COMN), one for the variables associated with the Euler equation integrator (E2COMN), and one for the feature finder (F2COMN). This is then followed by a detailed description of the procedures which are specific to grid adaptation

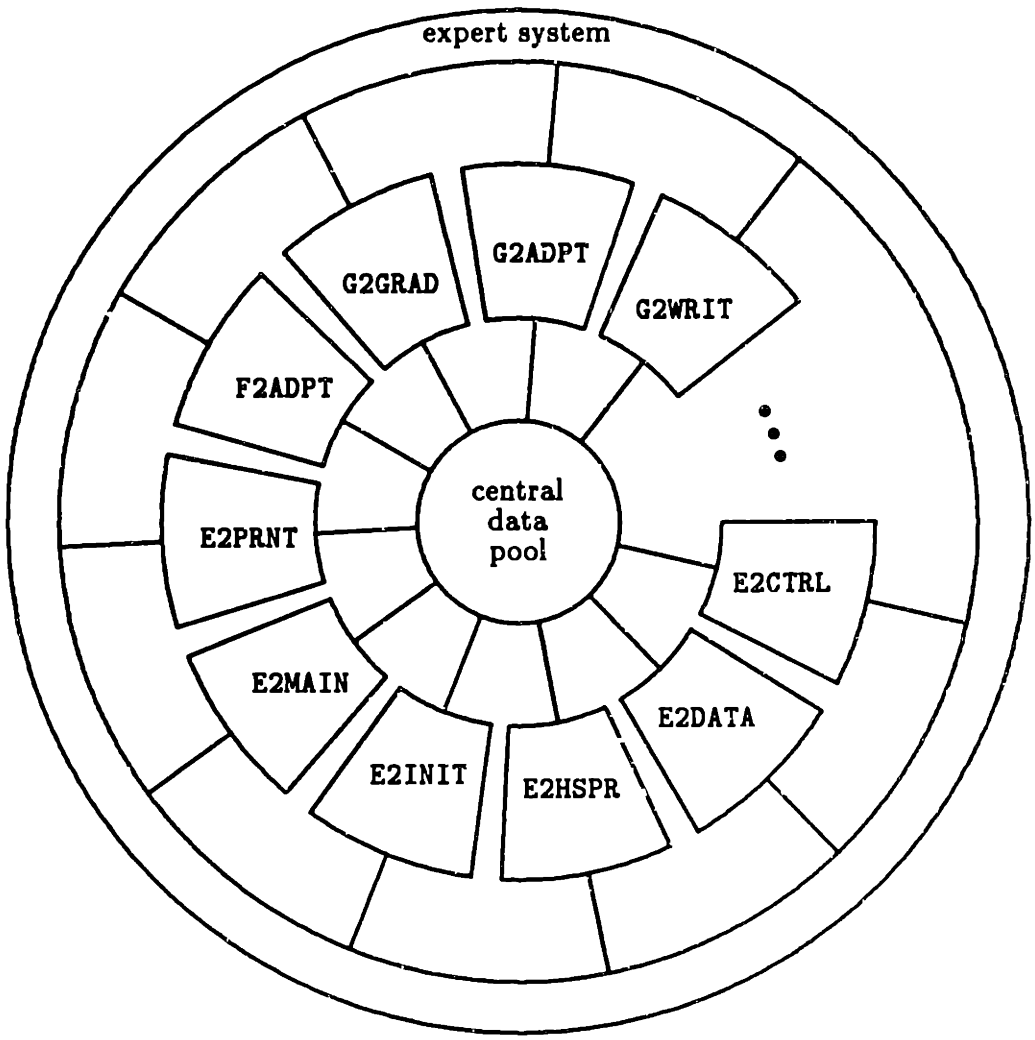


Figure 7.15: Organization of MITOSIS

and which are directly callable by EXPROC through action clause references. As is the case with the central data pool, these are broken into three groups: E2-routines which are concerned with the time integration of the two-dimensional Euler equations, F2-routines which are concerned with locating features, and G2-routines which are concerned with grid generation, cell division and fusion, and other data structure management chores.

7.6.1 Central data pool

As stated above, the central data pool is comprised of data stored in three COMMON blocks which are fully documented in Appendix C. Here a brief description of each will be given:

G2COMN — this COMMON block contains data which is used by E2-, F2-, and G2-procedures. It consists mostly of arrays, the contents of which can be summarized as

- nodal arrays containing the current values of all the independent and dependent field variables
- data structure arrays which define the logical structure of the grid through a series of pointers
- boundary arrays containing pointers to boundary nodes and cells as well as specific information required to impose boundary conditions (such as surface slope)
- a working array (same length as node arrays) which various procedures use as the source/target of their inputs/outputs

E2COMN — this COMMON block contains data which is only used by E2-procedures. It consists of both scalars and arrays which can be summarized as

- problem parameters such as reference values for the dependent variables, the ratio of specific heats, etc.

- solver parameters such as CFL-number and smoothing coefficients
- convergence parameters such as iteration number, normalized changes in the dependent variable for the last iteration, and force coefficients
- timer parameters used in the analysis of subroutine efficiencies
- nodal arrays storing temporary solver data such as the smoothing coefficient or changes in the dependent variables at each node
- control and index vector arrays used by the solver for efficient vector operations

F2COMN — this COMMON block contains data which is only used by F2-procedures. It consists of both scalars and arrays which can be summarized as

- parameters used in threshold determination
- arrays containing the refinement singularity locations and radii
- nodal arrays containing temporary variables used in gradient and Laplacian calculations

7.6.2 Procedures

The procedures which are specific to MITOSIS and which can be directly called through reference in a rule's action clause are summarized here. For each procedure, a brief description of its purpose is accompanied by a discussion of the input parameters which must be passed to it from the expert system as well as the output status information which the procedure returns to the expert system. Many of the routines are service routines and therefore require no input from and returns no status information to the expert system.

E2CTRL — this procedure is used to initialize the Euler solver whenever the grid structure changes. It uses information in the data structure to construct control and index vectors which are used throughout the Euler solver. Additionally, it takes as input parameters the levels of maximum residual change and change in force coefficients which are used to determine convergence on any one grid. There is no status information returned to the expert system.

E2DATA — this procedure is used to fill the work array in G2COMN with the flow-field variable specified by the input parameter. There is no status information returned to the expert system.

E2FORC — this procedure computes the force coefficients and stores them in E2COMN. A service routine.

E2HSPR — this procedure outputs the convergence parameters to the terminal and listing files. A service routine.

E2INIT — this procedure is used to interactively change the solver and/or problem parameters which are stored in E2COMN. A service routine.

E2MAIN — this procedure integrates the Euler equation forward in pseudo-time one full multiple-grid cycle, including all embedded regions. It uses no input parameters but returns to the expert system the maximum (over all nodes) change in the second dependent variable, the lift and drag coefficients.

E2PRNT — this procedure outputs to the listing file a table of the flow-field properties at all solid surface boundary points. The table entries include both independent variables, all four dependent variables, as well as a few derivable flow-field quantities. A service routine.

E2STAT — this procedure writes grid statistics such as the number of fine cells on each multiple-grid level and the sparsity of the index

vectors to a unit specified by the input parameter; it returns no status information.

E2TIME — this procedure writes a table of the CPU time consumed in various parts of the Euler integrator to a unit specified by the input parameter; it returns no status information.

E2VRFY — this procedure is used to verify the global conservation properties of a computed solution by integrating around closed loops; the results of the integration are written to the listing file. It takes three input parameters to describe the integration paths to be used and returns no status information.

E2WIND — this procedure uses a sliding window technique to check for convergence and divergence, using an input parameter which it maintains to tell which part of the algorithm is currently active. It returns to the expert system a status flag indicating if convergence has been detected, divergence has been detected, or neither has been detected. It uses as limits parameters inserted into E2COMN by the procedure E2CTRL (described above).

F2ADPT — this procedure is used to adapt the grid. It begins by inserting the refinement parameter into the work array in G2COMN and then raising the value within all the singularity parameters. It then calls G2ADPT, using division and fusion thresholds passed to it through input parameters. It returns to the expert system the number of cells which were either divided or fused.

F2DATA — much like E2DATA, this procedure is used to fill the work array in G2COMN, but this time with a feature finder variable as directed by the input parameter. There is no status information returned to the expert system.

F2GRAD — this procedure computes the first-difference of the variable currently in the work array in G2COMN, depositing the result in one of the F2COMN arrays. A service routine.

F2INIT — much like E2INIT, this procedure allows the user to interactively change parameters used in the thresholding algorithm. A service routine.

F2LAPL — this procedure computes the second-difference of the variable currently in the work array in G2COMN, depositing the result in one of the F2COMN arrays. A service routine.

F2PRNT — this procedure writes a table of the arrays in F2COMN onto the listing unit. A service routine.

F2THRS — this procedure uses information deposited in F2COMN by either F2GRAD, F2LAPL, or F2VALU as well as input feature finder parameters to determine the division and fusion thresholds, both of which it returns as status information. It requires no input parameters from the expert system.

F2VALU — this procedure copies the variable currently in the work array in G2COMN into one of the F2COMN arrays. A service routine.

G2ADPT — this procedure uses division and fusion thresholds provided and input parameters by the expert system to adapt the grid. It uses the variable currently in the work array in G2COMN as the refinement parameter. It returns as status information the number of cells which were modified. Normally this procedure will not be called directly from the expert system, but instead will be invoked through an expert system call to F2ADPT.

G2DIVD — this procedure divides the cell specified by the input parameter, returning a status word which indicates if the division was successful or not.

G2EXAM — this procedure allows the user to interactively examine and modify the independent variables, dependent variables, or data structure entries. A service routine.

G2FUSE — this procedure fuses the cell specified by the input parameter, returning a status word which indicates if the fusion was successful or not.

G2GRBG — this procedure compacts the data structure after either cell division or fusion and should be invoked before any other procedure uses the data structure. This procedure is called automatically at the end of G2ADPT (and hence F2ADPT). A service routine.

G2GROW — this procedure is used to grow embedded regions. It takes as input the amount that the embedded regions should be grown and returns as status information the number of cells which were modified.

G2PRNT — this procedure writes selected data structure information out to the listing file. An input parameter selects which part(s) of the data structure should be printed. This procedure returns no status information.

G2READ — this procedure reads in a new data structure (including independent and dependent variables) from a unit defined by an input parameter. It returns no status information to the expert system.

G2SUMY — this procedure writes a data structure summary to the unit specified by the input parameter; no status information is returned.

G2SURF — this procedure divides all cells in the vicinity of surface nodes which have boundary condition types specified by the input parameters. This procedure returns the number of divided cells.

G2VOID — this procedure checks for embedded regions which result either in voids or islands, and adjusts the embedded region so as to eliminate them. It takes no input parameters but returns the number of modified cells.

G2WRIT — this procedure dumps a copy of the data structure (including independent and dependent variables) to a unit defined by the input parameter. The file is written in a form that is readable by G2READ. This procedure does not return any status information.

Chapter 8

Grid Adaptation

In previous chapters, all of the components of an adaptive grid procedure were independently developed; this chapter integrates those components into one adaptive grid procedure, the MITOSIS program. The chapter begins with a summary of the grid adaptation strategy, which is then translated rather rigorously into a knowledge base which serves to control the grid adaptation process. Very few details have been left out of this discussion and the reader who is interested mainly in grid adaptation can skip (or skim) those sections.

The operation of the adaptive grid program and its knowledge base are then traced (both in physical terms and in terms of the expert system) by following the evolving solution for the primary test case (AGARD-06). The final adapted solution for this case is then compared with a globally refined solution, both in terms of accuracy and efficiency. Unfortunately, these solutions differ from the published solution and hence a study was conducted to determine the source of that discrepancy.

Once the discussion of the primary test case is complete, the utility of a hybrid system approach (EXPROC) is demonstrated by applying MITOSIS and its initial knowledge base to test cases with differing complex flow-field topologies; various deficiencies of the basic strategy are identified and

remedied by the trivial addition of rules to the knowledge base.

The chapter concludes by demonstrating the usefulness of a semi-structured grid for computing solutions on domains which are not logically rectangular.

8.1 Review of Basic Adaptive Grid Strategy

As mentioned in the Introduction, the basic approach of the adapted, embedded mesh procedure developed herein is to integrate the governing equations to steady state on successively refined grids until solutions on successive grids are "close" to each other, indicating grid resolution independence of the final solution. This is accomplished through the following steps:

1. Set up the fixed global grid using an elliptic grid generator.

2. Integrate the governing equations to convergence (steady state) on the current grid, using Ni's multiple-grid accelerator both to hasten convergence and to link solutions on the global and embedded regions.

3. If the latest solution is "close" to the previously computed solution, then STOP.

4. Locate pertinent features. This is accomplished by computing one or more refinement parameters at each computational node as well as division and fusion thresholds.

5. Refine the grid at all features and un-refine (fuse) the grid away from features.

6. Loop back to step 2.

Typically this loop is repeated fewer than ten times, each one requiring up to several hundred pseudo-time steps of the integrator.

8.2 Development of a Knowledge Base

In this section, the above adaptive grid strategy is developed into a knowledge base for use by a controlling expert system such as in the hybrid MITOSIS program. Recall that knowledge bases are comprised of two components, facts and rules, both of which are described in turn.

Recall too that within EXPROC, facts are stored as attribute:value pairs, with the attribute being some identifying name and the value being any real number or the value `UNKNOWN`. The attributes used within MITOSIS' knowledge base can be broken into two groups, constants and variables. The following three sections, which define the various attributes and rules in detail, is included for completeness; the reader may wish to skim them and refer back to the various definitions when needed.

8.2.1 Constant attributes

The constant attributes, which are somewhat analogous to *parameters* in FORTRAN, can be used both in a premise clause and as an input parameter of any action clause. These constant attributes generally serve the purpose of imposing limits or acting as a pointer to an input/output unit. The constant attributes are given in the following list, with the format of each entry being `attribute:initial_value` followed by a brief description of the attribute's meaning.

`terminal`:6.0 — the FORTRAN unit associated with the terminal, needed so that status information and summaries can be directed to the user interactively (or to the log file for BATCH running).

`printer`:7.0 — the FORTRAN unit associated with the printer (listing file), needed so that status information and summaries can be output.

`dump_file`:19.0 — if divergence is detected, a solution file is dumped to FORTRAN unit `dump_file`.

`final_soln`:20.0 — at final convergence, a solution file is written to FORTRAN unit `final_soln`.

`nbor_out`:2.0 — the FORTRAN unit number to which the neighbor table is written.

`nbor_in`:−2.0 — the FORTRAN unit number from which the neighbor table is read.

`density`:11.0 — the code in E2DATA to denote that the density should be dumped into the work array.

`max_iters`:1000.0 — the maximum number of allowable iterations on any one grid.

`max_cycle`:5.0 — the maximum number of allowable adaptation cycles.

`clift_tol`:0.010 — in order to achieve final convergence, the lift coefficients on two successive grids must be within `clift_tol` of each other.

`cdrag_tol`:0.002 — in order to achieve final convergence, the drag coefficients on two successive grids must be within `cdrag_tol` of each other.

`du_tol_in`:0.001 — the sliding-window convergence scheme commences when the maximum change in the second dependent variable (the x-momentum) in the Euler integrator is less than or equal to `du_tol_in`.

`cl_tol_in`:0.002 — for purposes of convergence checking on any grid, the minimum and maximum lift coefficients in the sliding-window must be within `cl_tol_in` of each other.

`cd_tol_in`:0.001 — for purposes of convergence checking on any grid, the minimum and maximum drag coefficients in the sliding-window must be within `cd_tol_in` of each other.

rpnummax:1.0 — the number of simultaneous refinement parameters.

rptable:11.0 — a table (of length rpnummax) of variable codes (as defined by E2DATA) for the various variables which are differenced to become refinement parameters.

cttable:1.0 — a table (of length rpnummax) of the order of the differences used in defining the refinement parameters.

8.2.2 Variable attributes

The attributes whose values vary through the firing of the expert system rules are described in the following list, the format of which is the same as above. These attributes can be used anywhere in premise and action clauses and therefore many do not have initial values assigned to them.

out_file:12.0 — after convergence has been achieved on each grid, out_file is incremented and the current solution is written to FORTRAN unit out_file.

cycle:0.0 — a counter which keeps track of the number of times that the grid has been adapted.

clift_old:1000.0 — the lift coefficient the previous time a solution converged. Since this is used to determine if the solutions on successive grids are essentially the same, this attribute is initialized with a ridiculous value so that final convergence will not be prematurely signalled after the initial grid.

cdrag_old:1000.0 — the drag coefficient the previous time a solution converged. Since this is used to determine if the solutions on successive grids are essentially the same, this attribute is initialized with a ridiculous value so that final convergence will not be prematurely signalled after the initial grid.

iters:UNKNOWN — the number of iteration of the Euler integrator so far on this grid.

conv_stage:UNKNOWN — the current stage of the convergence-checking algorithm. This attribute which is used and maintained by E2WIND must be set to 0.0 to initialize E2WIND on each grid.

conv_check:UNKNOWN — an attribute used as the output of E2WIND to signal the convergence status of the Euler integrator. conv_check=+1.0 denotes convergence has been detected, conv_check=−1.0 denotes divergence has been detected, and conv_check=0.0 denotes neither has been detected.

delta_du:UNKNOWN — the maximum change in the second dependent variable (x-momentum) in the previous step of the Euler integrator.

clift:UNKNOWN — the lift coefficient in the previous step of the Euler integrator.

cdrag:UNKNOWN — the drag coefficient in the previous step of the Euler integrator.

delta_cl:UNKNOWN — the difference between the lift coefficients on two successive cycles.

delta_cd:UNKNOWN — the difference between the drag coefficients on two successive cycles.

rpnum:UNKNOWN — the number of the refinement parameter currently being computed.

rp:UNKNOWN — the variable code (as defined by E2DATA) for the refinement parameter currently being computed.

`ct:UNKNOWN` — the difference type (as defined by F2CHNG) for the refinement parameter currently being computed.

`thr_divd:UNKNOWN` — the grid adaptation division threshold.

`thr_fuse:UNKNOWN` — the grid adaptation fusion threshold.

`ncells:UNKNOWN` — the number of cells which were marked by the previous adaptation.

`ncella:UNKNOWN` — the number of cells which were actually changed by the previous adaptation.

`ncellg:UNKNOWN` — the number of cells which were actually changed by the addition of a buffer.

8.2.3 Rules

Both to facilitate the development of the knowledge base and to improve the efficiency of the forward- and backward-chainers, the grid adaptation process has been logically broken into contexts or sub-tasks. Here, the seven contexts (`initial`, `integrate`, `converged`, `diverged`, `adapt_i`, `adapt_f`, and `return`), represent various parts of the grid adaptation solution process. This division of the entire task into groups is very similar to the common technique of subdividing a procedural program into a number of sub-programs or subroutines.

The interconnections of the contexts are shown in figure 8.1; a description of the purpose of each context as well as its rules follows. The format for these rules follows the EXPROC language syntax which is described in Appendix D.

`initial` — Each time that a new grid is established, the various initializations which are required are performed by the rule(s) in the context

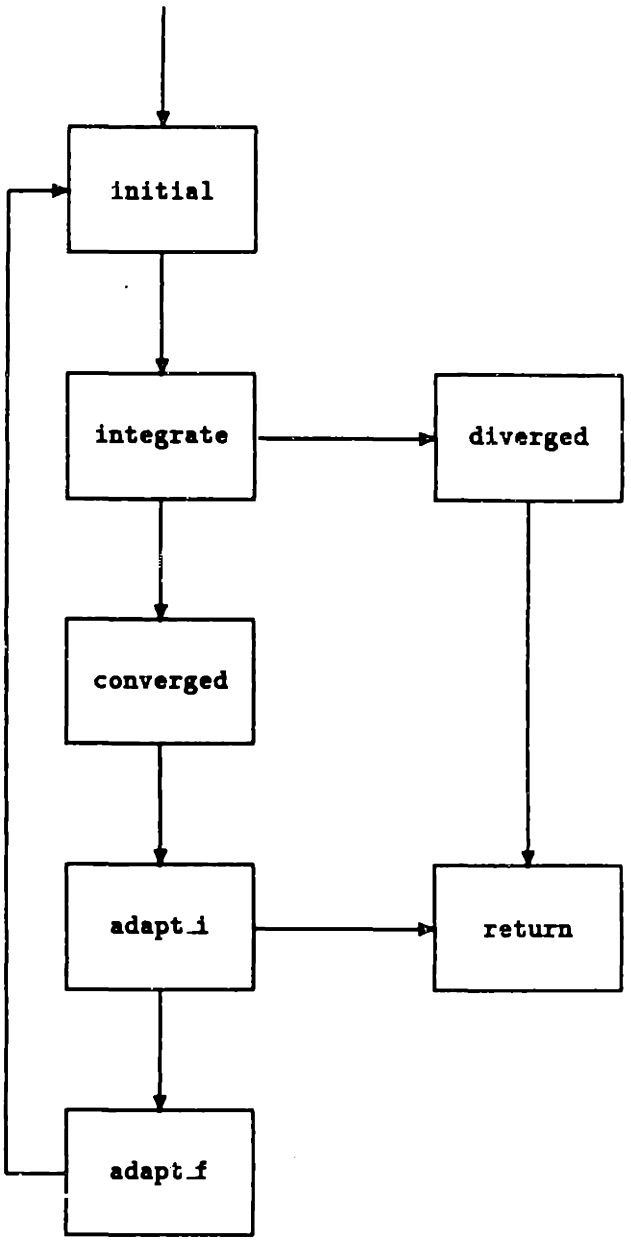


Figure 8.1: Interconnection of contexts in MITOSIS

initial. Since by default both EXPROC's forward- and backward-chainers begin with this context, one of the rules in this context is always the first to fire.

Currently there is only one rule in this context, given by:

```
RULE      SET CONTROL AND INDEX VECTORS
CONTEXT   initial
IF        * NOP
THEN      G2SUMY terminal
ANDTHEN   G2NBOR nbor_out
ANDTHEN   E2CTRL du_tol_in cl_tol_in cd_tol_in
ANDTHEN   E2STAT printer
ANDTHEN   SET %iters +0.0
ANDTHEN   SET %conv_stge +0.0
ANDTHEN   SET %conv_chck +0.0
ANDTHEN   CONSIDER integrate
```

This rule does not have any premise clauses (notice that * NOP is only a place-holder) and thus is available to fire any time that the context is initial. The actions performed by this rule are: write the neighbor table out to disk storage (G2NBOR), set up the control and index vectors for the Euler integrator (E2CTRL); output the grid and solver status information (G2SUMY and E2STAT); and initialize the attributes iters, conv_stage, and conv_check. The rule concludes by switching the current context to integrate.

integrate — The rules in this context are associated with integrating the Euler equations to steady state on a given grid. For the four rules in this context to function properly together, it is assumed that specificity conflict resolution is employed; that is the rule with the greatest number of premise clauses dominates.

The first rule in this context:

```
RULE      * TAKE A TIME STEP
CONTEXT   integrate
IF        * NOP
THEN      E2MAIN %delta_du %clift %cdrag
ANDTHEN   E2WIND %conv_stge %conv_chck
ANDTHEN   E2HSPR
ANDTHEN   ADD %iters iters +1.0
```

is the one which calls E2MAIN to integrate the Euler equations one iteration. It then calls the sliding-window convergence checking algorithm (E2WIND), outputs convergence information for the iteration just taken (E2HSPR), and increments the number of iterations taken on this grid. Notice that this rule has no premise clauses and is thus available for execution anytime the context is `integrate`.

The second rule in this context:

```
RULE      CONVERGENCE DETECTED
CONTEXT   integrate
IF        conv_chck .EQ. +1.0
THEN      CONSIDER converged
```

is responsible for noting that the Euler integrator has converged on the current grid by monitoring the value of `conv_check` which is updated each time the rule discussed previously is fired. Upon noting convergence, this rule switches the current context to `converged`. This rule will never be in the conflict set by itself since the previous rule will be in the conflict set anytime the context is `integrate`; therefore for this rule to perform properly it must dominate over the previous rule through a conflict resolution strategy such as specificity.

The third rule in this context is:

```
RULE      DIVERGENCE DETECTED
CONTEXT   integrate
IF        conv_chck .EQ. -1.0
THEN      CONSIDER diverged
```

This rule, which switches the current context to diverged upon noting the appropriate value of conv_check, is exactly analogous to the rule described above which notes convergence.

The final rule in this context:

```
RULE      OUT OF ITERATIONS
CONTEXT   integrate
IF        iters .GT. max_iters
THEN      CONSIDER diverged
```

is responsible for ensuring that no more than max_iters iterations of the Euler integrator are taken on any one grid. Upon noticing such a situation, this rules fires, changing the current context to diverged.

converged — This context is concerned with the processing required after convergence has been achieved on the current grid. This context currently has only one rule, given by:

```
RULE      * CONVERGENCE BOOKKEEPING
CONTEXT   converged
IF        * NOP
THEN      E2HSPR
ANDTHEN   E2TIME printer
ANDTHEN   G2NBOR nbor_in
ANDTHEN   SHOW clift cdrag
ANDTHEN   ADD %out_file out_file +1.0
ANDTHEN   G2WRIT out_file
ANDTHEN   SUBTRACT #1 clift_old clift
ANDTHEN   ABS %delta_cl #1
ANDTHEN   SET %clift_old clift
ANDTHEN   SUBTRACT #1 cdrag_old cdrag
ANDTHEN   ABS %delta_cd #1
ANDTHEN   SET %cdrag_old cdrag
ANDTHEN   CONSIDER adapt_i
```

This rule, which does not contain any premise clauses, begins by outputting various status information such as the residual information of the latest integration step (E2HSPR), a summary of the CPU time spent in various parts of the Euler integrator (E2TIME), and the final values of `clift` and `cdrag`. It reads the neighbor table from disk (G2NBOR) and increments `out_file` and writes a solution file to that unit. Finally it computes the differences in the lift and drag coefficients between the current grid's solution and the previous grid's solution. The current context is then set to `adapt_i`.

`diverged` — This context is concerned with the processing required after it has been noted that the Euler integrator has diverged on the current grid. The one rule currently in this context is:

```
RULE      DIVERGED SOLUTION DUMPED --- ABORT
CONTEXT   diverged
IF        * NOP
THEN      E2HSPR
ANDTHEN   E2TIME printer
ANDTHEN   G2NBOR nbor_in
ANDTHEN   G2WRIT dump_file
ANDTHEN   CONSIDER return
```

This rule simply writes out status information (E2HSPR and E2TIME), reads in the neighbor table (G2NBOR), and writes the current (diverged) solution to unit `dump_file`, and switches the context to `return`.

`adapt_i` — This context, which currently consists of three rules, is associated with initializing the grid adaptation process. In like manner to the rules in context `integrate`, the proper operation of the rules in this context requires that the specificity conflict resolution strategy be employed.

The first rule in this context, which is responsible for initializing the feature detector, is given by:

```
RULE      INITIALIZE ADAPTATION
CONTEXT   adapt_i
IF        * NOP
THEN      ADD %cycle cycle +1.0
ANDTHEN   F2INIT +2.0
ANDTHEN   SET %rpnum rpnummax
ANDTHEN   CONSIDER adapt_f
```

This premise-less rule begins by incrementing `cycle`, the number of grid adaptation cycles. It then initializes the feature detector (F2INIT) and sets the refinement parameter pointer `rpnum` to its maximum value.

The determination that the solutions on successive grids are essentially

the same and thus further adaptation is not required is the purpose of the next rule, given by:

```
RULE      FINAL CONVERGENCE REACHED
CONTEXT   adapt_i
IF        delta_cl .LE. clift_tol
ANDIF     delta_cd .LE. cdrag_tol
THEN      G2WRIT final_soln
ANDTHEN   E2PRNT
ANDTHEN   CONSIDER return
```

When this rule fires, it writes the final solution to unit `final_soln` and switches the context to `return`.

The final rule in this context is given by:

```
RULE      MAXIMUM NUMBER OF CYCLES REACHED
CONTEXT   adapt_i
IF        cycle .GE. max_cycle
THEN      CONSIDER return
```

This rule checks that the number of adaptation cycles has not exceeded `max_cycle`, and if it has, it switches the context to `return`, thereby preventing the rule which adapts the grid from firing.

adapt_f — This context is associated with detecting features and adapting the grid. The first rule:

```
RULE      FEATURE DETECTION WITH AUTOMATIC THRESHOLDS
CONTEXT   adapt_f
IF        rpnum .GT. +0.0
THEN      GET %rp rptable.0 rpnum
ANDTHEN   E2DATA rp
ANDTHEN   GET %ct cttable.0 rpnum
ANDTHEN   F2CHNG ct
ANDTHEN   F2THRS %thr_divd %thr_fuse
ANDTHEN   F2ADPT thr_divd thr_fuse %ncells
ANDTHEN   SHOW ncells
ANDTHEN   ADD %rpnum rpnum -1.0
```

uses the current refinement parameter pointer rpnum to determine which variable (rp) is to be loaded into the work array by E2DATA and which difference (ct) of it is to be taken by F2CHNG. Finally thresholds are computed (F2THRS) and cells are marked for division and fusion (F2ADPT); ncells indicates how many were marked. The rule concludes by decrementing the refinement parameter pointer such that all refinement parameters are processed sequentially.

The other rule in this context:

```
RULE      ADAPT GRID
CONTEXT   adapt_f
IF        * NOP
THEN      G2ADPT %ncella
ANDTHEN   G2GROW nbuffer +0.0 %ncellg
ANDTHEN   SHOW ncella ncellg
ANDTHEN   CONSIDER initial
```

fires after all refinement parameters have been processed. It adapts the grid (G2ADPT) and adds a buffer zone if required (G2GROW). After outputting the number of cells changed in each of the above operations, the rule switches the context to initial to begin a new cycle.

`return` — This context (which contains no rules) is used to halt the inference engine. Since there are no rules in this context, any time that the context is set to `return` the processing by the inference engine halts.

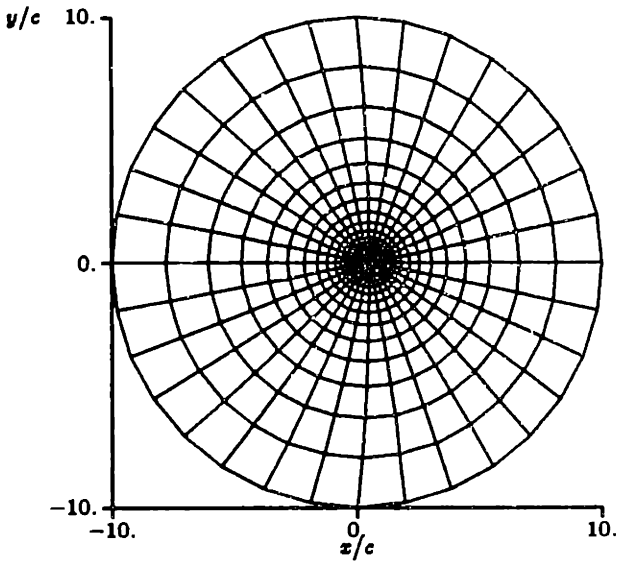
In the above discussion, reference was made to "the first rule of this context ..." only for reference. The conflict resolution strategy employed (**highest priority** and then **most premises**) is sufficient to resolve all conflicts so that the order in which the rules are defined is unimportant.

8.3 Primary Test Case — AGARD-06

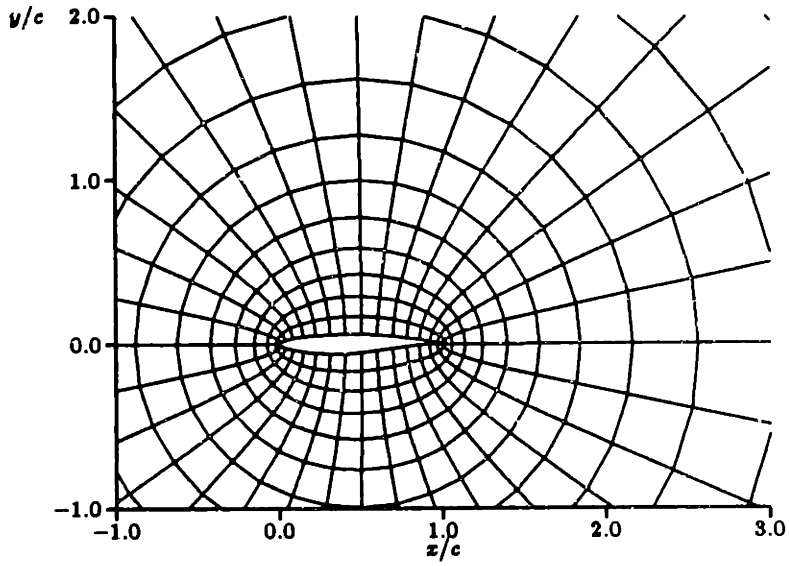
The primary test case through which the grid adaptation process has been developed and refined is the AGARD Working Group 7 test case number 6 [3], hereinafter referred to as AGARD-06. It consists of an RAE-2822 airfoil at a free-stream Mach number of $M_\infty = 0.75$ and an angle of attack $\alpha_\infty = 3.0°$. This case has a strong normal shock at about 70% chord on the upper surface.

The discussion which follows has two major themes which are intertwined: the adaptive grid evolution is examined as is the operation of the expert system's inference engine. This was done in order to provide a concrete example through which to explain the adaptation strategy and its evolution. It should be noted however that the expert system is used here only for convenience; the grid adaptation scheme could most certainly be developed without it.

The global computational grid, shown in figure 8.2, is generated using the elliptic grid generator described in section ??. It is an O-mesh, consisting of 32 cells around the airfoil (which are clustered at the leading- and trailing-edges) by 16 cells outward to the far-field computational boundary, which is placed arbitrarily at 10 chords from the airfoil (centered about the airfoil leading edge).



a: entire grid



b: near airfoil

Figure 8.2: Grid-0 for AGARD-06

Since more than one computational grid is used, each grid is named by the number of adaptation cycles used to generate it. For example, the global computational grid in figure 8.2 is not the result of a prior adaptation cycle and hence is called grid-0. Notice that the number of adaptation cycles is not necessarily the same as the number of grid levels due to possible pre-embedding an/or an adaptation which does not divide any finest-level cells (and hence does not create a new fine grid level).

8.3.1 Adaptive grid solution

In this section, the operation of the above knowledge base is traced by following the evolution of the solution for AGARD-06 through the various steps of the adaptation procedure.

Before the forward-chaining inference engine begins controlling the adaptation process, various initializations are performed. Amongst these are the definition of all solution procedure parameters (such as the smoothing coefficients and threshold selector constants) as well as the definition of refinement singularities. Also, the initial computational grid (grid 0) is generated.

When the EXPROC forward-chaining expert system begins, it starts with the context `initial` so that the rule `SET CONTROL AND INDEX VECTORS` (which is the only rule with that context) fires, resulting in the initialization of the integration scheme as well as convergence-checking attributes in the expert system. Upon completion of that rule, the context is switched to `integrate` for the next expert system cycle.

In the knowledge base, there are four rules whose context is `integrate` and hence the forward-chainer examines each; only `TAKE A TIME STEP` has no false premises and thus is selected. The firing of this rule results in the execution of one multiple-grid cycle (integration step), the application of the sliding-window convergence checking algorithm, and the incrementing of the number of iterations (`iters`). As far as the expert system is concerned, the

only outputs of this rule are the iteration count (iters) and conv_chck, which remains 0.0 as long as neither convergence or divergence is detected.

As long as conv_chck remains 0.0 and iters < max_iters, the rule TAKE A TIME STEP continues to be fired.

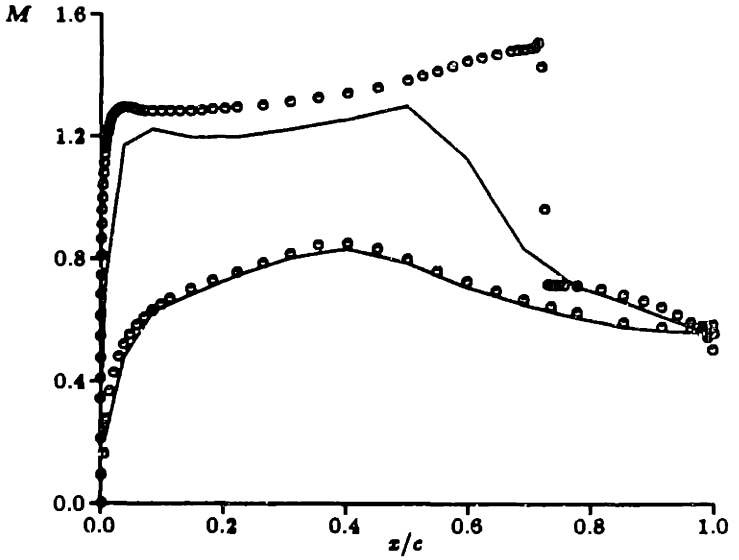
Eventually, either the convergence checker detects convergence or divergence or the number of iterations exceeds the limit max_iters. In either case, one of the other rules in this context becomes triggered. By the specificity (most premises) conflict resolution strategy the newly included rule is selected and hence fired. In AGARD-C6, conv_chck becomes +1.0 after 133 iterations (applications of TAKE A TIME STEP) and thus CONVERGENCE DETECTED is fired, resulting in the context being switched to converged.

For context converged there is only one rule, CONVERGENCE BOOKKEEPING, which is fired, resulting in the current solution being written to out_file, delta_cl and delta_cd being computed, and clift_old and cdrag_old being updated. The rule concludes by switching the context to adapt_i.

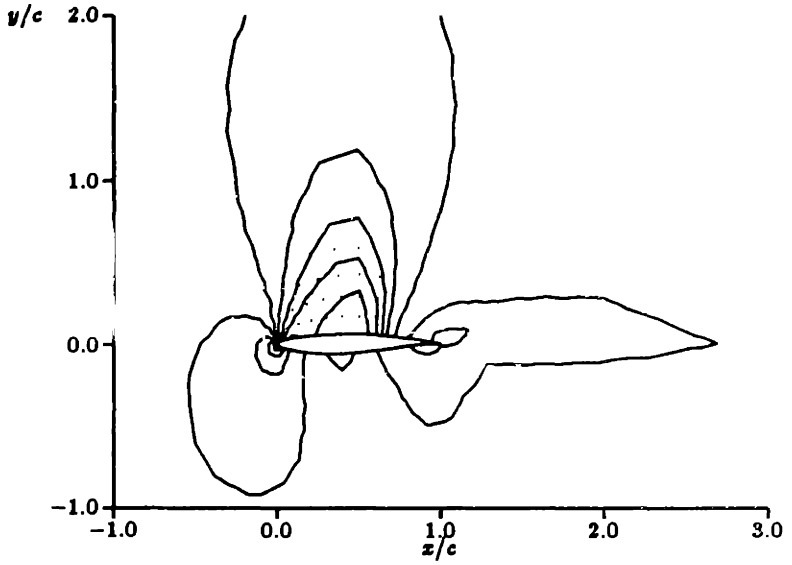
Ex post facto plots of the grid 0 solution (generated from the solution file written by the above rule) are given in figure 8.3. In part a, the surface Mach number distribution for the grid 0 (line) and final (grid 4) (symbols) are compared. The upper surface solutions disagree considerably, especially in the level of the Mach number ahead of the shock as well as in the shock location, strength, and width; the grid 0 solution smears the shock over about 25% of the chord. In contrast, the lower surface Mach numbers are in reasonable agreement. Part b of the figure shows Mach number contours, with the nodes in the supersonic region being indicated by dots.

There are three rules with the context adapt_i, only one of which does not have a false premise at this point. That rule, INITIALIZE ADAPTATION, increments the adaptation cycle count cycle, initializes the feature detector, and sets the context to adapt_f.

Rule FEATURE DETECTION WITH AUTOMATIC THRESHOLDS is the only rule



a: surface distribution.
Line is grid 0 solution, symbols are grid 4 solution.



b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.3: Mach number for grid 0, test case AGARD-06. Computed $C_L = 0.9026$, $C_D = 0.0364$

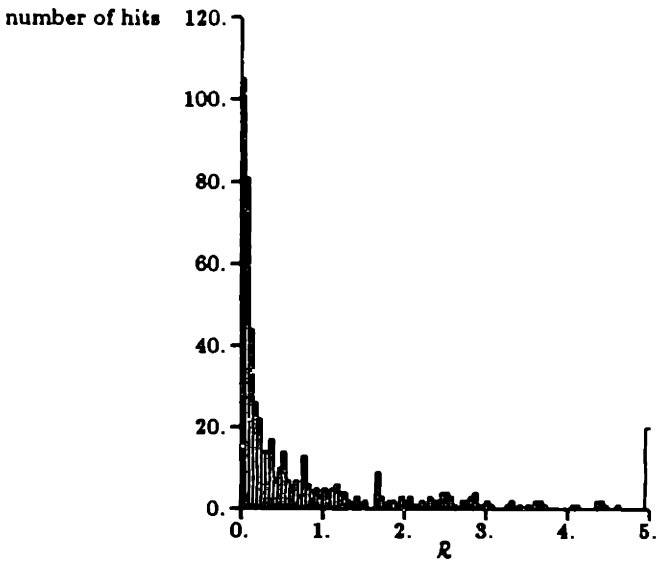
in context adapt_f without false premises, and is therefore fired. It uses the first (and only) refinement parameter definition ($\mathcal{R} = \tilde{\nabla}\rho$) to determine which cells should be divide and/or fused. It ends by decrementing the refinement parameter pointer rpnum, in this case making its value 0.0.

Plots resulting from the application of the feature detector to the grid 0 solution are shown in figure 8.4. Part a shows the histogram of the frequency of occurrence of the various refinement parameters levels, which is largest near $\mathcal{R} = 0.0$ and decreases as $\mathcal{R}$ increases. The smoothed cumulative distribution function of this histogram is shown in part b, along with the automatically detected "knee" in the curve and *a priori* imposed limits shown as stippled areas; for this case, $\mathcal{R}_d$ is limited to be above 1.25 and the number of nodes for which $\mathcal{R} > \mathcal{R}_d$ must not exceed 25% of the total number of nodes. As can be seen from the figure for this case, the division threshold is set by the former limit so that $\mathcal{R}_d = 1.25$. As usual, the fusion threshold is set to $\mathcal{R}_f = 0.50$.

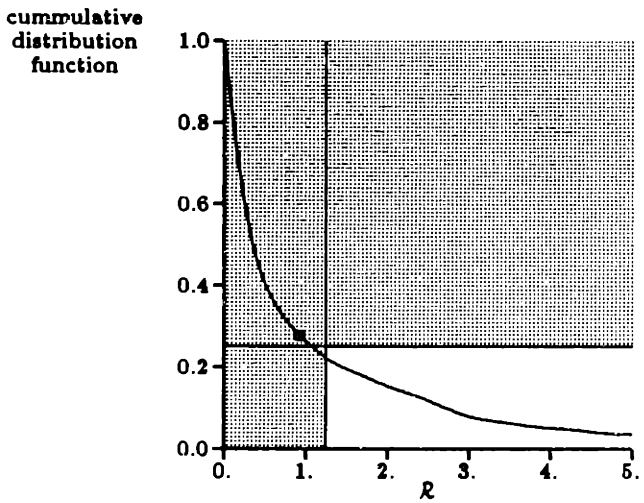
Figure 8.5 shows the relative magnitude of $\mathcal{R}$ at each node for the thresholded values determined above. Notice that the refinement parameter for all the nodes on the airfoil surface exceeds the division threshold as does the refinement parameter for nodes in the supersonic zone and in the vicinity of the leading and trailing edges. Nodes for which the refinement parameter is less than the fusion threshold (which are marked by o) surround this region.

The next cycle of the expert system now triggers both rules in context adapt_f since the refinement parameter counter has reach 0.0. By the specificity conflict resolution strategy, rule ADAPT GRID is fired, resulting in the new embedded grid, grid 1, and the switch back to context initial.

The newly adapted grid is shown in figure 8.6. Notice that the grid is basically adapted around all nodes for which $\mathcal{R} \geq \mathcal{R}_d$, that is near the airfoil surface, around the leading and trailing edges, and in the supersonic zone. However, an exception occurs on the lower surface near the trailing edge.



a: histogram (of R)



b: cumulative distribution function (with guards)
"Knee" is shown as a square symbol

Figure 8.4: Feature detector applied to grid 0 solution, test case AGARD-06

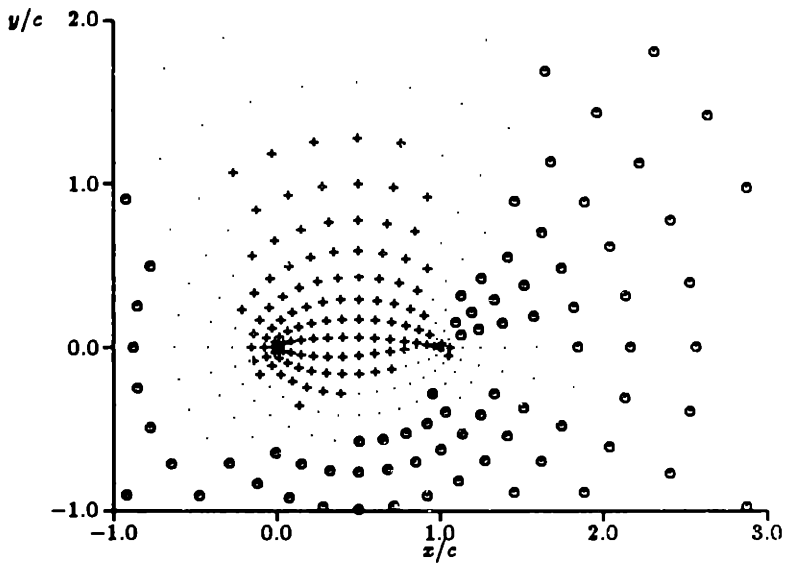


Figure 8.5: Relationship of $\mathcal{R}$ at each node versus the division and fusion thresholds. + indicates nodes where $\mathcal{R} \geq \mathcal{R}_d$ and ∘ indicates nodes where $\mathcal{R} \leq \mathcal{R}_f$

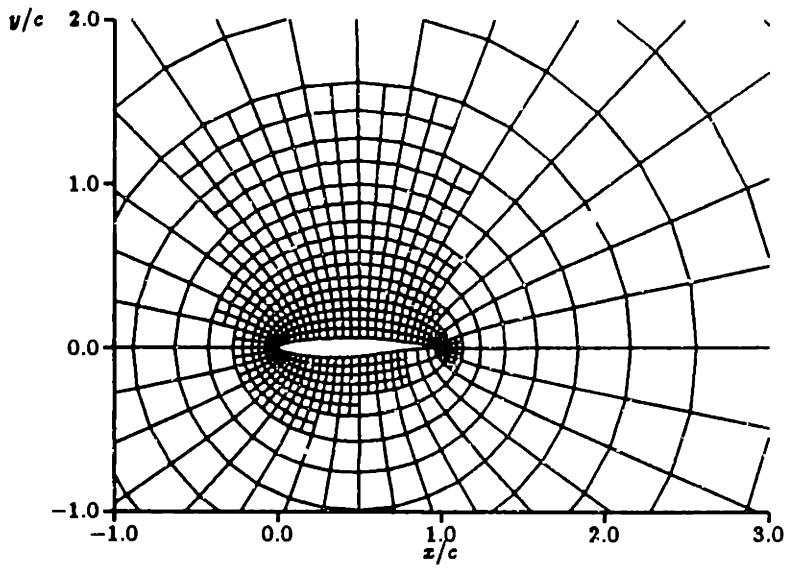


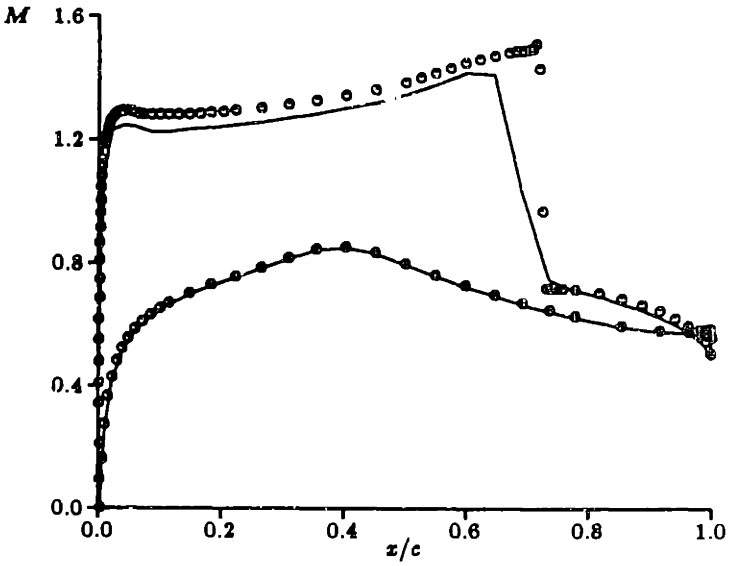
Figure 8.6: Grid 1 near airfoil

That undivided region is a result of the island detector and elimination algorithm which noticed and adapted a region one cell wide which it then eliminated. Another small difference from the grid expected (based upon figure 8.5) is a small region a few cells below the leading edge; the actual grid (figure 8.6) shows that this region has been filled in by the void detection and elimination algorithm.

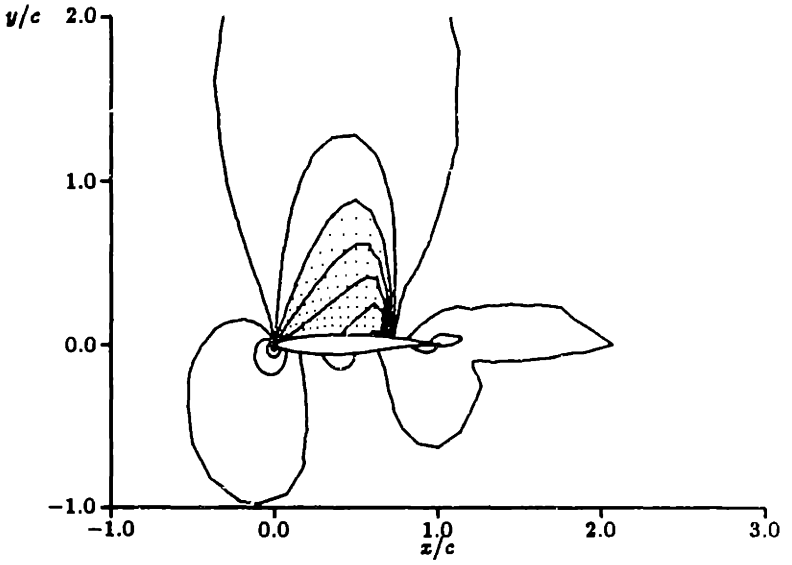
With the context now `initial`, the process as described above repeats. For brevity, the discussion of the rules which are triggered and fired is now eliminated from the following discussion.

Using grid 1, the flow integrator requires 102 iterations to converge to the solution shown in figure 8.7. Notice that the surface Mach number distribution is now closer to the final (grid 4) solution, both in the level ahead of the shock and in the shape and location of the shock itself.

The feature detector applied to this case is shown in figure 8.8. Notice

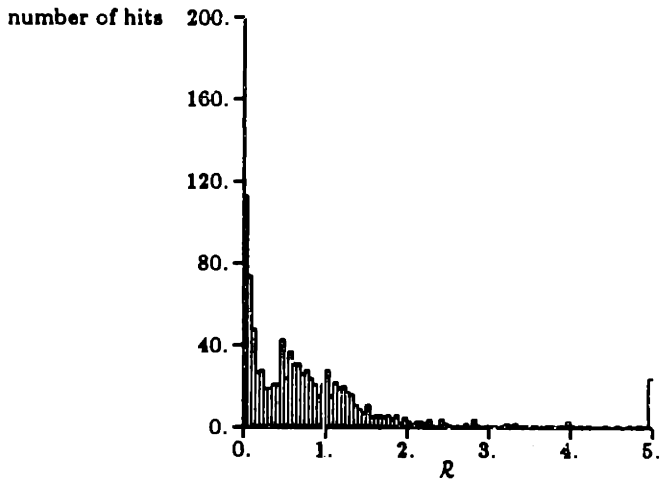


a: surface distribution.
Line is grid 1 solution, symbols are grid 4 solution.

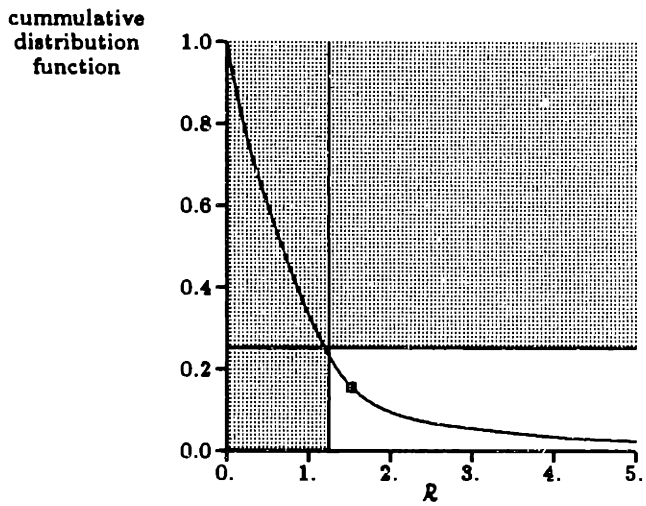


b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.7: Mach number for grid 1, test case AGARD-06. Computed $C_L = 0.9977$, $C_D = 0.0357$



a: histogram (of R)



b: cumulative distribution function (with guards)
"Knee" is shown as a square symbol

Figure 8.8: Feature detector applied to grid 1 solution, test case AGARD-06

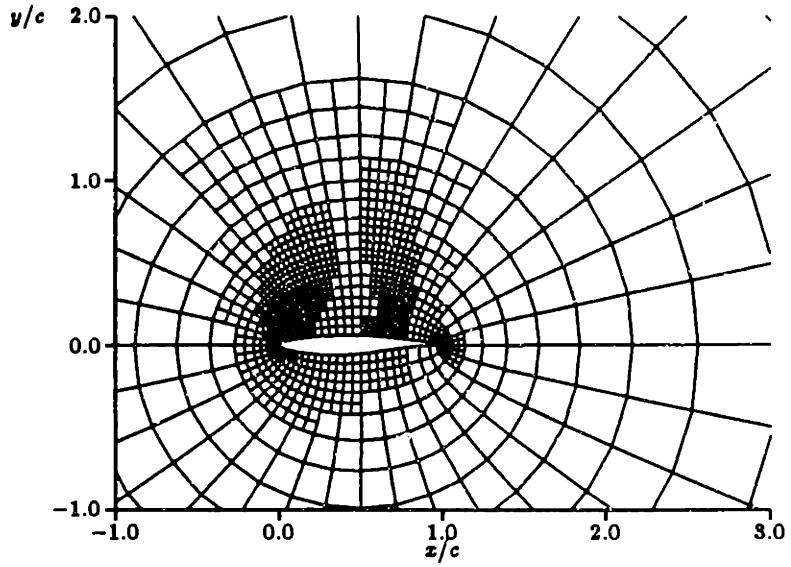
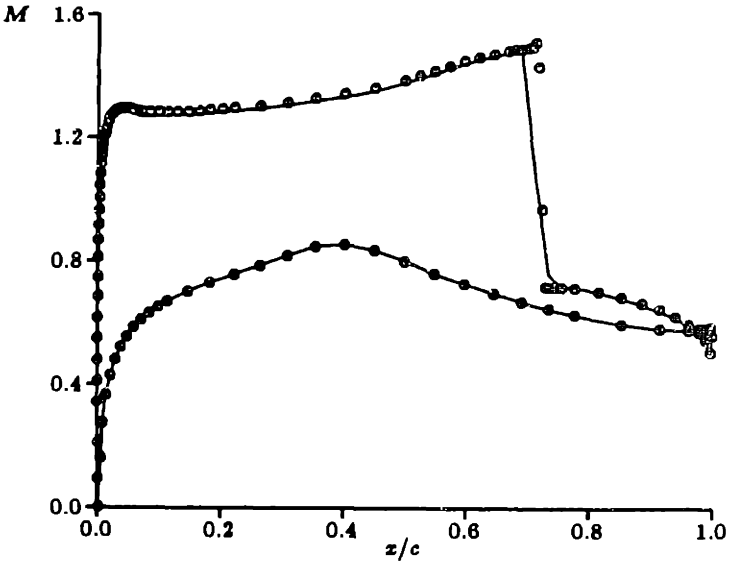


Figure 8.9: Grid 2 near airfoil

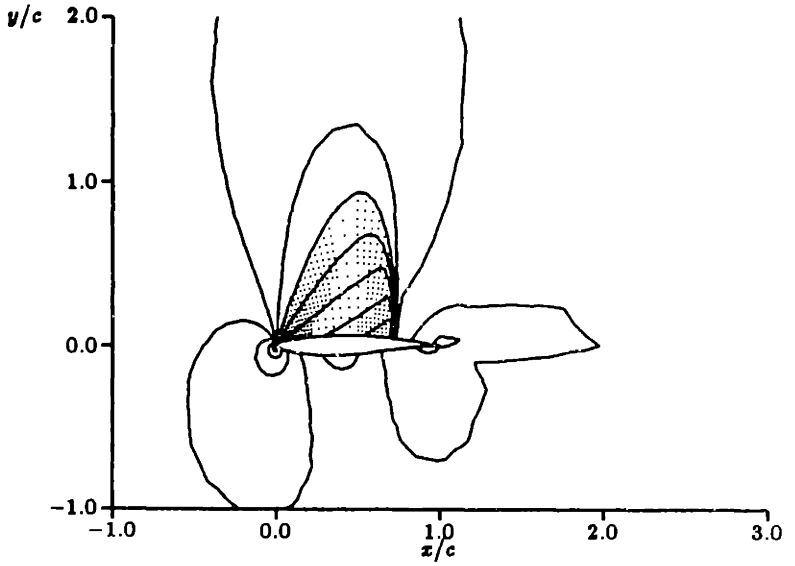
from the histogram plot (part a), that there are still about 120 nodes with a nearly zero refinement parameter (as in figure 8.4), but that there are now a substantial number of nodes with a refinement parameter close to the average value of 1.0. This results in the "knee" of the cumulative distribution function being moved to the right and out of the restricted region. Hence for this grid, the value of the division threshold is set by the "knee" rather than some limit.

The resulting adapted grid (grid 2) is shown in figure 8.9. Notice here how the grid is now concentrated near the upper surface shock and near the stagnation and leading edge expansion regions. Here the shape of the level 1 grid (cells which are one level finer than the global grid) is unaltered by the second adaptation.

The Mach number distribution for grid 2, shown in figure 8.10, approaches still more closely to the level 4 grid solution as expected, the major



a: surface distribution.
Line is grid 2 solution, symbols are grid 4 solution.



b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.10: Mach number for grid 2, test case AGARD-06. Computed $C_L = 1.0449$, $C_D = 0.0385$

discrepancy being the crispness of the shock. From part b of the figure, one can see the shape of the embedded region in the supersonic zone (by the dots which are placed at each node location). From this one can clearly see that the contours pass very smoothly through the edges of the various embedded regions, as desired.

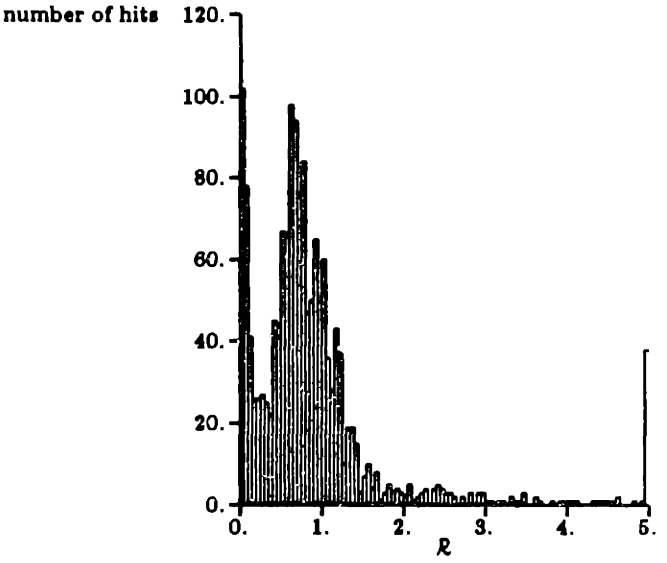
The feature detector plots which result from the grid 2 solution are shown in figure 8.11. Again due to the shift of the histogram toward 1.0, the automatically determined value of the threshold is used rather than a value imposed by the guards.

The third adapted grid (grid 3), its solution, and the resulting feature detector are shown in figures 8.12, 8.13, and 8.14 respectively. Again the grid is concentrated near the shock and leading edge region, resulting in a sharper shock.

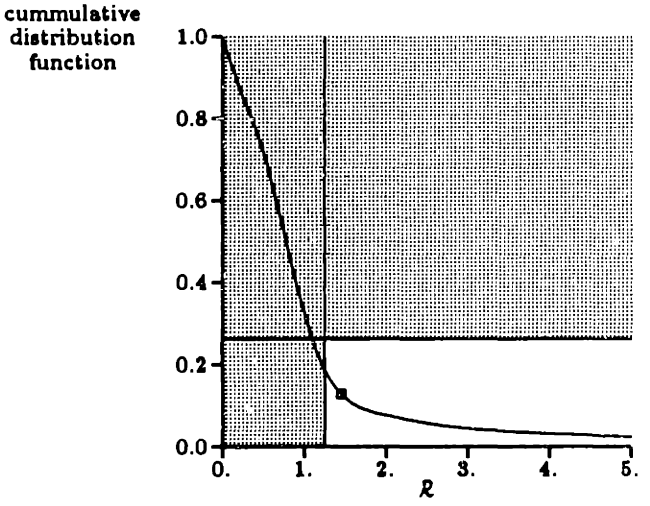
The fourth (and it turns out final) adapted grid is shown in figure 8.15. Notice that the embedded regions are nested; for example in the vicinity of the shock, they become smaller in extent as the grid size decreases. This results in efficient use of the computational resources (storage and computer time). Another interesting point from figure 8.15 is that the shock is not aligned with the grid, as indicated by the irregular shapes of the embedded regions.

The solution on this final grid is plotted in figures 8.16 and 8.17. In part a of these figures, the line and symbols now correspond to the same computed solution (the symbols indicate the grid spacing). The contour plots in part b of the figures clearly show the crispness of the shock.

One of the measures of solution accuracy is the degree to which the Rankine-Hugoniot shock jump is predicted. From figure 8.16a, the peak Mach number is 1.500, which from shock jump relations predict a Mach number immediately after the shock of 0.701 and a total pressure loss of $\eta = 0.070$. These compare reasonably well with the numerically predicted



a: histogram (of R)



b: cumulative distribution function (with guards)
"Knee" is shown as a square symbol

Figure 8.11: Feature detector applied to grid 2 solution, test case AGARD-06

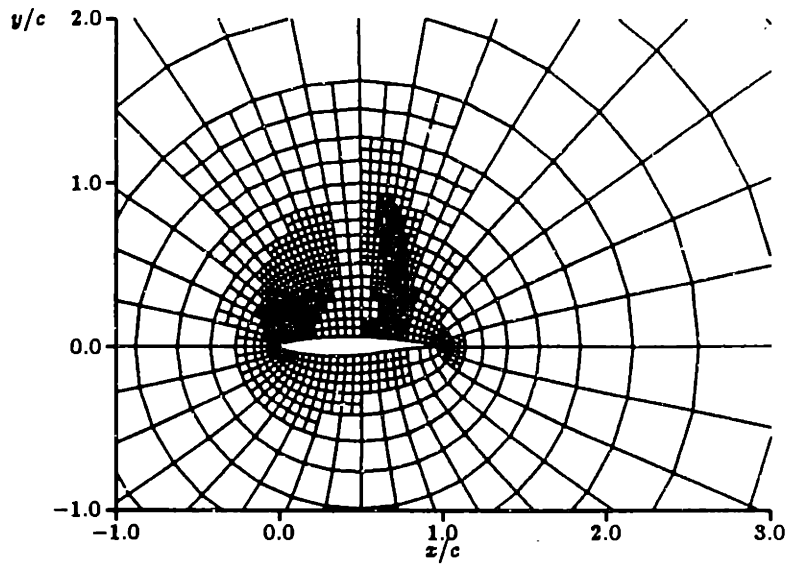
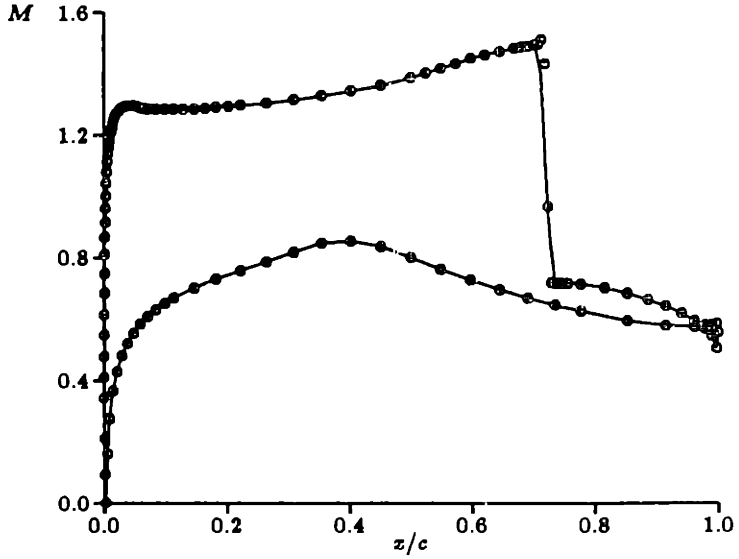


Figure 8.12: Grid 3 near airfoil

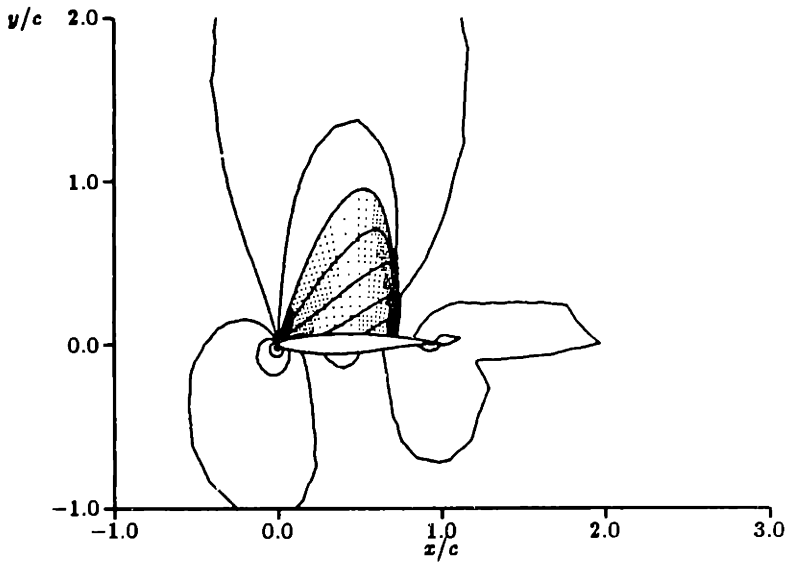
values of 0.719 and 0.078 respectively.

After this computed solution is obtained, the expert system context becomes **adapt_1**. Only rule FINAL CONVERGENCE REACHED will have true premise clauses and hence is selected over INITIALIZE ADAPTATION (because it has more premises). This results in the former being fired, which writes out a final solution and switches the context to **return**. Since there are no rules within context **return**, the forward-chainer halts at this point.

The convergence history for this adapted solution can be analyzed in a number of ways. First, one can examine various norms of the maximum change in the x-momentum as a function of iteration; two such plots are shown in figure 8.18. In both cases, the change decays until iteration 133 at which point the grid is adapted, resulting in an increase in the norms by more than two orders of magnitude. The norm then decreases until the second adaptation, etc. (Note that the convergence histories show convergence

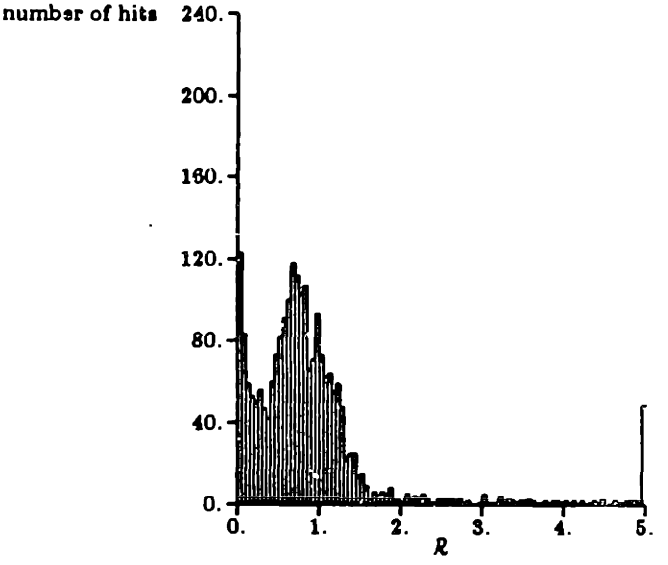


a: surface distribution.
Line is grid 3 solution, symbols are grid 4 solution.

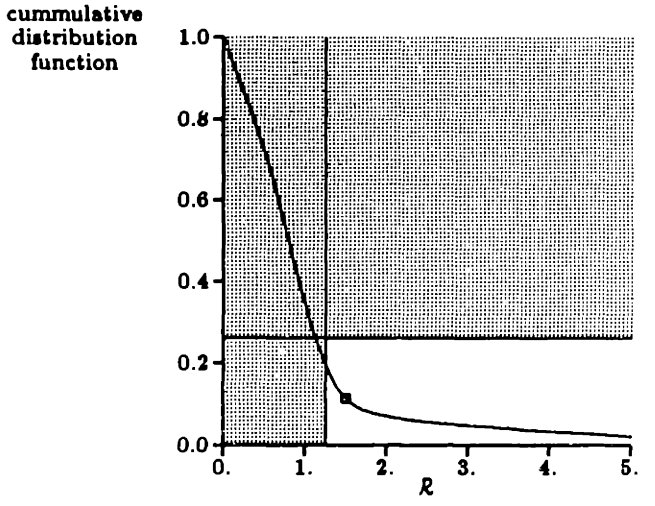


b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.13: Mach number for grid 3, test case AGARD-06. Computed $C_L = 1.0574$, $C_D = 0.0393$



a: histogram (of R)



b: cumulative distribution function (with guards)
"Knee" is shown as a square symbol

Figure 8.14: Feature detector applied to grid 3 solution, test case AGARD-06

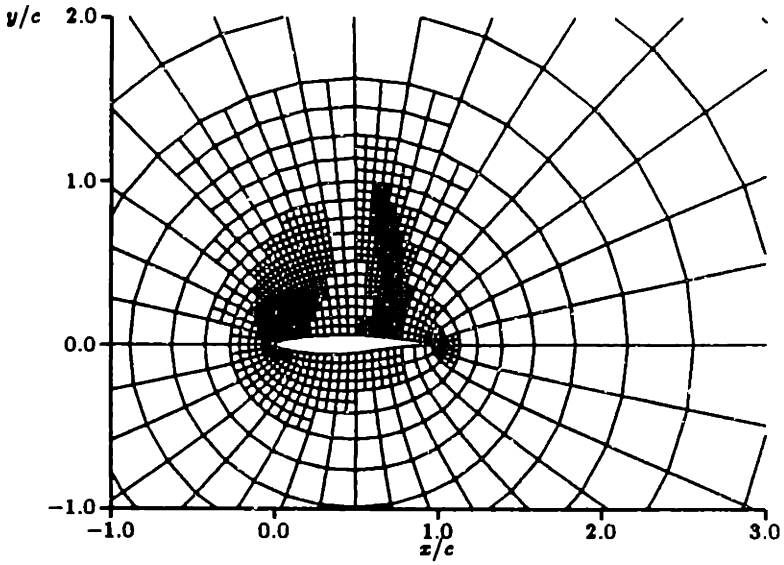
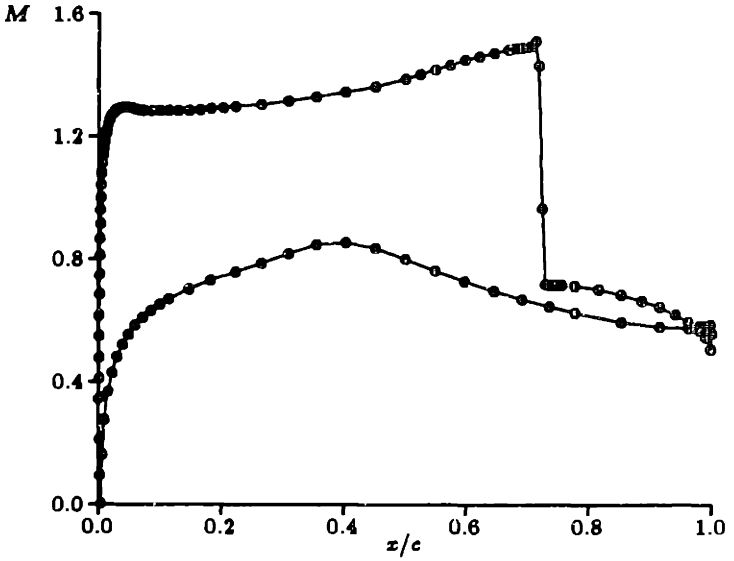


Figure 8.15: Grid 4 (final grid) near airfoil

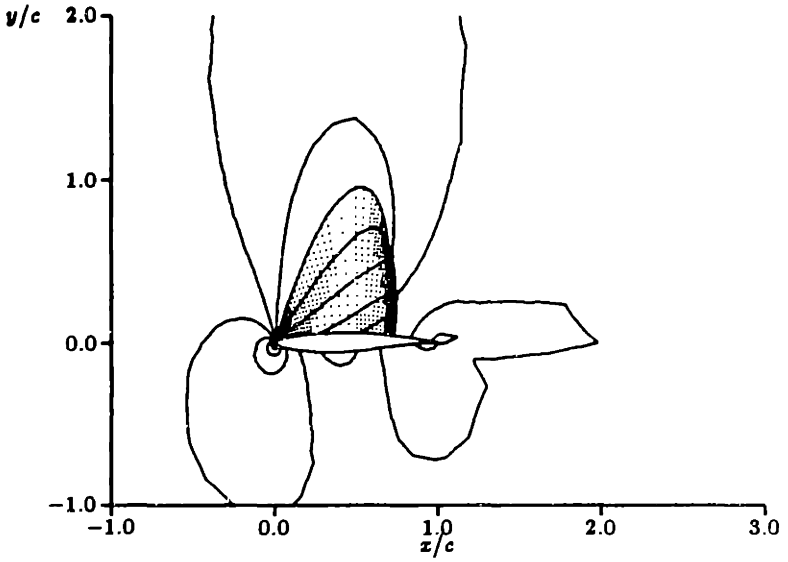
on six grids; this is due to the fact that the plots were generated from a MITOSIS run which had a tighter global convergence limit `clift_tol` = 0.002 and **cdrag_tol** = 0.001 to ensure grid independence).

It is interesting to note that the level of the norm at "convergence" is different for the various grid levels due to the sliding-window convergence checking algorithm (section 3.6). Another interesting point to notice from figure 8.18 is that the rate of convergence (as evidenced by the slopes) is essentially independent of grid level. This implies that the embedded regions do not affect the convergence rates, as expected from chapter 4. Unfortunately, the actual computation time required on successive grid levels is strongly dependent on the number of computational nodes, and hence the required CPU time increases on successive levels; this is shown in figure 8.19.

An alternative method of analyzing convergence follows from a plot of the force coefficient versus iteration count as in figure 8.20. Here the force

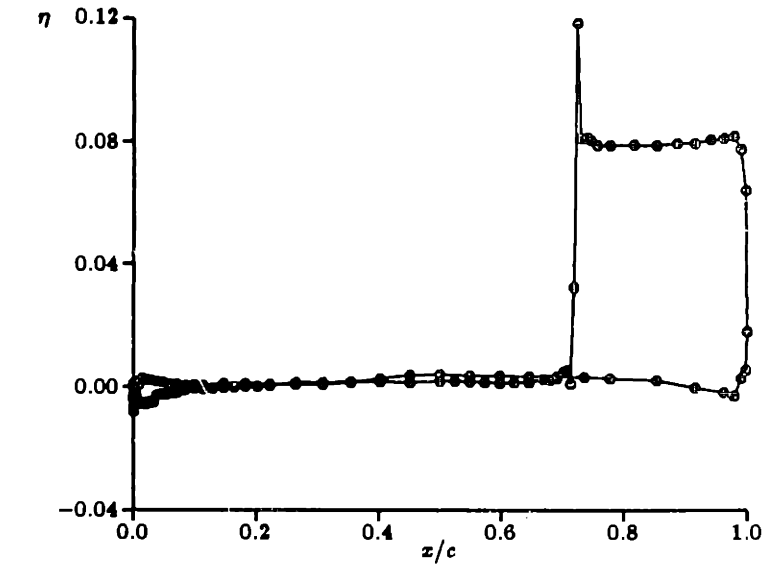


a: surface distribution.
Line and symbols are both computed solutions.

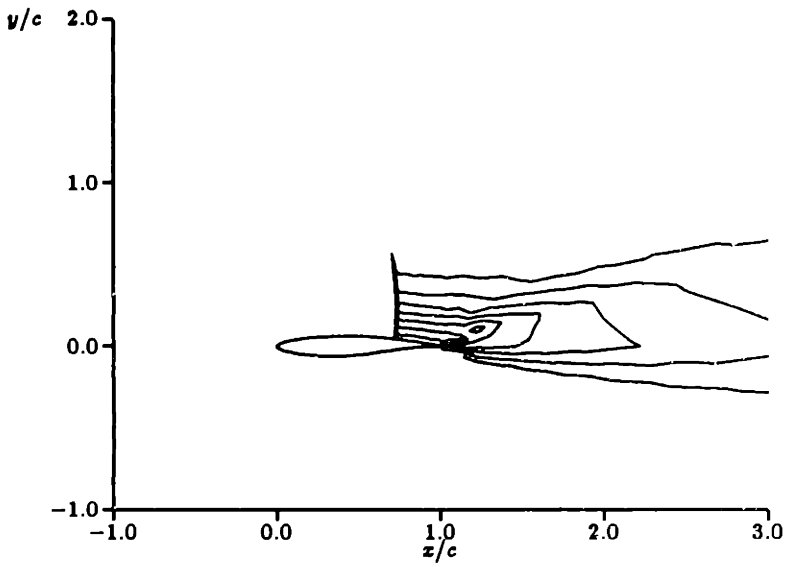


b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.16: Mach number for final grid (grid 4), test case AGARD-06.
Computed $C_L = 1.0602$, $C_D = 0.0394$

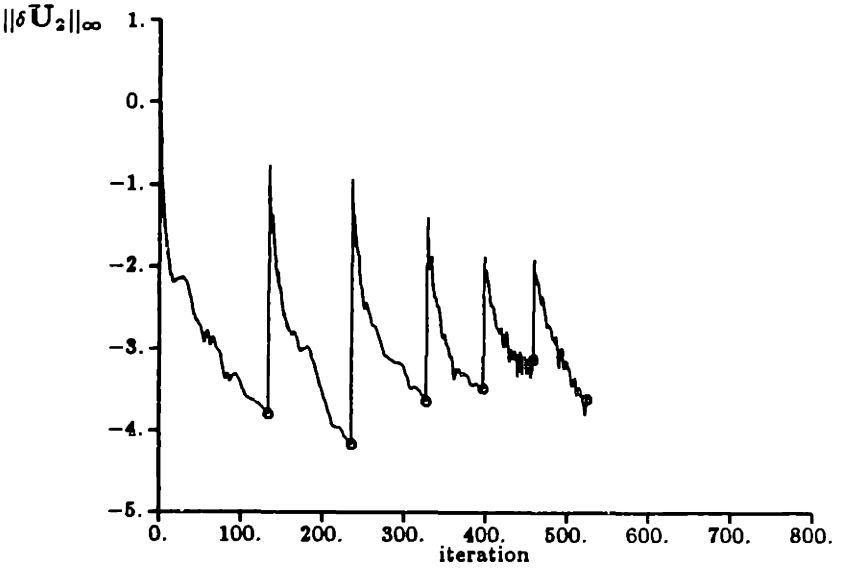


a: surface distribution.
Line and symbols are both computed solutions.

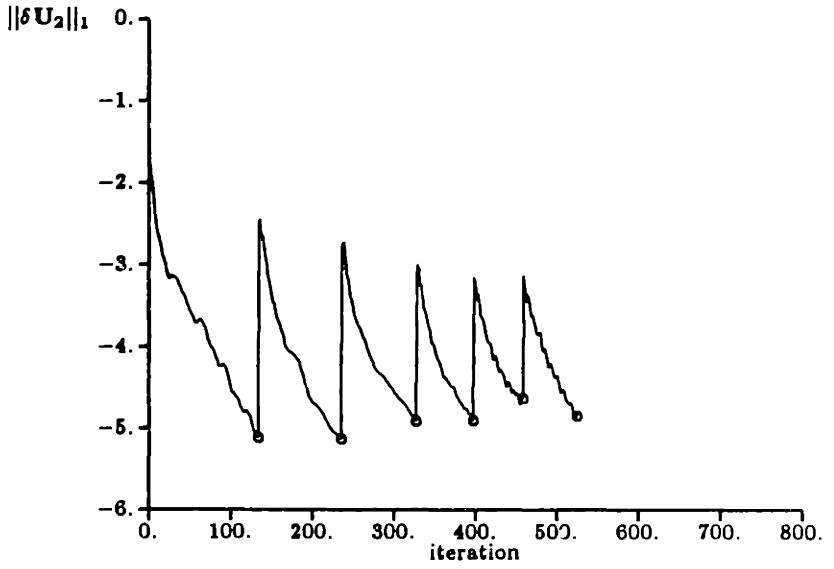


b: contours, $\Delta\eta = 0.01$

Figure 8.17: Total pressure loss for final grid (grid 4), test case AGARD-06



a: L_∞-norm



b: L_1-norm

Figure 8.18: Convergence histories for AGARD-06

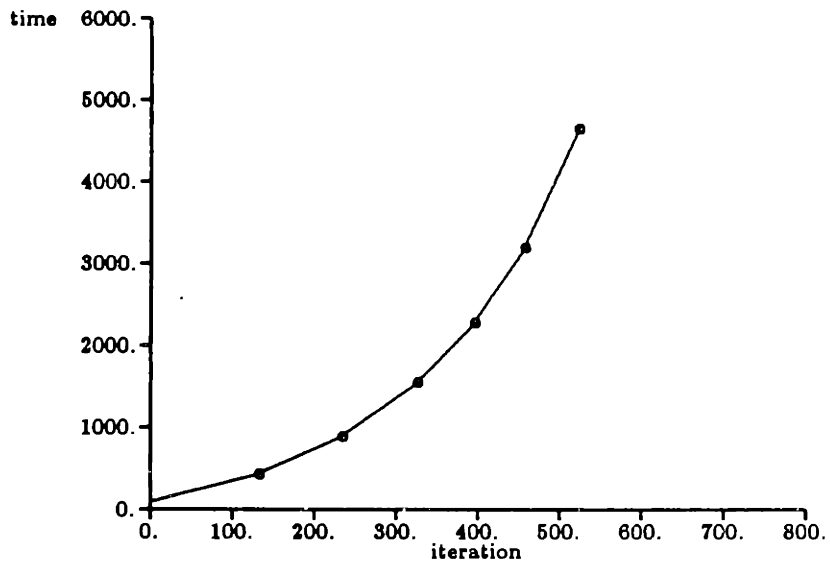


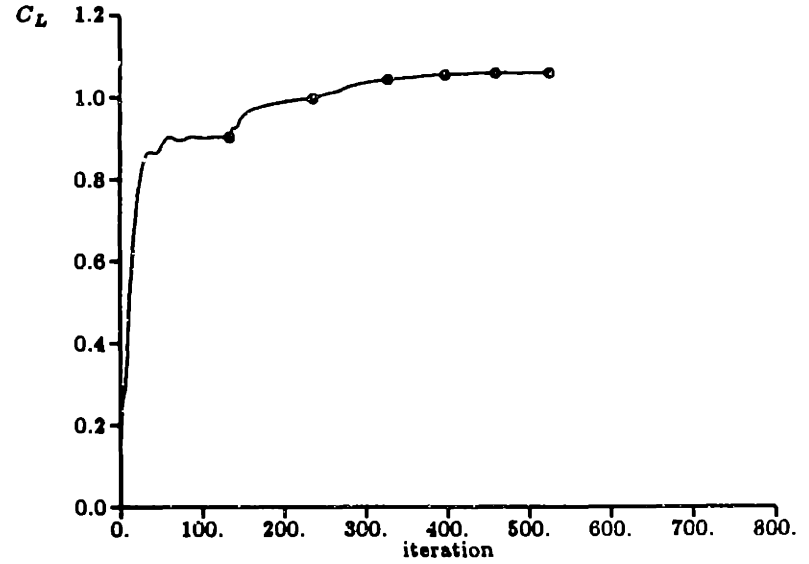
Figure 8.19: Required CPU time for AGARD-06

coefficients clearly converge to different values (as a function of adaptation level). The figure also clearly shows that the difference between computed force coefficients diminishes as the grid is continually refined.

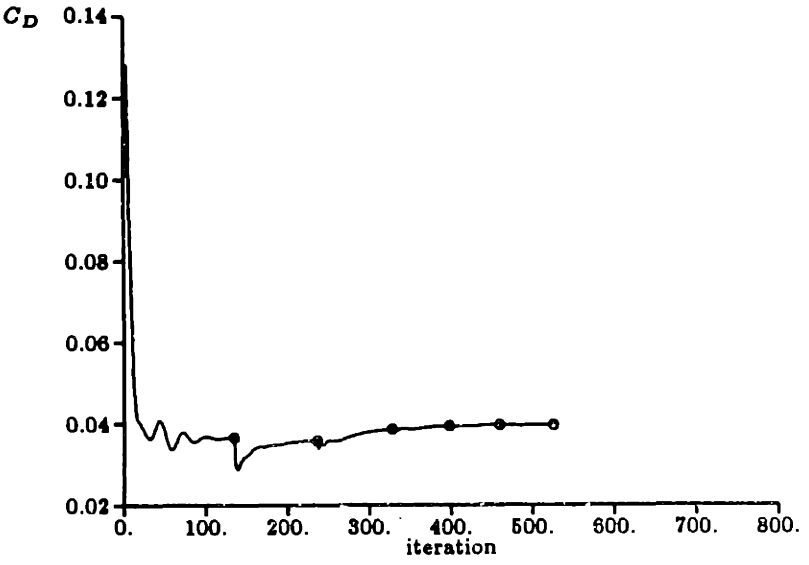
Table 8.1 summarizes the grid adaptation process for AGARD-06. Each line represents the calculation on a successive adaptation cycle. In order to compare solutions on different computers, a pseudo-CPU time is used. It is computed by ptime = number of nodes × number of iterations/200, where the 200 is close to the constant which yields seconds on a VAX-750.

8.3.2 Global grid solution

When one develops an adaptive solution procedure, care must be taken to ensure that the solution is independent of the adaptive strategy. A globally refined solution therefore is presented in this section. Comparisons of the accuracy and computational efficiency of the adaptively and globally



a: lift coefficient



b: drag coefficient

Figure 8.20: Convergence of force coefficients for AGARD-06

cycle number	number of nodes	number of cells	C_L	C_D	number of iterations	pseudo-CPU time
0.	544.	672.	.9026	.0364	133.	362.
1.	978.	1200.	.9977	.0357	235.	861.
2.	1565.	1912.	1.0449	.0385	327.	1580.
3.	2250.	2744.	1.0574	.0393	397.	2368.
4.	3225.	3924.	1.0602	.0394	458.	3352.

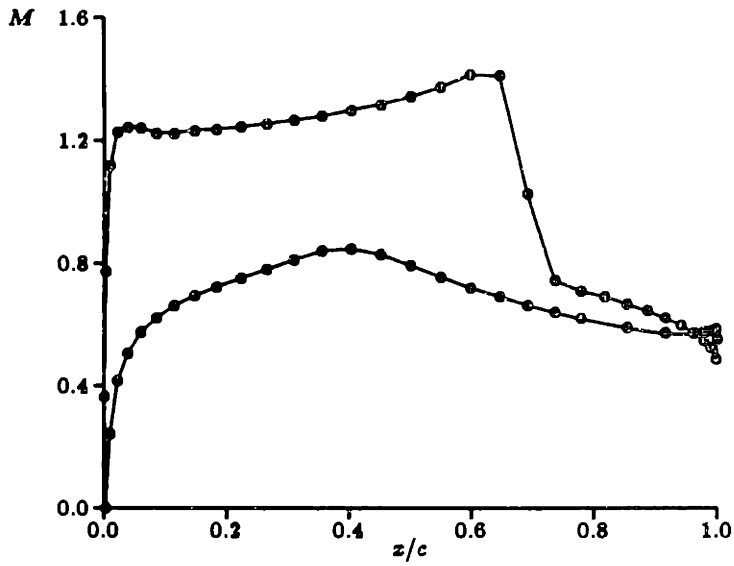
Table 8.1: Summary of solution evolution for AGARD-06

refined solutions can then provide realistic evidence of the effectiveness of the adaptive solution procedure.

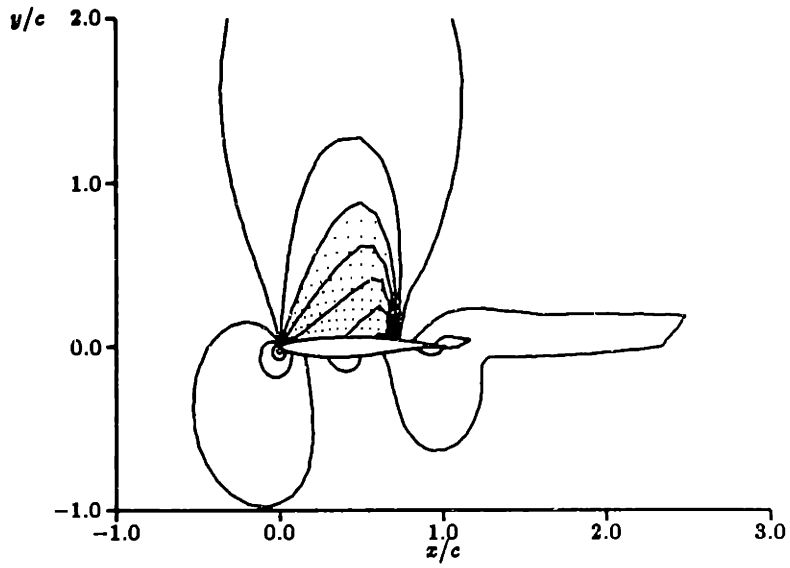
The globally refined solution is computed by beginning with the same grid 0 as in the adaptively refined case (shown in figure 8.2). The integration scheme is applied on that grid, yielding the exact same solution as the first solution in the adaptive grid case. But then, instead of detecting features and embedding only at those features (as in the adaptive case), the globally refined solution procedure divides all cells, thereby forming a grid which is twice as fine in each computational direction. After integrating to steady state on the new grid, a new grid is generated and a solution computed; this process was continued until the computer's memory was exhausted.

The globally refined solution evolution is summarized in Table 8.2. Here the final computed solution is for grid 3 (due to computer memory limitations); the grid 4 line has been extrapolated so that the large computer resource requirements can be compared with the adaptively refined solution's requirements.

Figures 8.21, 8.22, and 8.23 show the Mach number distributions obtained on the successive grids; included in the figures are the solutions from the adaptively refined grid solution obtained in the previous section. Part a

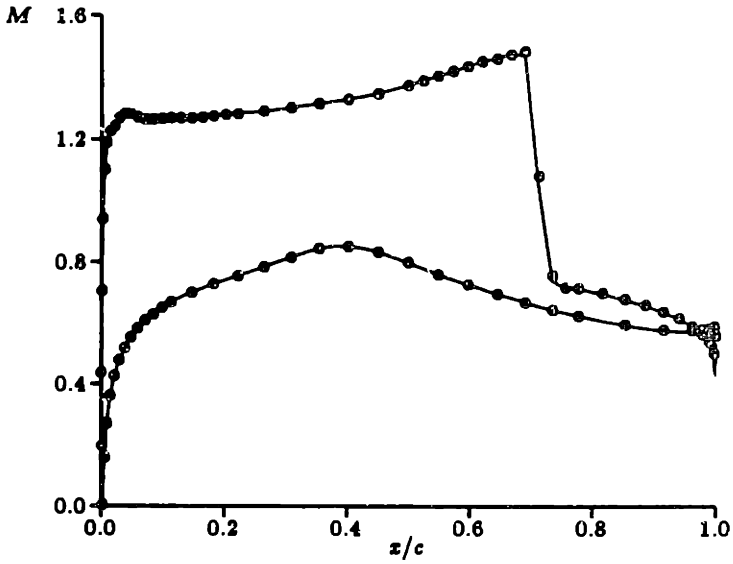


a: surface distribution.
Line is globally refined, symbols are adaptively refined.

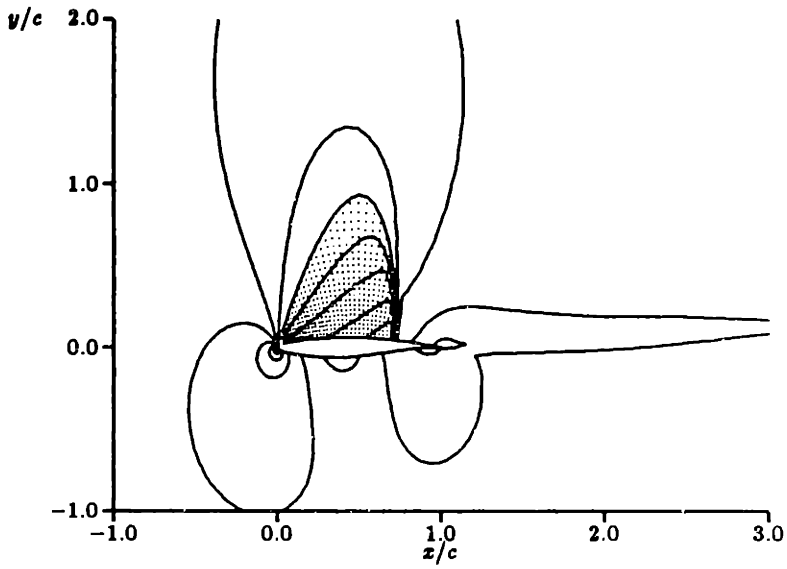


b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.21: Mach number for grid 1, test case AGARD-06

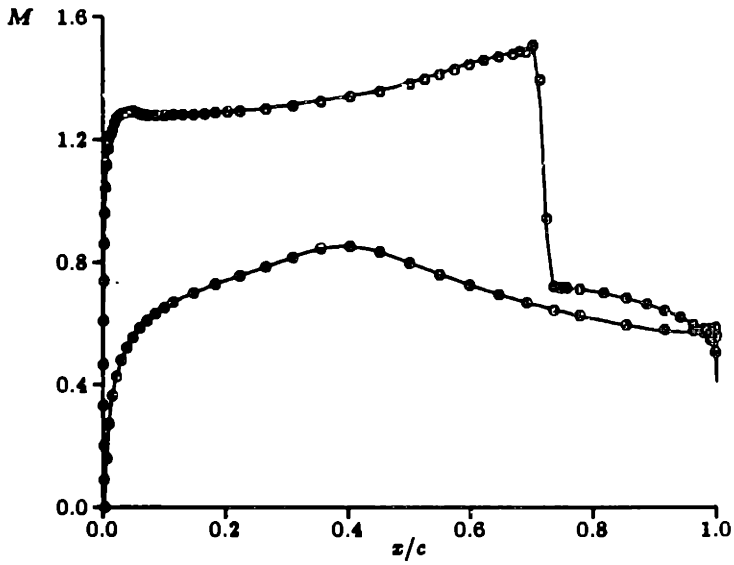


a: surface distribution.
Line is globally refined, symbols are adaptively refined.

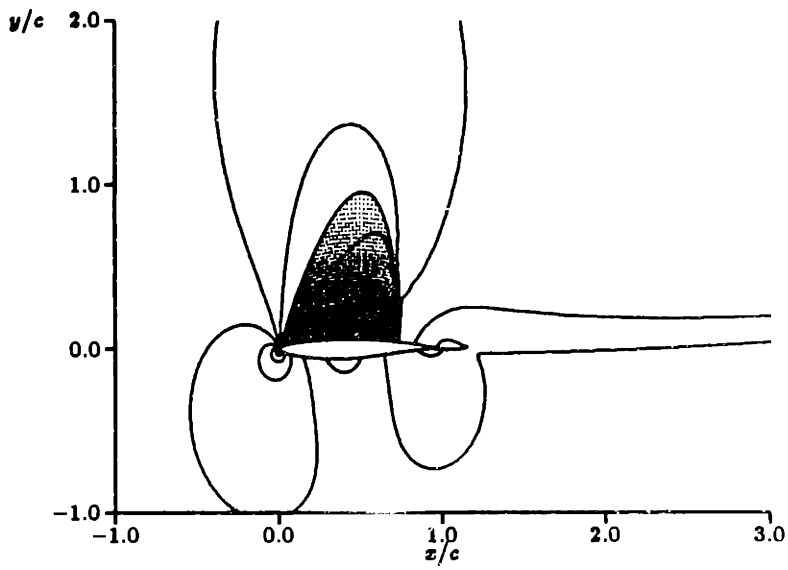


b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.22: Mach number for grid 2, test case AGARD-06



a: surface distribution.
Line is globally refined, symbols are adaptively refined.



b: contours, $\Delta M = 0.10$
Nodes in supersonic regions are dotted.

Figure 8.23: Mach number for grid 3, test case AGARD-06

cycle number	number of nodes	number of cells	C_L	C_D	number of iterations	pseudo-CPU time
0.	544.	672.	.9026	.0364	133.	362.
1.	2112.	2720.	1.0002	.0357	226.	1344.
2.	8320.	10912.	1.0460	.0381	316.	5088.
3.	33024.	43680.	1.0633	.0392	389.	17142.
4.	131584.	174752.	??	??	e460.	e63854.

Table 8.2: Summary of successive global solutions for AGARD-06 (e indicates an estimated value)

of the figures show almost perfect agreement between the adaptively and globally refined solutions on the airfoil surface. On the other hand, the contour plots in part b of the figures show some discrepancy, especially in the wake region. Hence, although the adaptively refined and globally refined solutions do differ away from the airfoil, their solutions agree well on the airfoil surface.

The agreement of the solutions on the airfoil surface are summarized in figure 8.24, where the lift coefficient at convergence on each of the computational grids is plotted versus the level of refinement. Since the two solutions share a common grid 0, the size of the finest cells in the two computations are the same at a given level of refinement. The small discrepancy at level 3 is due to small differences at the shock.

The main advantage of the adaptive solution algorithm employed here can be seen in figure 8.25, where the lift coefficient at convergence for the two cases is plotted versus the pseudo-CPU time required. Here it is clear that the adaptively refined solution is obtained much faster than comparable globally refined solutions.

The true saving is plotted in figure 8.26, where the ordinate represents

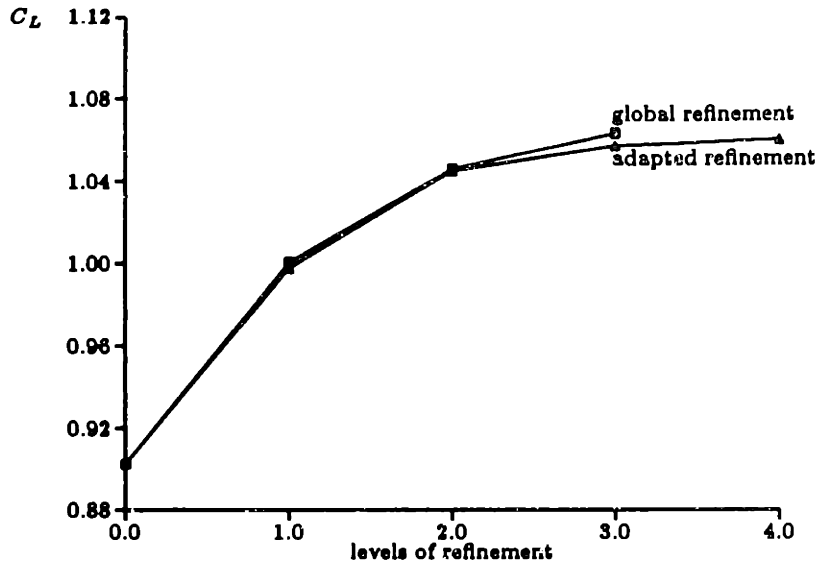


Figure 8.24: Globally and adaptively refined solutions for AGARD-06

the ratio of the resources required for a globally refined solution to the resources required for the adaptively refined solution; the abscissa represents the difference between the solution at a given level and the final solution, normalized by the final solution. This figure shows that adaptive solutions can be obtained with 2 to 20 times less computer time than is required for globally refined solutions, depending on the level of accuracy required. Actually, this curve can be extended if more accuracy is required, yielding a savings of up to a factor of 100 or more. The storage savings is even more dramatic.

Hence, for a given resource allocation, the adaptive solution procedure can yield significantly better solutions; alternatively, for a given level of accuracy, the adaptive solution can achieve it while expending considerably fewer resources.

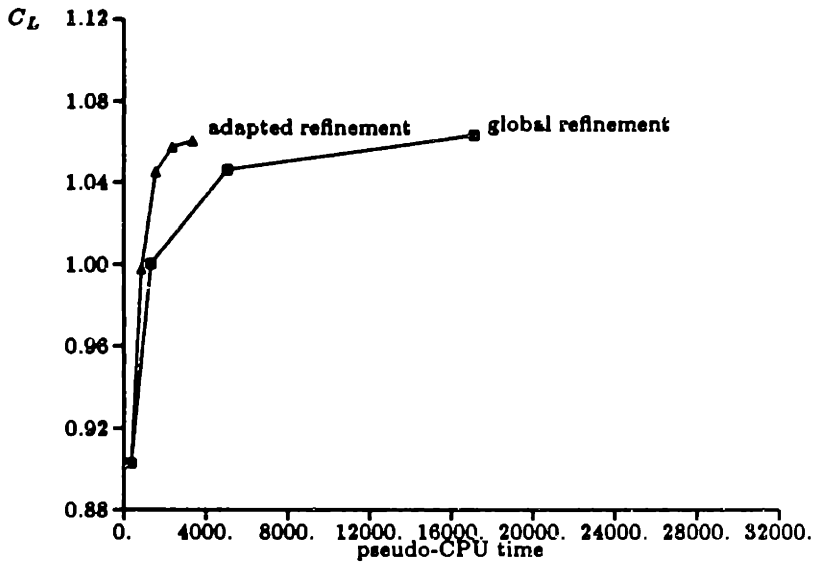


Figure 8.25: Time required for globally and adaptively refined solutions for AGARD-06

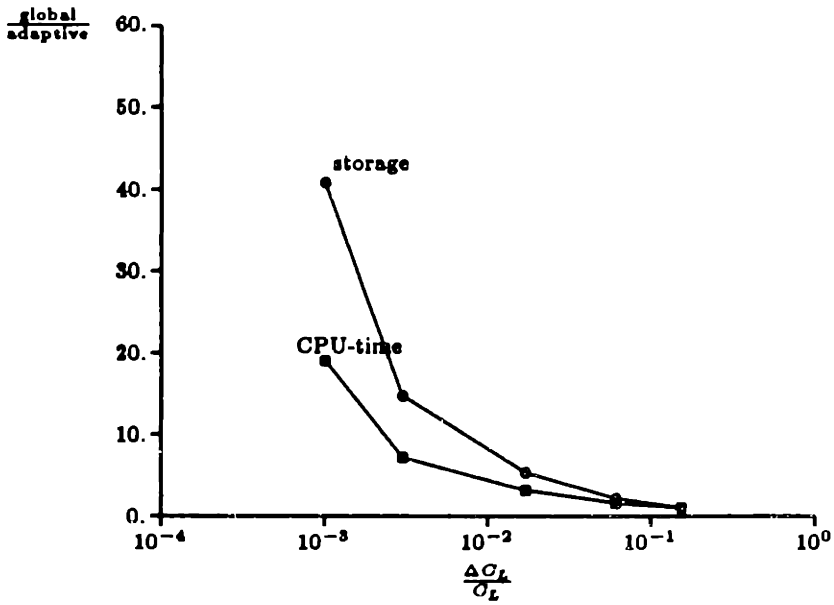


Figure 8.26: Savings factor as a function of required accuracy for AGARD-06

8.4 Comparison with Published Solution

The previous sections have demonstrated that the globally and adaptively refined solutions agreed well with one another. Unfortunately, their final computed lift and drag coefficients differ by more than 4% from published values[2]. In this section, the possible sources of this discrepancy are systematically eliminated.

8.4.1 Published solutions

In 1983, AGARD published a set of "test cases for steady inviscid transonic or supersonic flows" [3] with the express purpose of providing a suite of test cases against which authors of Euler-equation methods could validate their results. The test cases chosen for two-dimensional flows were largely those for which no known exact solutions existed; the test case discussed in

	average	standard deviation	minimum	maximum
C_L	1.1058	0.0134	1.0971	1.1292
C_D	0.0467	0.0023	0.0448	0.0504

Table 8.3: Group 2 solutions from AGARD report for AGARD-06

the previous section is one such case.

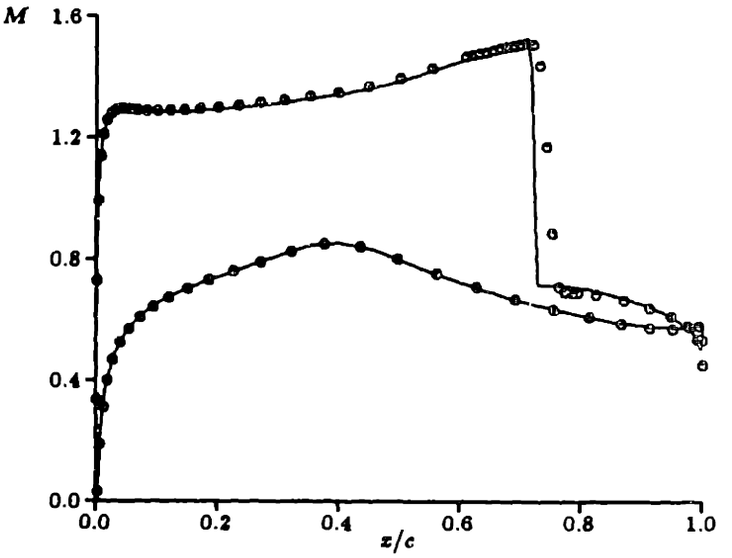
Thirteen sets of solutions were submitted and reported in [2]. In order to determine which solution(s) was(were) "correct", the AGARD Working Group 7 undertook a lengthly evaluation of the received results.

During the first stage of the evaluation process, the computed aerodynamic coefficients from all of the submitted solutions were compared in an attempt to eliminate those solutions which clearly were not in agreement with the others for some discernible reason (such as proximity of the far-field boundary or unreasonable mesh coarseness). The remaining solutions were then classified as *group 1* solutions.

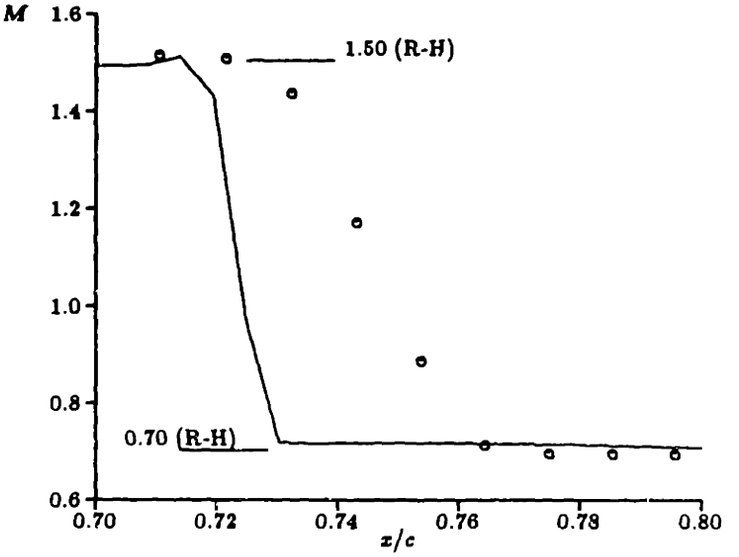
If more than one solution was submitted by an author, only the best was included in *group 2* to eliminate any bias. The criteria used to select the best was based upon largest far-field distance, finest grid, etc. For test case AGARD-06, five solutions appeared in group 2; their computed lift and drag coefficients are summarized in Table 8.3.

Of the group 2 solutions, the Working Group chose solution 9 (a submission by Schmidt and Jameson) as the "best", mainly because of its fine grid (320×64), large far-field radius (50 chords) and crispness of the shock; its force coefficients are given by $C_L = 1.1044$ and $C_D = 0.0448$. Since the publication of reference [2], Pulliam and Barton[71] have computed the same case with the results $C_L = 1.1228$ and $C_D = 0.0464$.

Unfortunately, the adaptively refined solution in section 8.3 resulted in

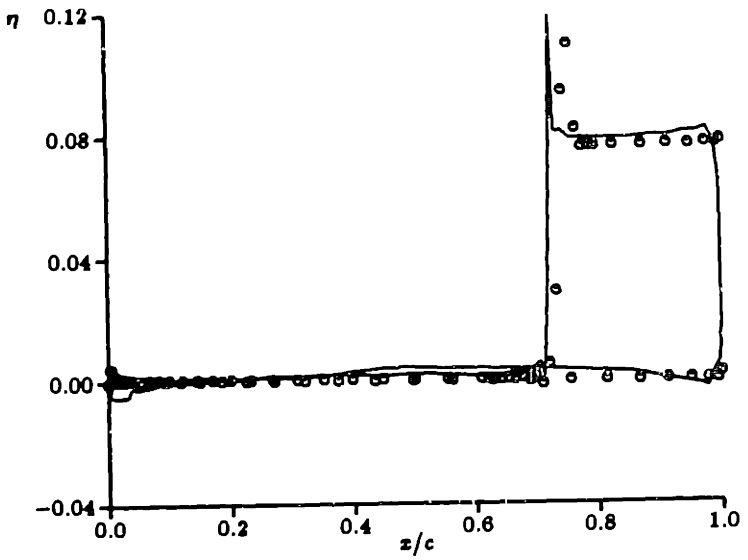


a: entire airfoil

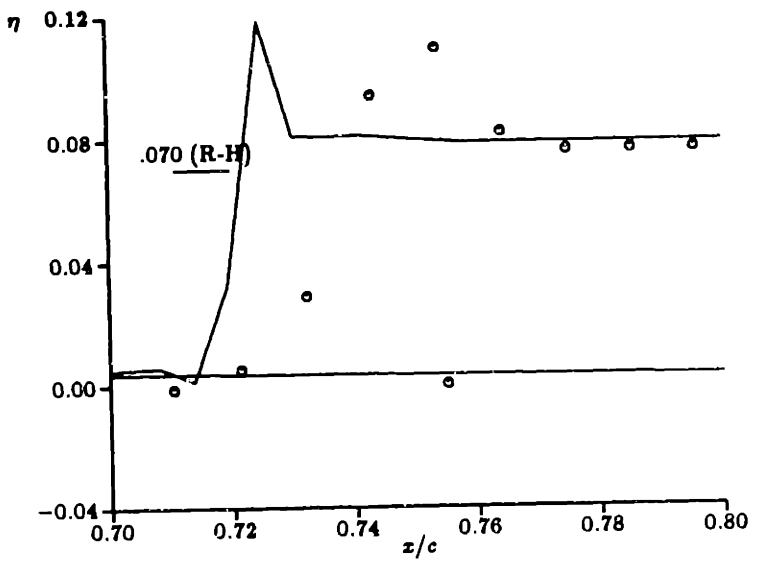


b: near shock

Figure 8.27: Surface Mach number distributions for AGARD-06. Line is from present adaptive solution, symbols are from AGARD reference solution.



a: entire airfoil



b: near shock

Figure 8.28: Surface total pressure loss distributions for AGARD-06. Line is from present adaptive solution, symbols are from AGARD reference solution.

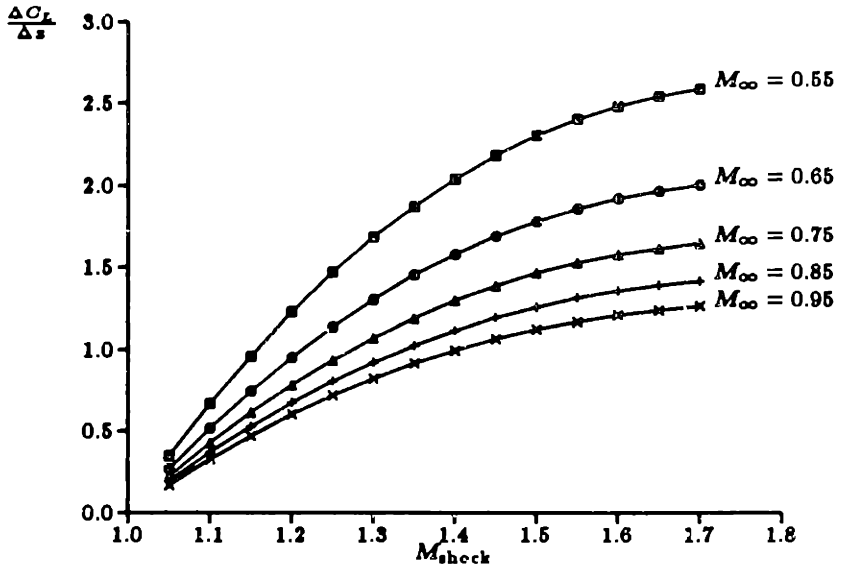


Figure 8.29: Effect of shock movement on computed lift coefficient (as a function of free-stream and shock Mach numbers)

$C_L = 1.0602$ and $C_D = 0.0394$ which is about 4.3% low on lift as compared with the average values in Table 8.3. The surface Mach number and total pressure loss distributions from the final adaptive grid solution and AGARD's "best" are shown in figures 8.27 and 8.28, respectively. The figures show in part a the entire surface distribution and in part b the distribution in the vicinity of the shock. Also shown in the figures are the Rankine-Hugoniot shocks jumps which are well predicted in both solutions. Notice that both shocks are predicted to begin at the same location, but that the adaptively refined solution's shock is much thinner than in the reference solution.

To compute the effect of shock motion (or smearing), a simple model has been employed in which it is assumed that the pressure is constant both upstream and downstream of the shock. If the airfoil surface in the

vicinity of the shock is oriented such that the displacement of the shock only contributes to a change in lift, then the change in lift can be approximated by

$$\Delta C_L = \frac{2(p_u - p_d)}{\rho_\infty U_\infty^2} \Delta x \tag{8.1}$$

where p_u is the pressure just upstream of the shock and p_d is the pressure just downstream of the shock. Equation (8.1) can be rewritten in terms of the free-stream Mach number and the Mach number just upstream of the shock, resulting in figure 8.29, where the abscissa represents the pre-shock Mach number and the ordinate the ratio of the change in lift coefficient to the change in shock location. Five lines are shown, each representing a different free-stream Mach number.

Based upon figure 8.27b, the distance between the mean shock locations is estimated to be $\Delta x \approx 0.022$, resulting in a lift discrepancy of $\Delta C_L \approx 0.033$. This difference accounts for more than half the error between the lift predictions of the current adaptive and the reference solutions.

Unfortunately, the obvious conclusion that the present solution is more accurate because it contains a narrower shock region is not a valid one, even though the majority of the discrepancy between the present adapted and published solutions is due to the differences in the shock region. That conclusion fails when one considers the results of figure 8.24, which states that as the shock is better refined, the computed lift coefficient increases; if the conclusion were valid, one would expect that the lift coefficient computed by the present scheme would be greater than the group 2 results.

Hence some other mechanism must be responsible for the discrepancy between the present and published solutions. In the following sections, several possible sources are examined. To eliminate as many variables as possible in the studies that follow, no grid refinement was allowed in any of the calculations. Furthermore, the convergence criteria was tightened by ensuring that the maximum change (over all nodes) in the x-momentum was less than

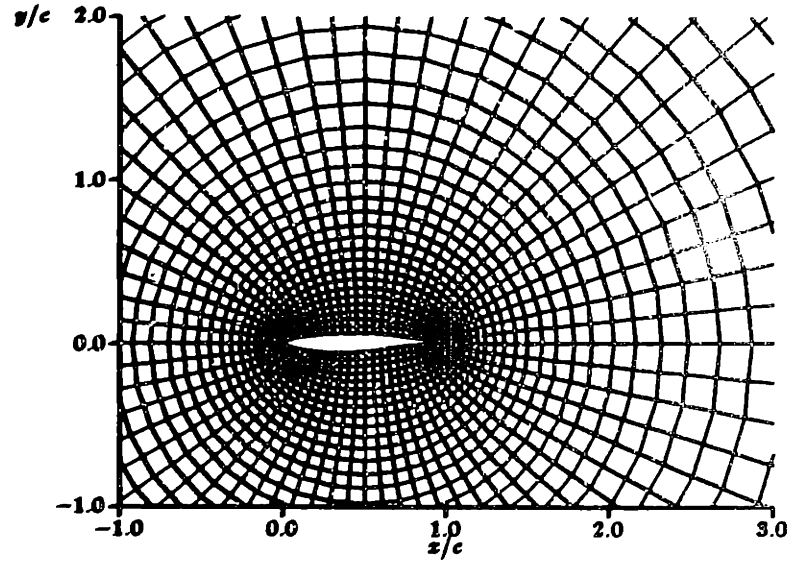


Figure 8.30: Computational grid 64 × 64 used in far-field study

0.00001, or approximately a 5-order drop in the residual from the initial conditions.

8.4.2 Far-field boundary conditions

In the AGARD report of published solutions[2], a considerable effort was expended to determine the effect of the location of and approximations at the far-field boundary. A similar study has been performed for the current integration scheme.

To determine the effect of the location of the far-field boundary, a 64 × 64 O-mesh was generated with its outer boundary placed at 100 chords from the airfoil leading edge. This grid, shown in in figure 8.30 has approximately the same cell aspect ratio as the global grid in section 8.3.

Five successive grids were generated by truncating the 100 chord grid, making the resulting grids all identical in the vicinity of the airfoil. Thus

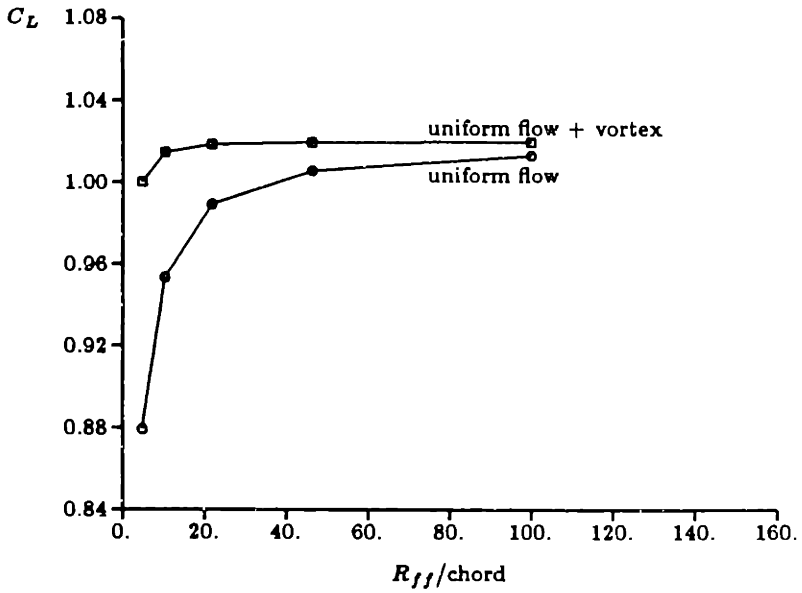
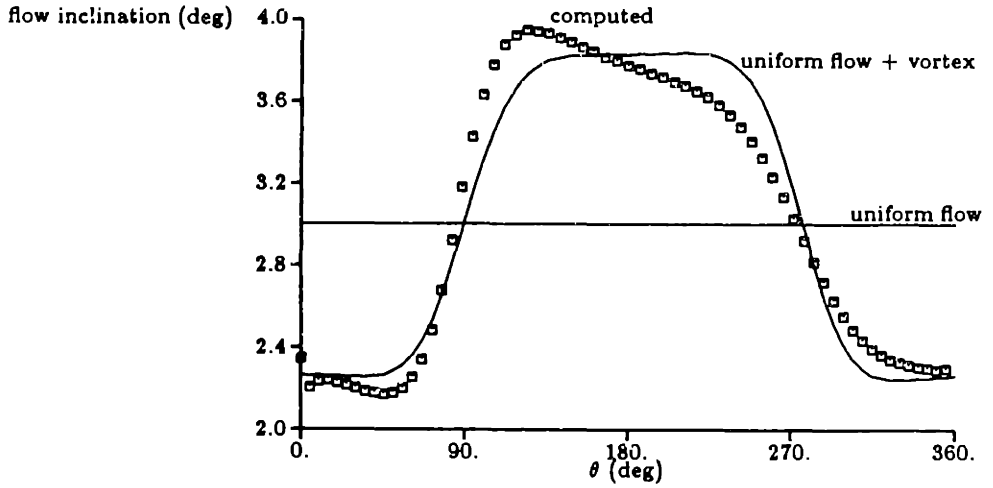


Figure 8.31: Computed lift coefficient as a function of far-field radius

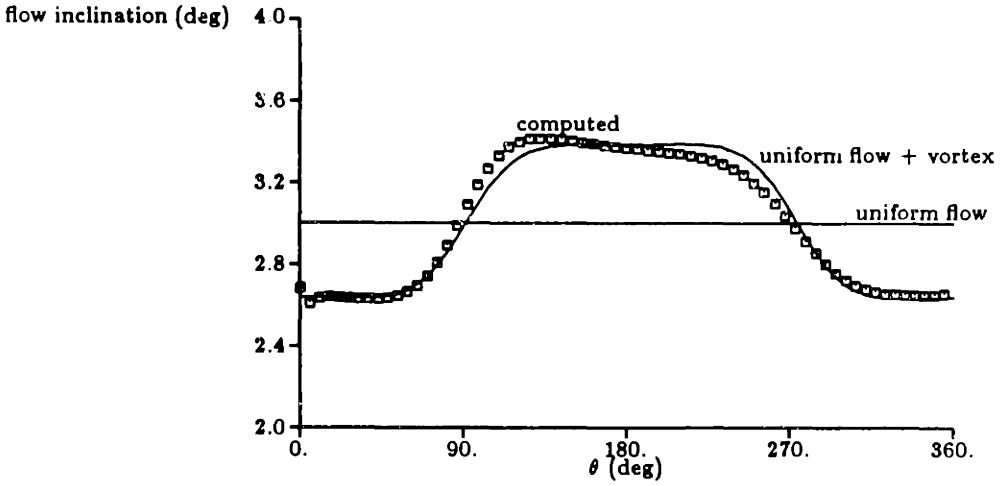
the only difference between the solutions on these grids is due to far-field boundary placement.

Two sets of computations were performed on these five grids. In the first, the far-field was approximated by a uniform flow, whereas in the second, the effect of a vortex centered at the airfoil quarter-chord was included as well. These results are presented in figure 8.31, where the abscissa represents the extent of the computational domain and the ordinate the computed lift coefficient. The figure clearly shows that the lift reaches 1% of its final value at R_{ff}/chord $\approx$ 10 if the vortex correction is applied but not until R_{ff}/chord $\approx$ 50 if only a free-stream approximation is used.

To understand these results, one must consider the influence of the vortex on the far-field solution. Upstream of the airfoil, the major effect of the vortex term in the far-field is to modify the local flow inclination, as can be seen in figure 8.32, which is a plot of the local flow inclination as a function

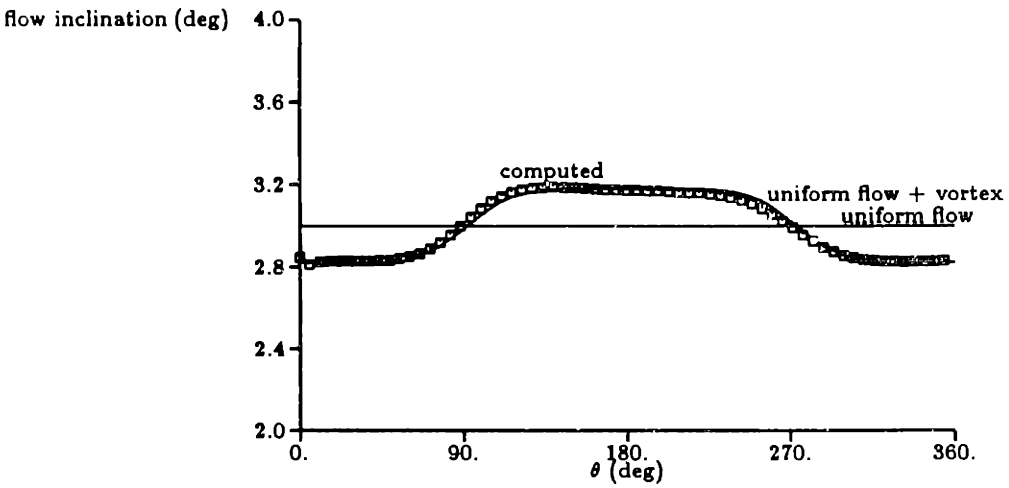


a: O-mesh line at R_{ff}/chord = 4.9

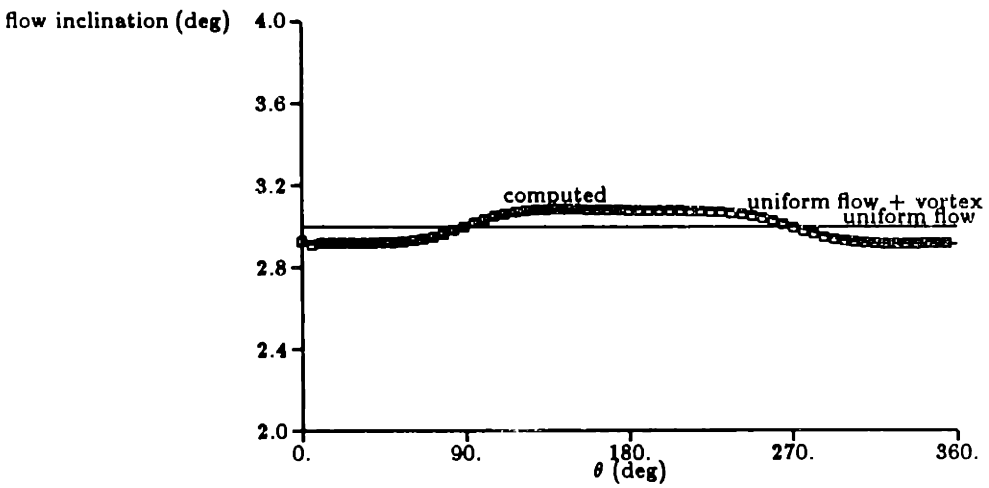


b: O-mesh line at R_{ff}/chord = 10.3

Figure 8.32: Local flow inclination for circuits around AGARD-06



c: O-mesh line at R_{ff}/chord = 21.8



d: O-mesh line at R_{ff}/chord = 46.4

Figure 8.32: Local flow inclination for circuits around AGARD-06

of angular position relative to the airfoil (180° corresponds to upstream of the airfoil). Each part of the figure corresponds to a traverse of one of the O-mesh grid lines. The computed flow inclinations, which are plotted as symbols, are taken from the calculation in a domain whose far-field boundary was placed 100 chords from the airfoil. Also shown in the figure are the predicted flow inclinations as computed using a uniform flow and a uniform flow plus vortex.

As can be seen, the computed values agree well with the higher order approximation, whereas the agreement with the uniform flow model is poor. This is to be expected based upon the poor accuracy of flow predictions for the uniform flow approximation as seen in figure 8.31.

A qualitative assessment of the vortex's importance is obtained if one assumes that a flow inclination error on the upstream stagnation streamline acts as an angle of attack error on the airfoil. Using the flat plate lift-curve slope ($2\pi/\beta$) (which is corrected for compressibility effects) as an approximation for this airfoil, the 0.8° error at $\theta = 180°$ for $R_{ff}/\text{chord} = 4.9$ can be translated into a lift error of $\Delta C_L = 0.1326$ which accounts for the lift error as shown in figure 8.31.

In summary, the adapted solution computed in section 8.3 used the far-field vortex correction applied at $R_{ff}/\text{chord} = 10.0$. Figure 8.31 shows that the expected influence of this far-field approximation is on the order of one percent. Hence, it does not appear that the far-field approximation is a major contributor to the lift error between the present and computed solutions.

8.4.3 Trailing edge

Another boundary condition assumption made in the calculations was to not impose a flow angle at the trailing edge node. In this section, the effect of that approximation is discussed.

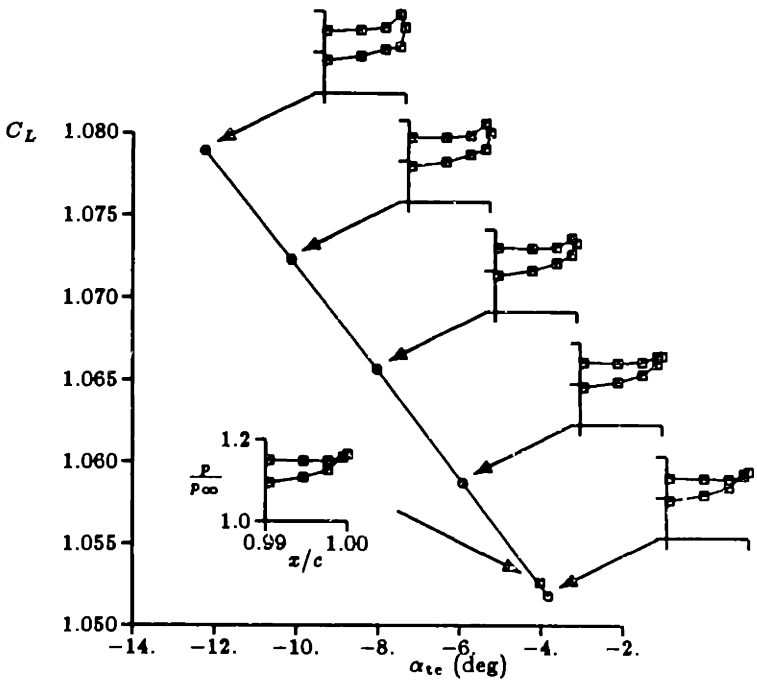


Figure 8.33: Effect of trailing edge flow angle on computed lift coefficient, test case AGARD-06

To determine the effect of the trailing edge flow inclination, five calculations were performed in which a flow angle was *imposed* at the trailing edge node. The directions were chosen to be tangent to the upper surface, tangent to the lower surface, and three intermediate conditions. The computed lift coefficients for these cases are plotted as circles in figure 8.33 as a function of the trailing edge flow angle ($-4°$ corresponds to tangency to the lower surface). Notice that the lift coefficient varies by only about 3% for these calculations. Also plotted as a square is the computed lift and flow angle from a computation in which the flow angle was not imposed; the latter point falls on the same line defined by connecting the former results.

The figure also contains inset plots of the static pressure distribution over the last 1% of chord for each of the calculations; appropriate scales are included on the inset plot for the solution in which the angle was not imposed. These inset plots clearly show the pressure difference across the airfoil one node upstream of the trailing edge. When the flow leaves tangent to the upper surface ($\alpha_{te} \approx -12°$), there is about a 10% pressure difference in the pressures whereas for tangency to the lower surface the pressures are nearly identical. According to the Kutta condition, the latter is the expected behavior, which also is in agreement with the conclusions of section 2.6.2.

In the present adaptive scheme, the trailing edge boundary condition allows the flow angle to float to its preferred direction as discussed in section 3.2.2.3. Figure 8.33 clearly shows that such a trailing edge treatment yields results which are consistent with the expected behavior. Hence the trailing edge boundary condition is not responsible for the lift discrepancy between the present adapted and published solutions.

8.4.4 Grid aspect ratio

Numerical schemes are often influenced by the aspect ratio of computational cells, especially in the vicinity of bodies such as airfoils. In order to

determine the effect of grid aspect ratio on the present solution technique, a series of three computations were performed. In each case, the far-field boundary was placed at ten chords from the airfoil. Figure 8.34 shows the largest and smallest aspect ratio grids.

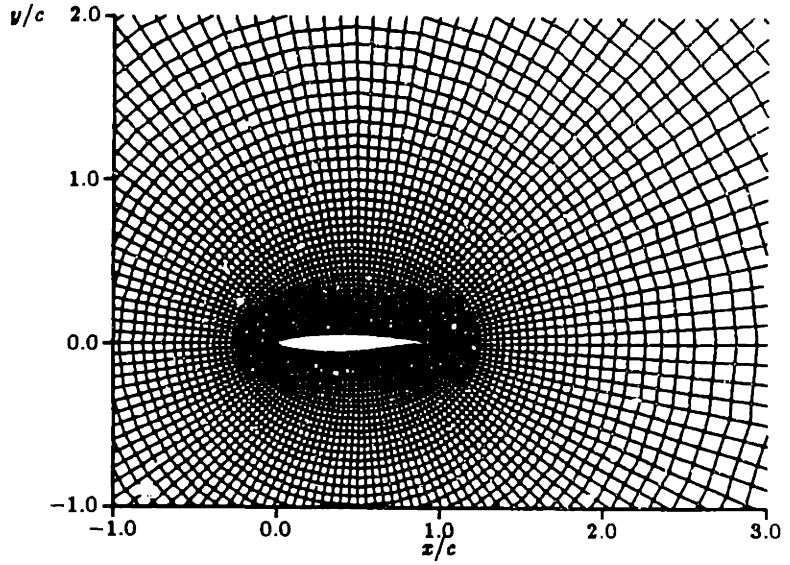
The computed lift coefficients for these cases are shown in figure 8.35, plotted as a function of the number of cells between the airfoil surface and the far-field boundary. Notice that as the grid spacing in the direction normal to the airfoil surface decreases, the lift coefficient increases slightly (only about 1%). This is consistent with the behavior noted in section 8.3, where the finest computational grids yielded the largest lift coefficients. In fact, the magnitude of the difference in the lift coefficients is the same as the difference between the predicted values for solutions on comparable grids (grid 2 and grid 3) in section 8.3. Hence the differences noted in figure 8.35 may actually result from the grid size change and not the cell aspect ratio differences.

8.4.5 Smoothing

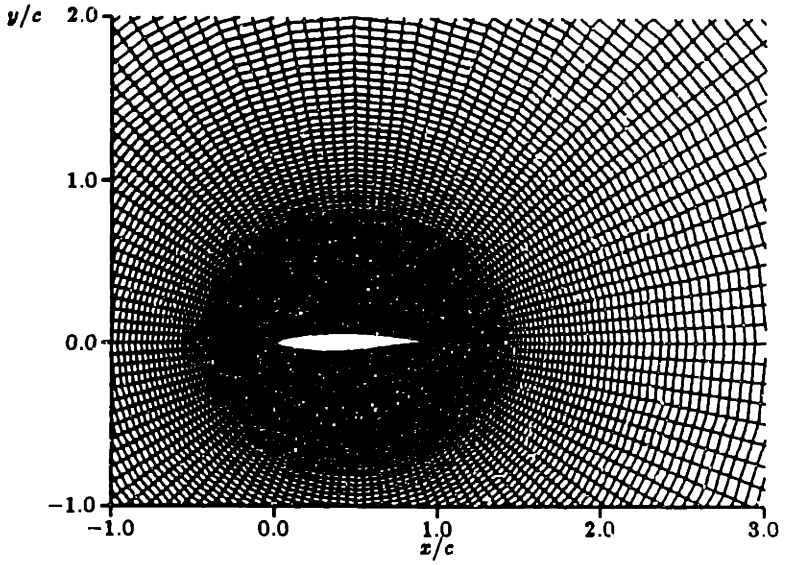
In order to compute solutions containing discontinuities such as shock waves, artificial viscosity (or smoothing) must be added to the basic scheme as discussed in section 3.5. In order to rule it out as the source of the lift discrepancy, a series of test cases were performed in which the level of the minimum smoothing coefficient was varied.

Figure 8.36 contains the surface Mach number distributions for two different levels of smoothing; the level of smoothing used in section 8.3 is the average of the two shown here. Notice that the distributions are very similar, the major difference being in the oscillations before and after the shock.

The integrated effect of these difference is shown in figure 8.37, where the lift coefficient for various solutions is plotted versus the minimum smoothing coefficient. (Recall from section 3.5 that the level of the smoothing coefficient



a: 128 × 64 grid



b: 128 × 128 grid

Figure 8.34: Computational grids in the vicinity of the airfoil for test case AGARD-06

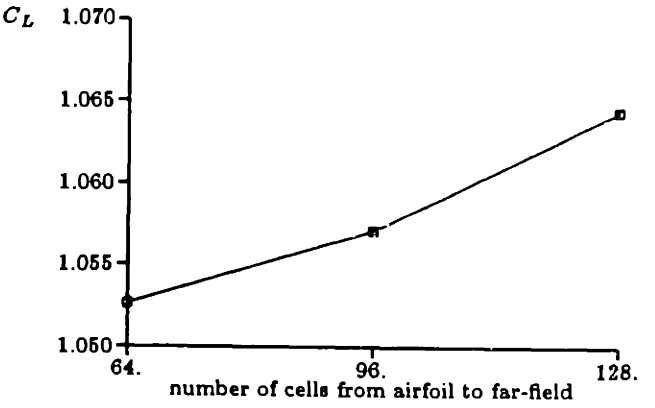
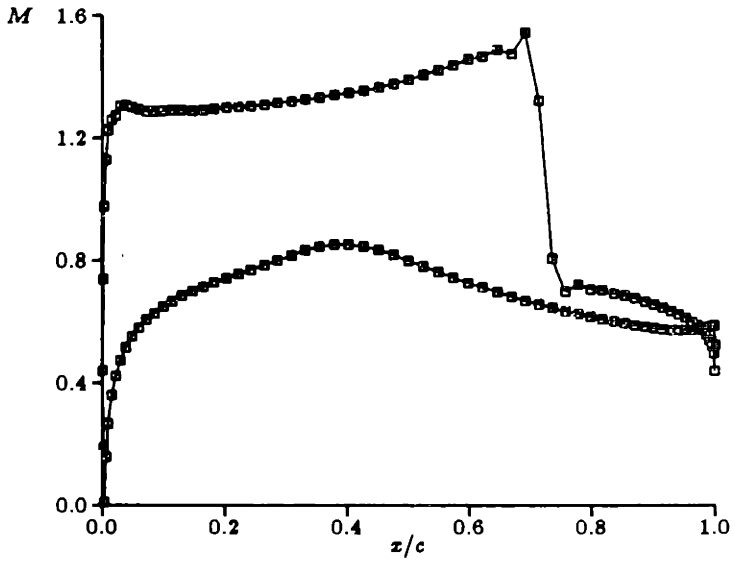


Figure 8.35: Effect of cell aspect ratio on the computed lift coefficient, test case AGARD-06

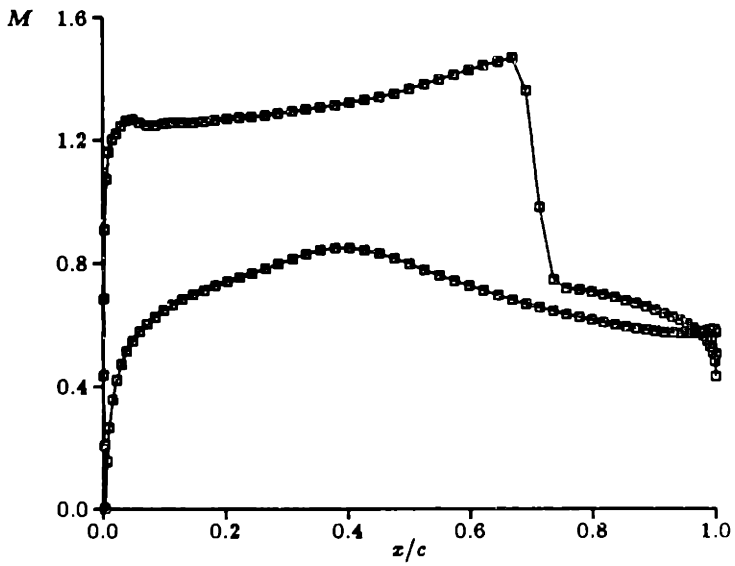
varies spatially). Also plotted in the figure is a $*$, indicating the extrapolated lift coefficient for a solution without smoothing, even though such a solution is not possible with this scheme.

Hence, even though smoothing seems to have a modest effect on the computed lift coefficient, it is not possible to conclude that it is the reason why solutions with the current program do not match the published solutions.

In conclusion, none of the effects described in the last four sections exhibit sufficient influence to be the source of the lift discrepancy. All of those check solutions were computed on a fixed global grid without embedding and all were well converged. Unless some other important effect has been neglected, there does not seem to be any cause for the current lift prediction to be in error. Therefore at the present time, it is not clear whether the present results are in error, or whether the published results are incorrect (or perhaps neither is correct).



a: Low level of smoothing ($\mu_{\min} = 0.010$)



b: High level of smoothing ($\mu_{\min} = 0.030$)

Figure 8.36: Computed surface Mach number distributions with different levels of smoothing for test case AGARD-06

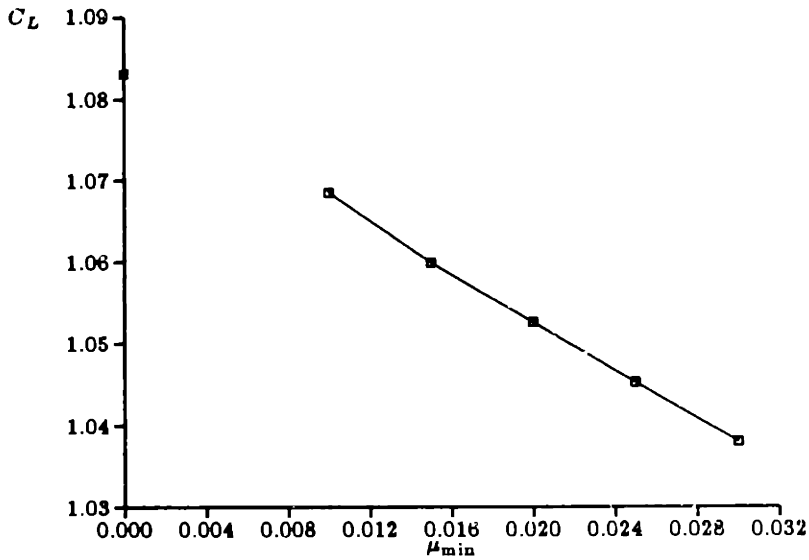


Figure 8.37: Effect of smoothing on the computed lift coefficient, test case AGARD-06

8.5 Sensitivity of Adaptation Algorithm

In section 8.3, an arbitrary choice of refinement parameter and threshold values was used to yield the adaptively refined solution. In this section, the effects of choosing alternative refinement parameters and thresholds is examined.

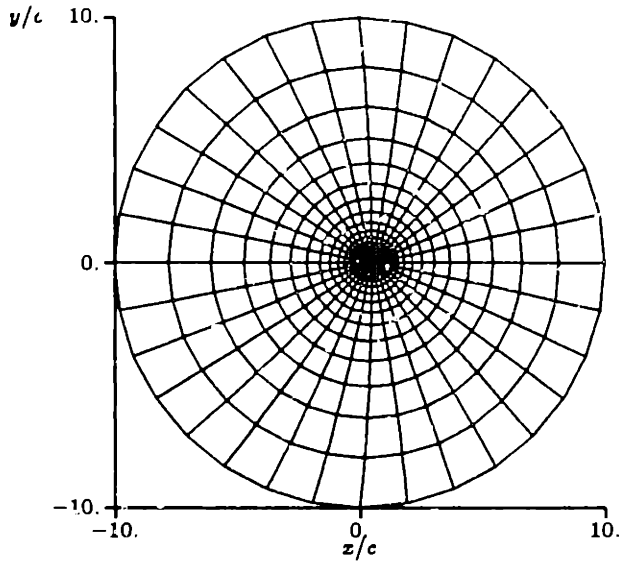
8.5.1 Refinement parameter

As stated in section 6.3, the refinement parameter can be any combination of the flow field variables whose value is larger at the expected features than elsewhere in the flow domain. Figures 6.2 through 6.13 contain contours of twelve candidate refinement parameters for case AGARD-06. Based upon these contours, it was decided that the first difference of density, $\tilde{\nabla}\rho$, would be a reasonable refinement parameter choice for such transonic, airfoil flow fields.

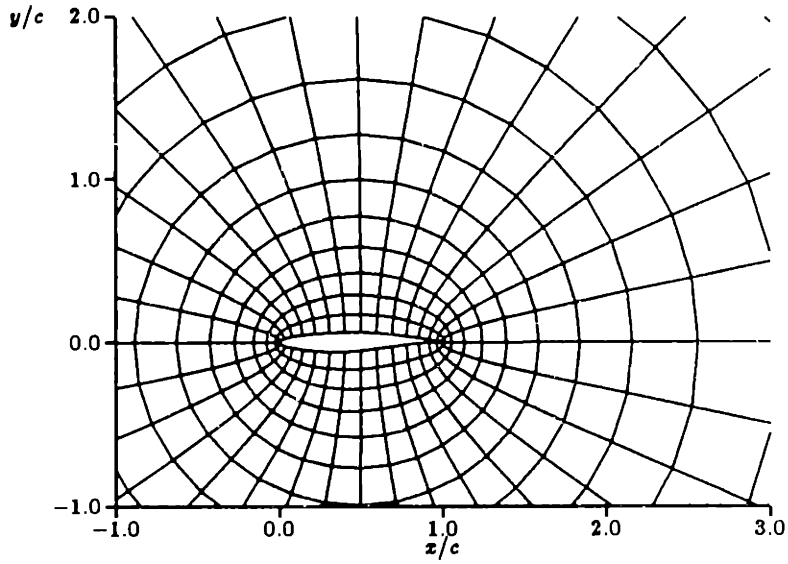
In order to determine the importance of that refinement parameter choice, a series of calculations were performed for case AGARD-06 using each of the twelve refinement parameters identified in section 6.3. The resulting embedded grids are shown in figures 8.38 through 8.49. Each figure contains two views of the final computational grid which results from applying the MITOSIS program with the automatic threshold selector. In each case, the number of cycles of refinement (and hence grid levels) was determined automatically by the adaptation algorithm contained in the knowledge base.

There are a number of interesting features to note from the cases which use density (or one of its derivatives) as the refinement parameter (figures 8.38 through 8.40).

First, direct use of the density as the refinement parameter results in no adaptation; this is to be expected because nowhere in the flow field does the density exceed 1.25 times the average density; the 1.25 results from a guard placed in the threshold selection algorithm. This makes density a clearly

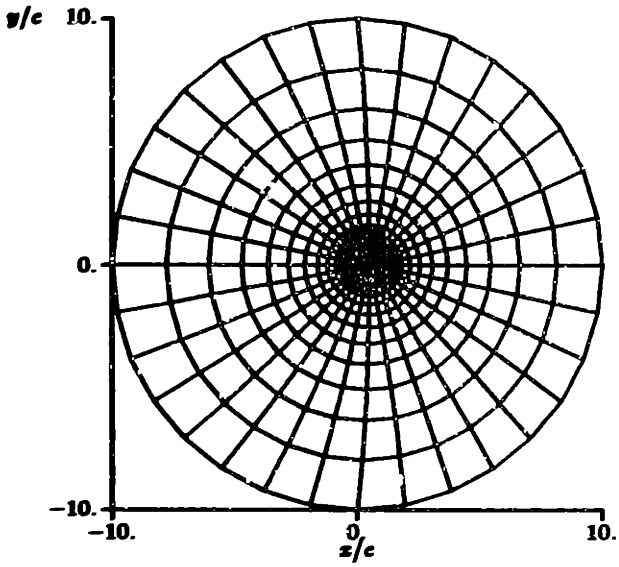


a: entire grid

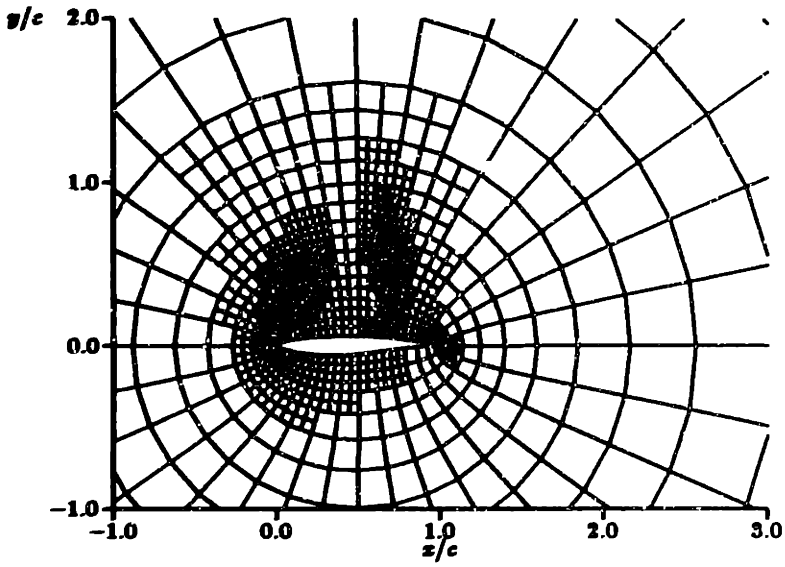


b: near airfoil

Figure 8.38: Final computational grid for $\mathcal{R} = \rho$, test case AGARD-06.

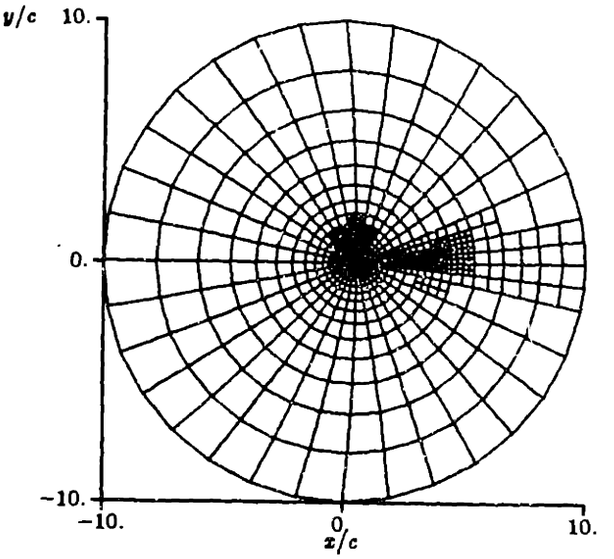


a: entire grid

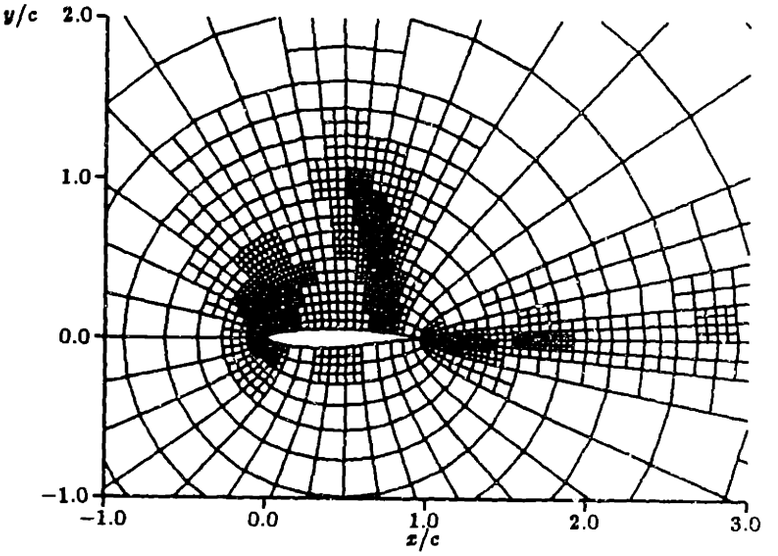


b: near airfoil

Figure 8.39: Final computational grid for $\mathcal{R} = \tilde{\nabla}\rho$, test case AGARD-06.



a: entire grid



b: near airfoil

Figure 8.40: Final computational grid for $\mathcal{R} = \tilde{\nabla}^2\rho$, test case AGARD-06.

unacceptable refinement parameter for flow fields such as this one.

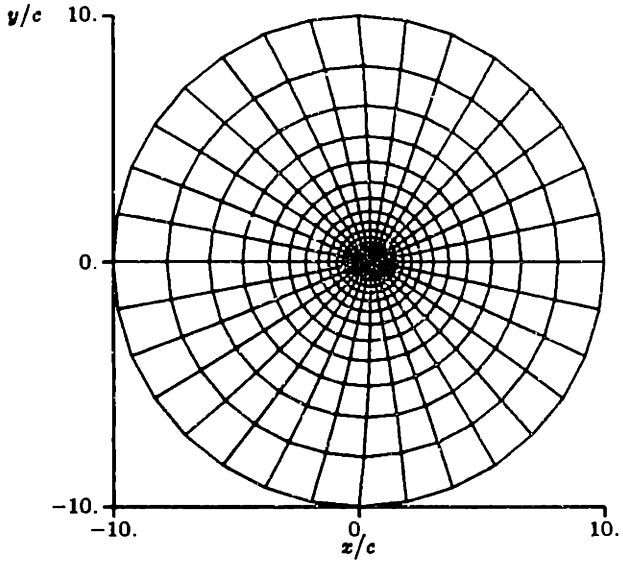
Second, the first difference of density ($\tilde{\nabla}\rho$) yields adapted regions which clearly follow the shock and leading edge expansion regions. This refinement parameter results in modest lower surface and trailing edge resolution, but no adaptation in the wake region.

Third, the second difference ($\tilde{\nabla}^2\rho$) results in adapted regions near the shock, leading edge expansion region, and all the way to the far-field boundary in the wake. The figure (8.40) also clearly shows that these regions are not smoothly shaped, especially in the wake region where isolated embedded regions are seen to exist. This is due to the difficulty encountered in accurately computing the required second differences. Since errors are introduced at the embedded mesh interfaces (see chapter 4), such behavior should be avoided if possible.

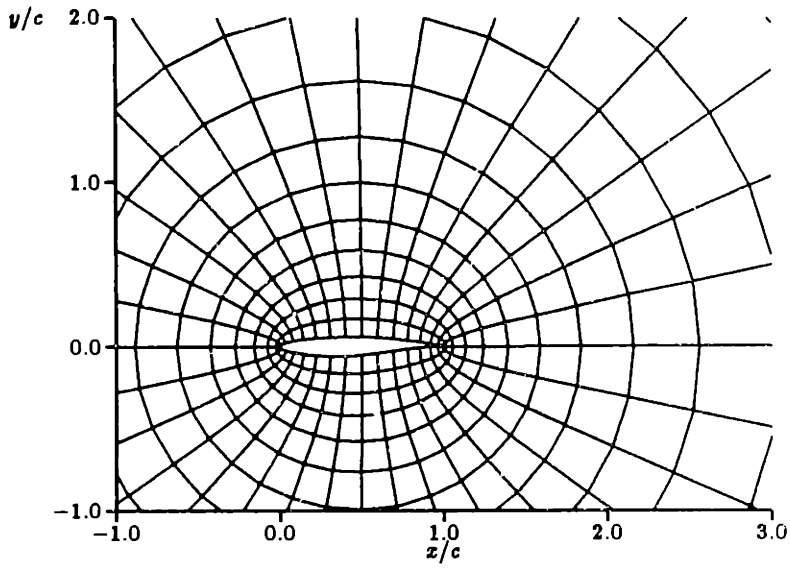
The solutions which use pressure or one of its derivatives as the refinement parameter (figures 8.41 through 8.43) show generally the same behavior as do the density cases. One important distinction is in the second difference case (figure 8.43), where the wake is much less resolved than in figure 8.40.

In a similar manner, the velocity magnitude and its derivatives exhibit the same behavior as the density and its derivatives. Here the major difference is contained in the $\mathcal{R} = q$ case, where a small number of cells in the supersonic region were adapted. Unfortunately, these cells turn out to be in the smooth region of the supersonic flow where enhanced refinement is not required.

Alternatively, figures 8.47 through 8.49 demonstrate the behavior of using the local total pressure loss or its derivatives. In all three of these cases, the wake caused by the shock is very well resolved with multiple levels of refinement. The shock (and to a lesser degree the leading edge expansion) are well embedded for the second difference case (figure 8.49).

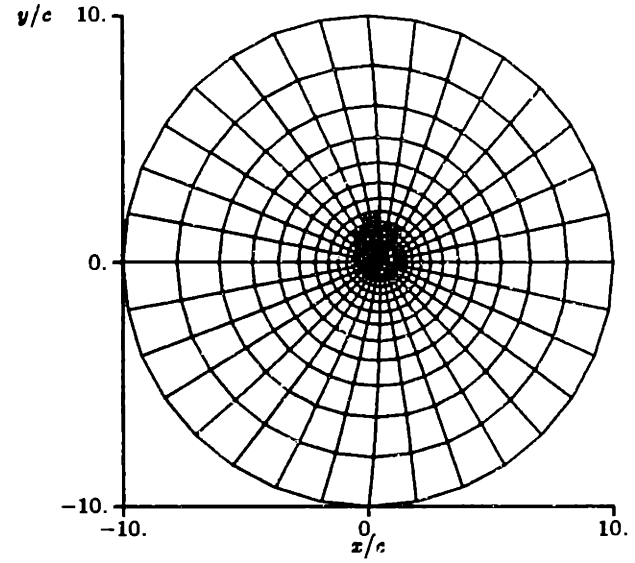


a: entire grid

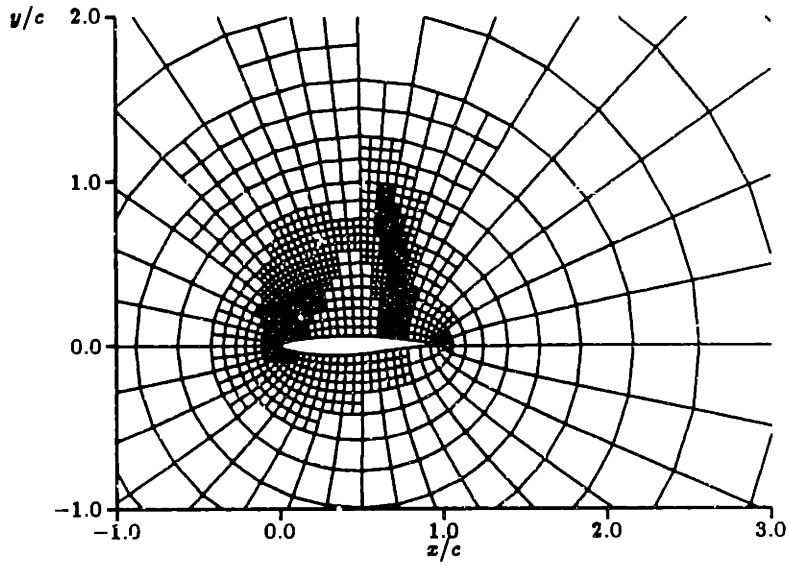


b: near airfoil

Figure 8.41: Final computational grid for $\mathcal{R} = p$, test case AGARD-06.

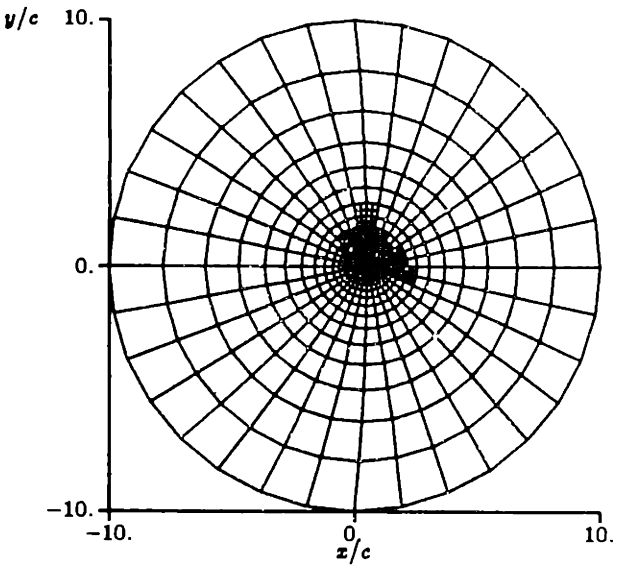


a: entire grid

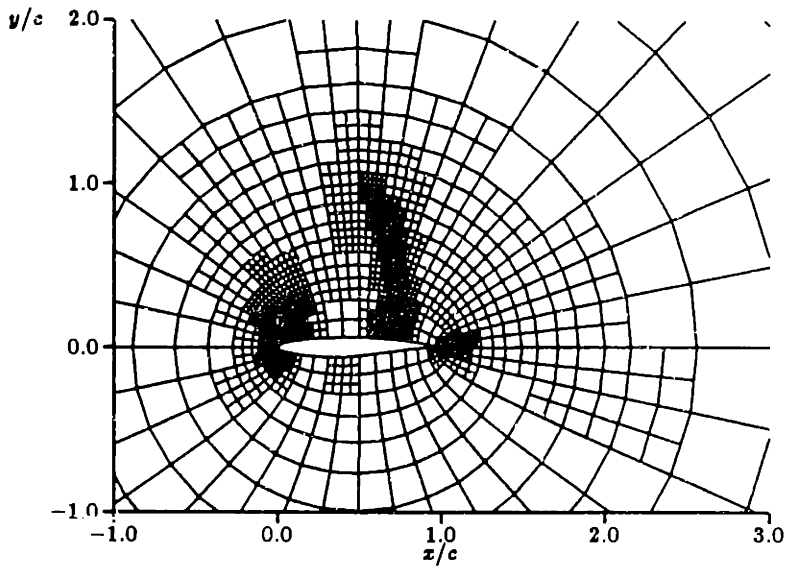


b: near airfoil

Figure 8.42: Final computational grid for $\mathcal{R} = \tilde{\nabla} p$, test case AGARD-06.

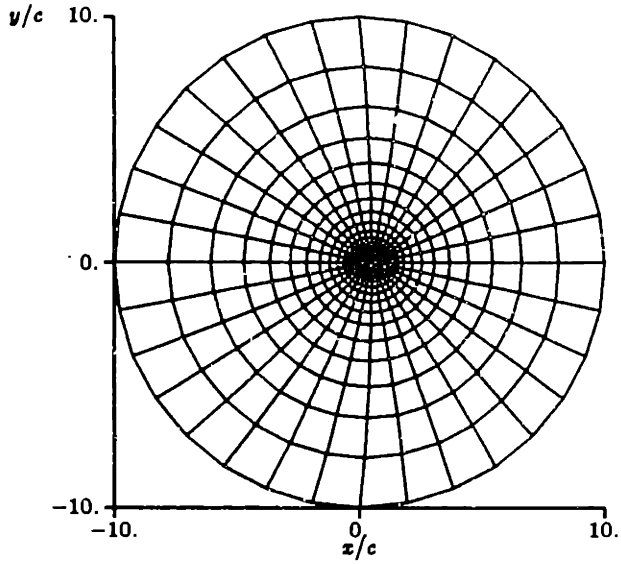


a: entire grid

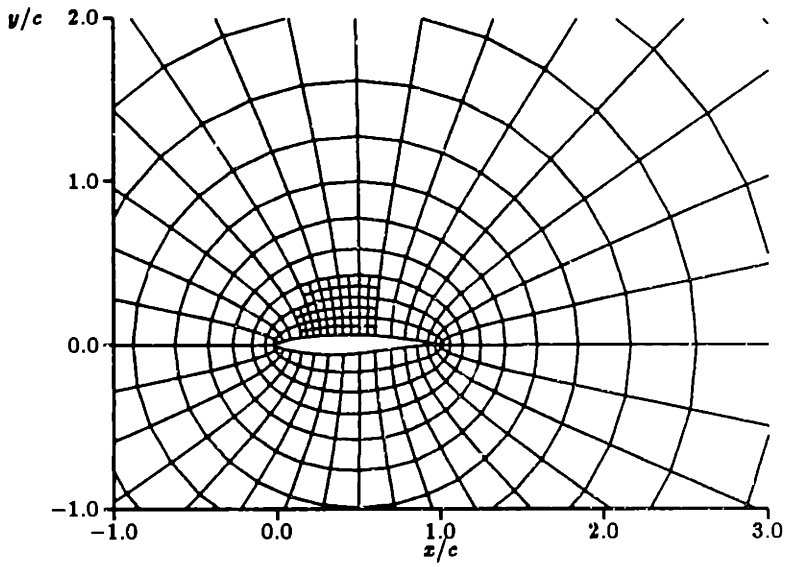


b: near airfoil

Figure 8.43: Final computational grid for $\mathcal{R} = \tilde{\nabla}^2 p$, test case AGARD-06.

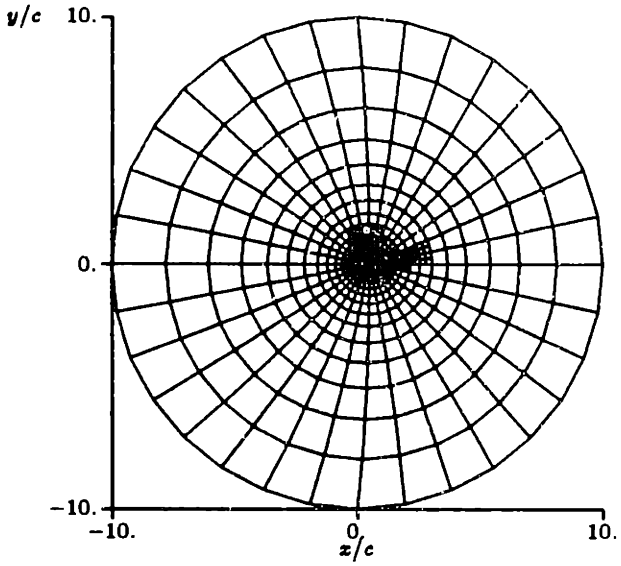


a: entire grid

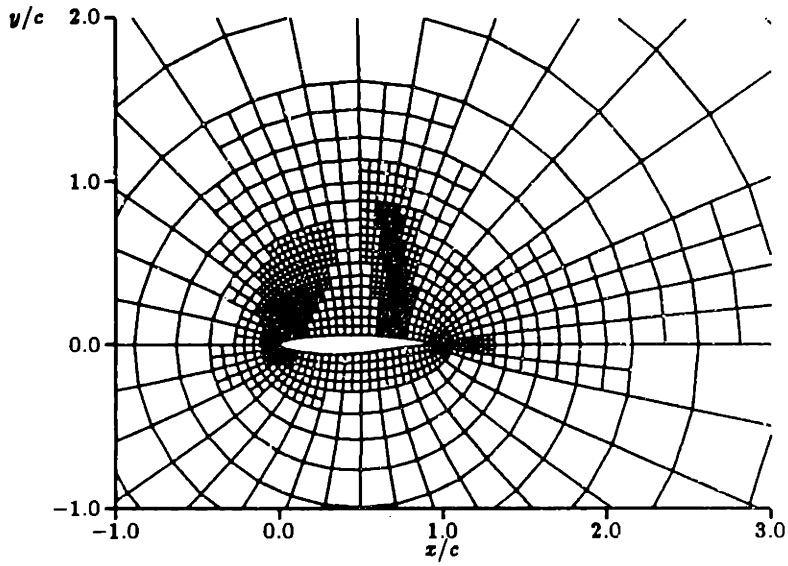


b: near airfoil

Figure 8.44: Final computational grid for $\mathcal{R} = q$, test case AGARD-06.

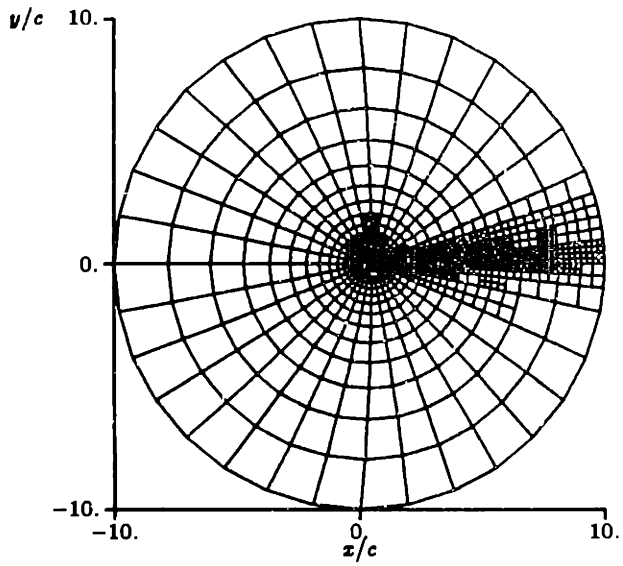


a: entire grid

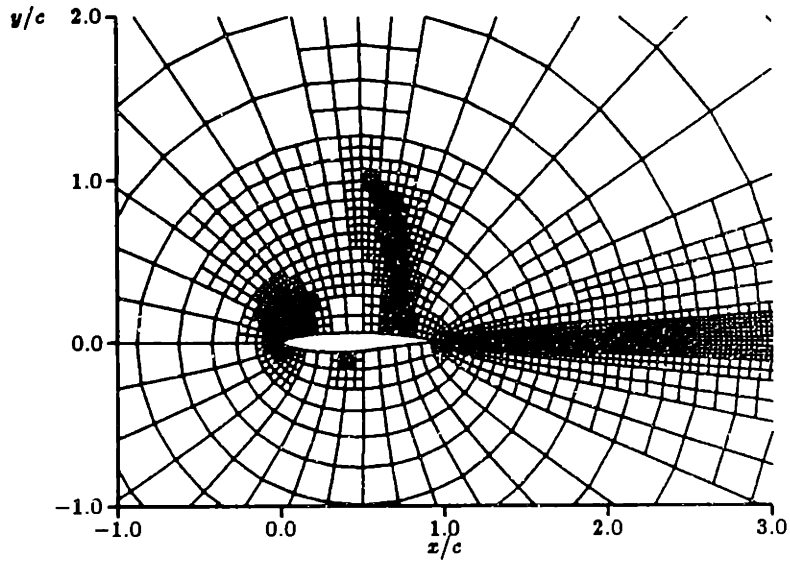


b: near airfoil

Figure 8.45: Final computational grid for $\mathcal{R} = \tilde{\nabla} q$, test case AGARD-06.

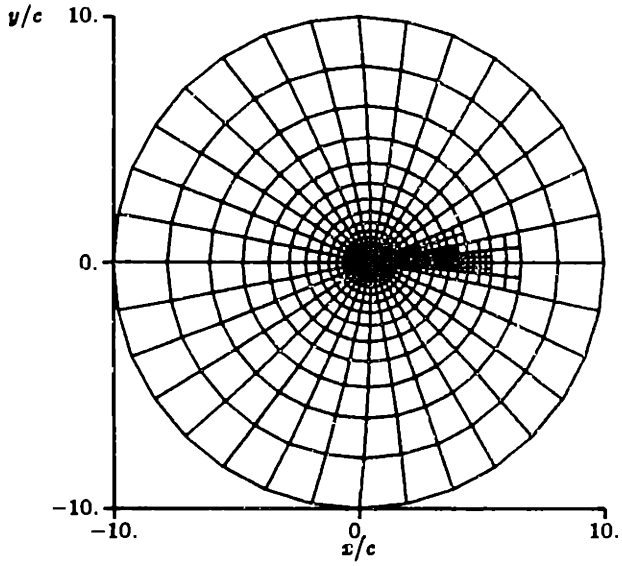


a: entire grid

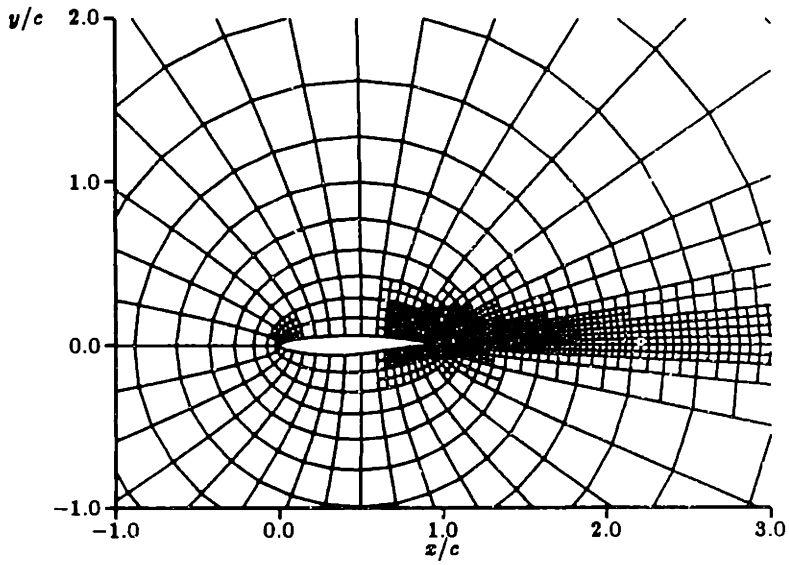


b: near airfoil

Figure 8.46: Final computational grid for $\mathcal{R} = \tilde{\nabla}^2 q$, test case AGARD-06.

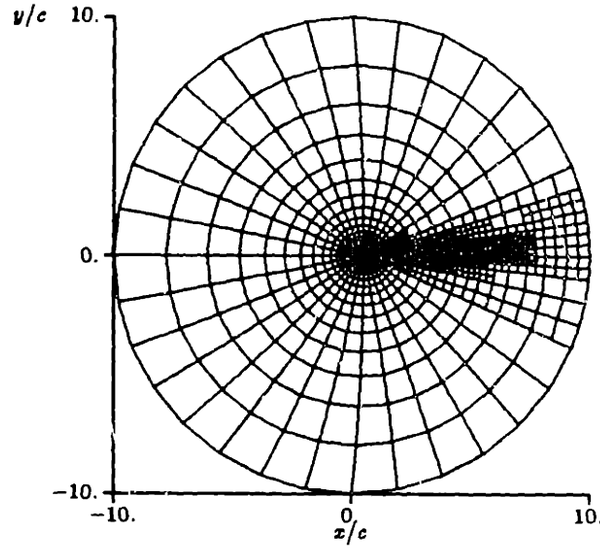


a: entire grid

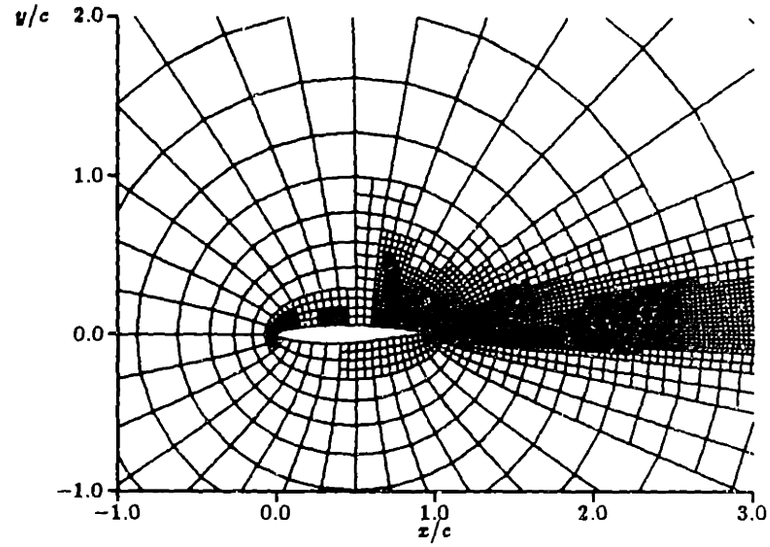


b: near airfoil

Figure 8.47: Final computational grid for $\mathcal{R} = \eta$, test case AGARD-06.

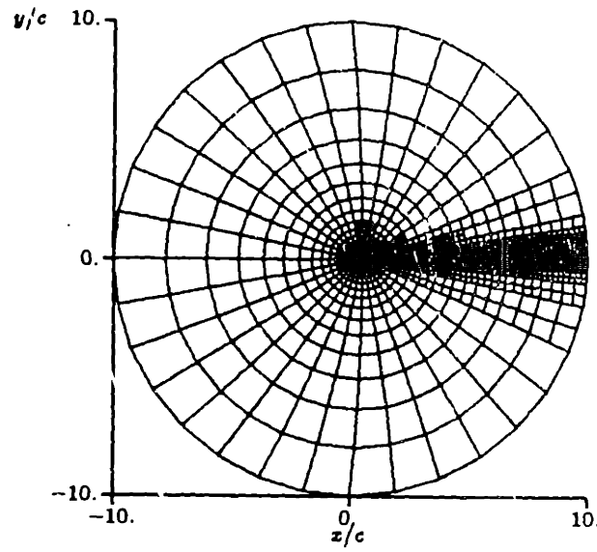


a: entire grid

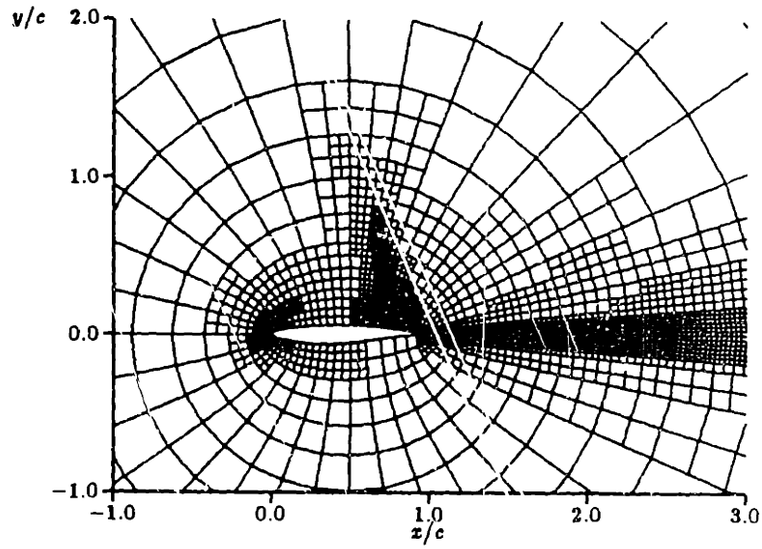


b: near airfoil

Figure 8.48: Final computational grid for $\mathcal{R} = \tilde{\nabla}\eta$, test case AGARD-06.



a: entire grid



b: near airfoil

Figure 8.49: Final computational grid for $\mathcal{R} = \tilde{\nabla}^2\eta$, test case AGARD-06.

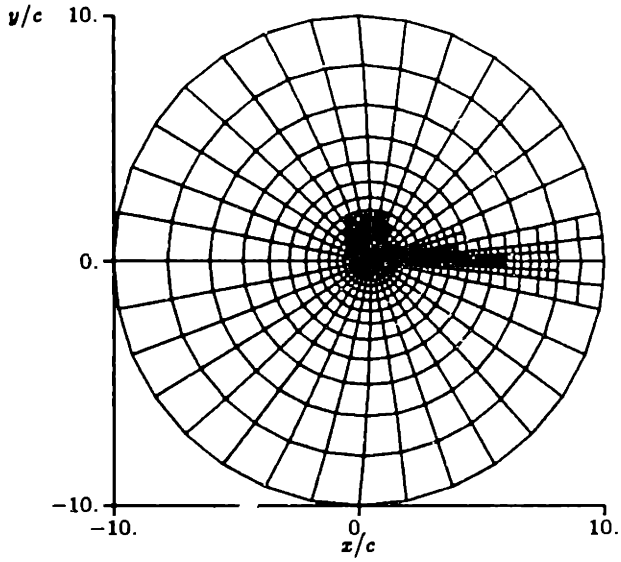
It is also possible to use multiple refinement parameters in a single computation. The effect of such a scheme is shown in figure 8.50. Here the first difference of density $\tilde{\nabla}\rho$ was used to embed in the shock and leading edge expansion regions and the total pressure loss η was used to ensure refinement in the wake.

The results of an alternative combination approach is shown in figure 8.51, where a combinations of the first density difference $\tilde{\nabla}\rho$ and a trailing edge refinement singularity were employed. Notice from the figure that in this case the shock, leading edge, and trailing edge regions are all well refined.

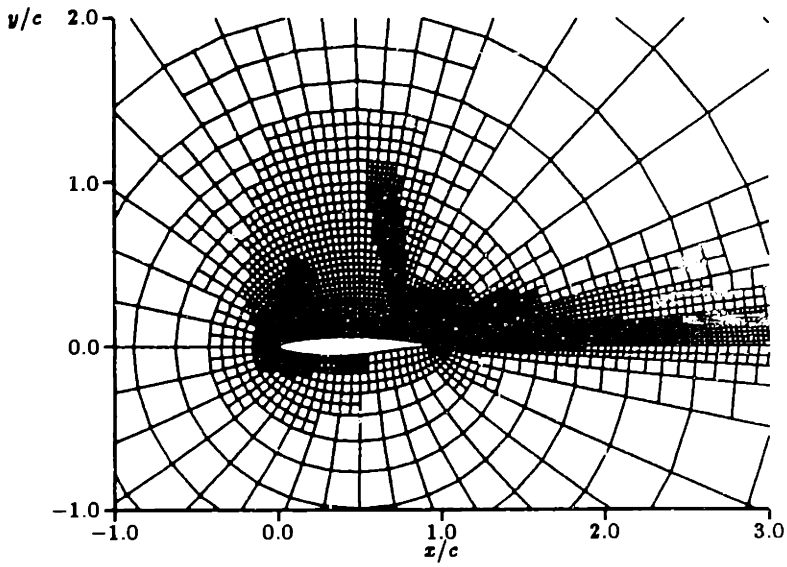
In order to determine the effectiveness of the various refinement parameters, the evolution of the lift coefficient is plotted for each of the fourteen cases as a function of the grid refinement cycle, as shown in figure 8.52. Also shown in the plot as squares is the lift evolution for a globally refined solution. (The latter only was computed to cycle 3 because of computer memory limitations.) The figure shows that for the majority of refinement parameters, the lift evolution is in very good agreement with the evolution on a globally refined grid.

The figure also shows that using the density, velocity, pressure, or pressure loss directly produces erroneous results. Hence these should not be used in transonic airfoil calculations.

To determine the relative effectiveness of the various cases, the final computed lift coefficient in each case is plotted in figure 8.53 as a function of the pseudo-CPU time required to achieve the solution. The figure shows that all of the simple refinement parameters achieve solutions within 1% of one another in a time between 3000 and 5500; the solutions with multiple refinement parameters each require a time of about 8000 whereas the globally refined solution requires approximately 17000. The data from this figure is summarized in Table 8.4.

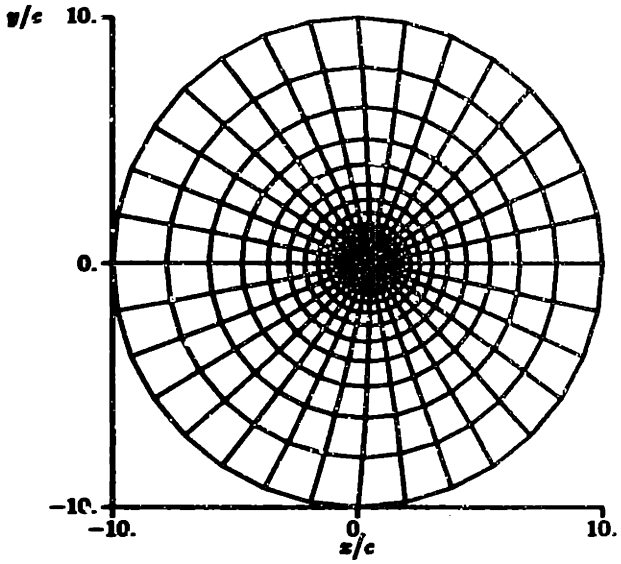


a: entire grid

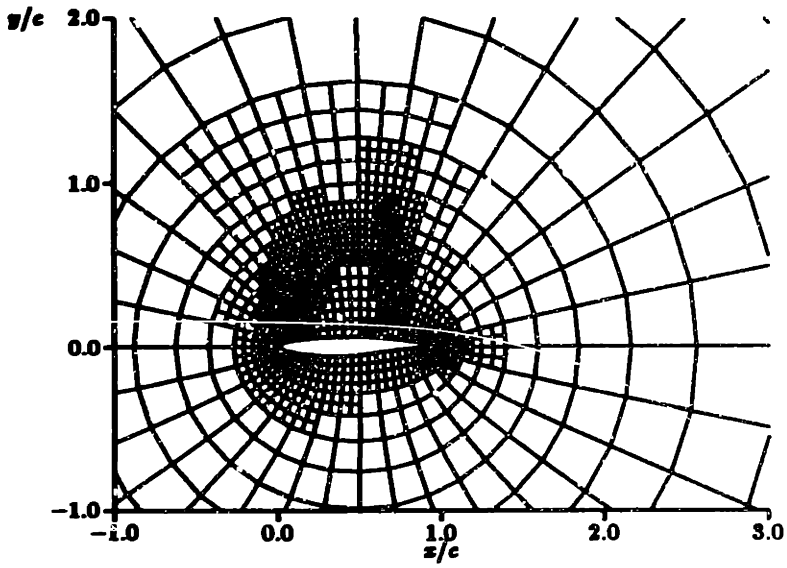


b: near airfoil

Figure 8.50: Final computational grid for $\mathcal{R} = \tilde{\nabla}\rho$ and $\mathcal{R} = \eta$, test case AGARD-06.



a: entire grid



b: near airfoil

Figure 8.51: Final computational grid for $\mathcal{R} = \tilde{\nabla}\rho$ with refinement singularity at trailing edge, test case AGARD-06.

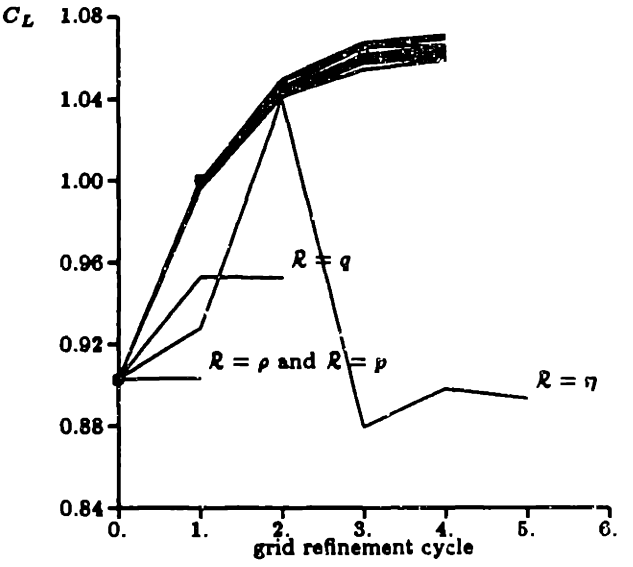


Figure 8.52: Computed lift coefficient versus grid refinement cycle for various refinement parameters, automatically computed thresholds, test case AGARD-06.

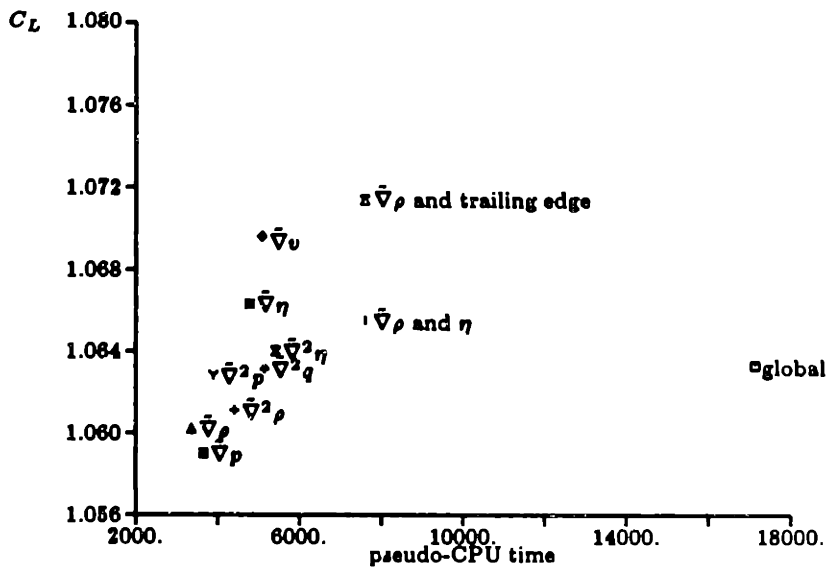


Figure 8.53: Computed lift coefficient versus required pseudo-CPU time for selected refinement parameters, test case AGARD-06.

$\mathcal{R}$	number of adaptation cycles	final number of nodes	C_L	C_D	total number of iterations	total pseudo-CPU time
global	3	33024	1.0633	0.0392	389	17142
ρ	1	544	0.9029	0.0363	165	449
$\tilde{\nabla}\rho$	4	3225	1.0602	0.0394	458	3352
$\tilde{\nabla}^2\rho$	4	4804	1.0611	0.0393	452	4397
q	2	594	0.9525	0.0392	264	751
$\tilde{\nabla}q$	4	3695	1.0696	0.0403	532	5066
$\tilde{\nabla}^2q$	4	6543	1.0631	0.0396	451	5126
p	1	544	0.9029	0.0363	165	449
$\tilde{\nabla}p$	4	3320	1.0590	0.0395	469	3645
$\tilde{\nabla}^2p$	4	4166	1.0628	0.0398	455	3886
η	5	2716	0.8931	0.0276	673	6184
$\tilde{\nabla}\eta$	4	5903	1.0663	0.0407	457	4767
$\tilde{\nabla}^2\eta$	4	6829	1.0640	0.0399	450	5396
$\tilde{\nabla}\rho$ and η	5	16260	1.0674	0.0393	512	12401
$\tilde{\nabla}\rho$ and t.e.	4	7181	1.0714	0.0403	521	7588

Table 8.4: Summary of solutions to AGARD-06 computed with various refinement parameters

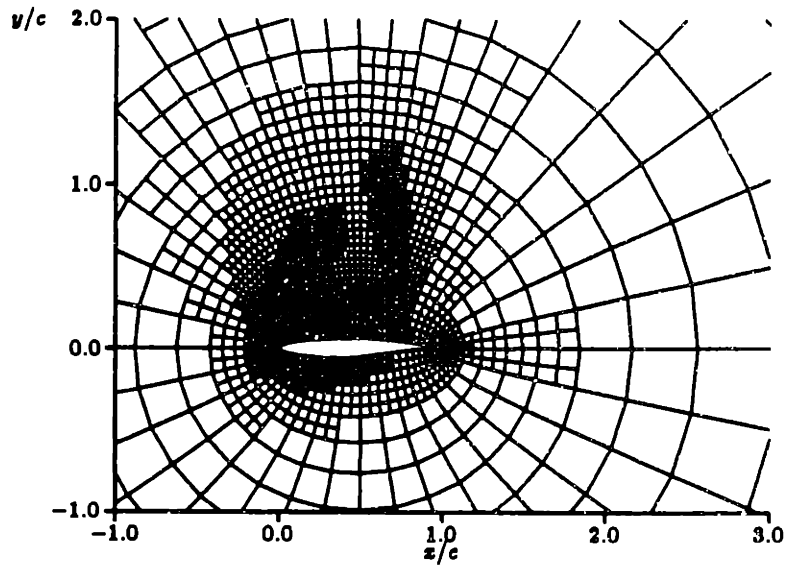


Figure 8.54: Final computational grid for $\mathcal{R}_d = 1.0$, test case AGARD-06.

One can note from the figure that a wide variety of embedded grid shapes all yield approximately the same computed force coefficients. This implies that the exact choice of refinement parameter definition is not critical, as long as those which do not detect the features at all (the values of the density, velocity, pressure, and total pressure loss) are avoided. Hence the choice of first difference of density in section 8.3 is in fact reasonable (for transonic flow).

8.5.2 Threshold selection

The other arbitrary choice which was made when constructing the feature detector was the threshold selection algorithm. To determine the importance of the threshold value used in adaptive grid calculations, five test cases were performed for AGARD-06, with the first difference of density used as the refinement parameter.

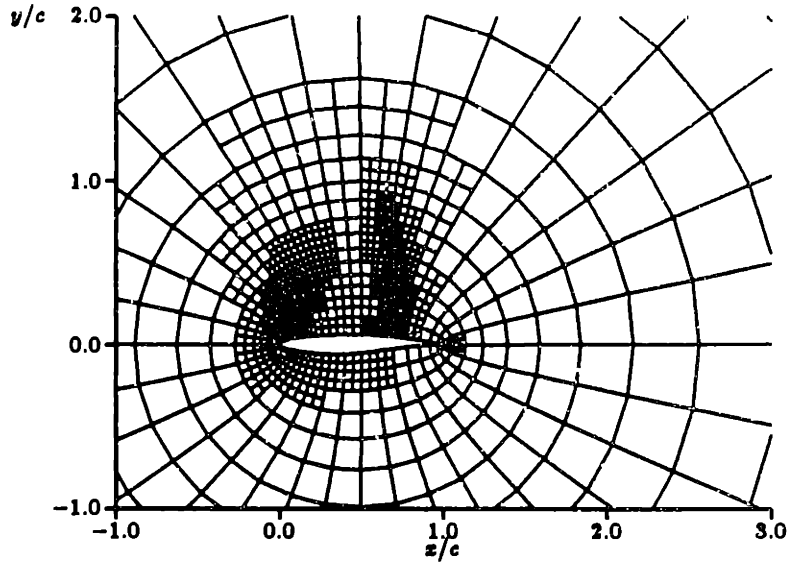


Figure 8.55: Final computational grid for $\mathcal{R}_d = 1.5$, test case AGARD-06.

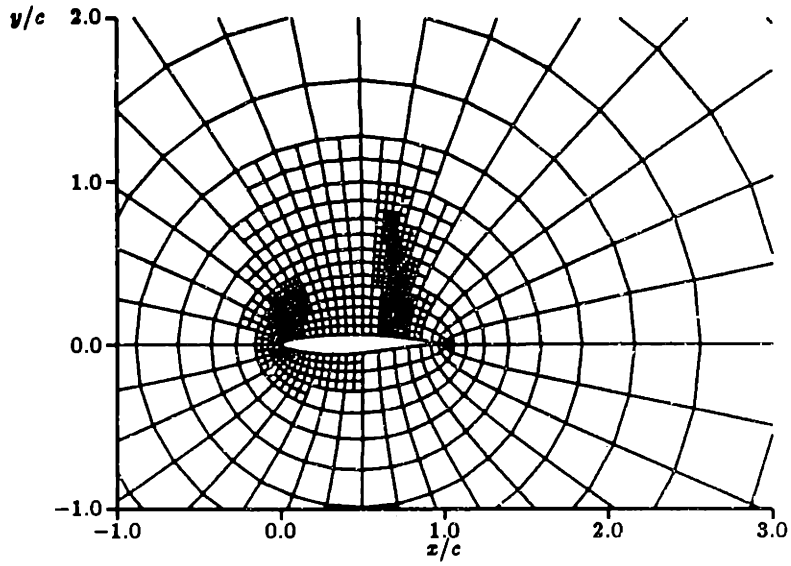


Figure 8.56: Final computational grid for $\mathcal{R}_d = 2.0$, test case AGARD-06.

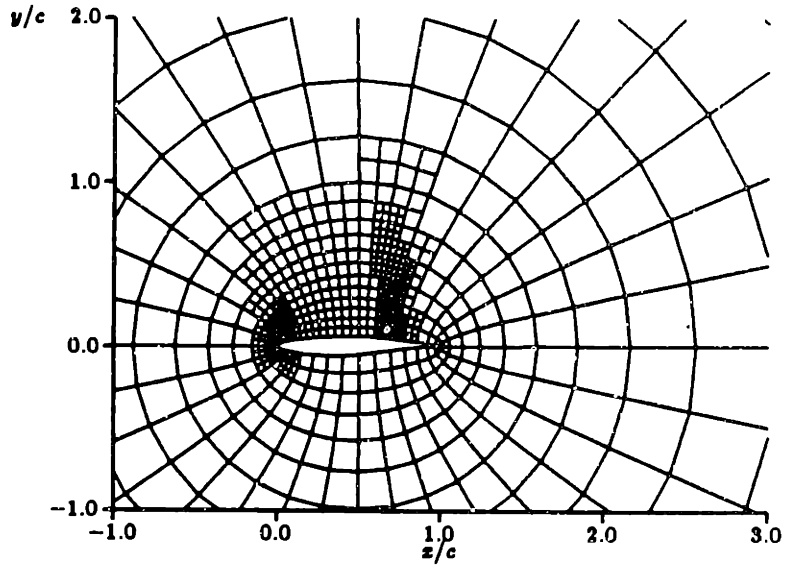


Figure 8.57: Final computational grid for $\mathcal{R}_d = 2.5$, test case AGARD-06.

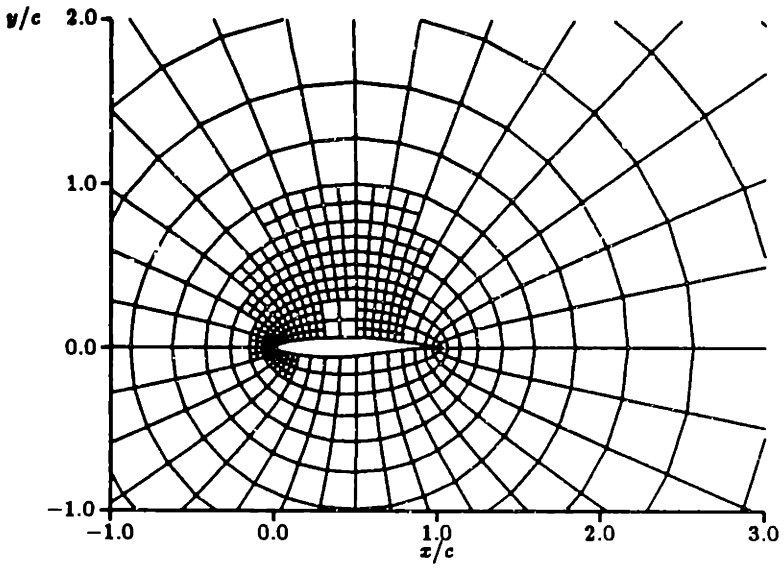


Figure 8.58: Final computational grid for $\mathcal{R}_d = 3.0$, test case AGARD-06.

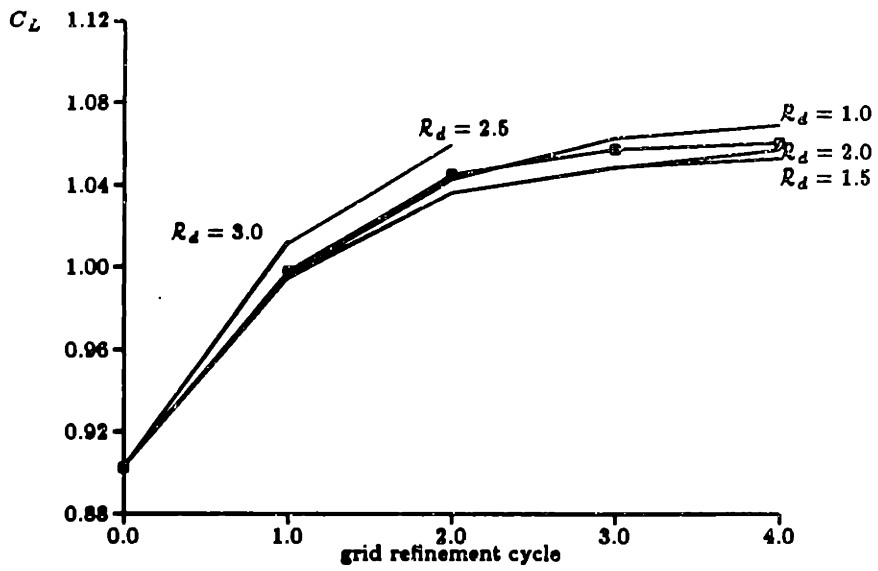


Figure 8.59: Computed lift coefficient versus grid refinement cycle for various division thresholds, $\mathcal{R} = \tilde{\nabla}\rho$, test case AGARD-06. The square symbols represent the solution with automatically computed thresholds.

Figures 8.54 through 8.58 show the grids which result from various threshold settings. Notice that more of the grid is refined for the smaller value of $\mathcal{R}_d$, as expected. One can also note from figures 8.57 and 8.58 that the number of levels of refinement which resulted is significantly less than for the lower thresholds. This is because the integration scheme diverged, due principally to the proximity of the edge of the embedded region to the upper surface shock, which in turn caused the interface error to grow unbounded.

The computed lift evolutions for these cases is plotted in figure 8.59 along with the evolution of the lift coefficient for the automatically chosen threshold case (as noted by the symbols). The predicted values for this latter calculation lies between the solutions computed with fixed thresholds. Also notice that for the three cases which converged at cycle 4, there is no

$\mathcal{R}_d$	number of adaptation cycles	final number of nodes	C_L	C_D	total number of iterations	total pseudo-CPU time
automatically chosen	4	3225	1.0602	0.0394	458	3352
1.0	4	6470	1.0685	0.0399	496	6430
1.5	4	3024	1.0526	0.0389	456	3204
2.0	4	2200	1.0570	0.0397	466	2721
2.5	2+	1027	1.0595	0.0400	319	1203+
3.0	1+	747	1.0126	0.0373	225	835+

Table 8.5: Summary of solutions to AGARD-06 computed with various threshold values

discernible relationship between the final lift coefficient and the threshold level.

Table 8.5 contains a summary of this data. Notice that the calculation with the automatically chosen threshold yields its solution in approximately the same time as the $\mathcal{R}_d = 1.50$ calculation. This is to be expected since the average threshold computed for that case was $\mathcal{R}_d = 1.45$. Furthermore, a value of $\mathcal{R} \approx 1.50$ seems to be a reasonable choice for both accuracy and efficiency. Therefore, the automatic threshold selection algorithm is expected to produce good results for flows which are similar to that of AGARD-06.

8.6 Application of MITOSIS to Complex Flow Topologies

The MITOSIS grid adaptation program with the knowledge base described in section 8.2 has been applied to the computations of flows over a NACA-0012 airfoil at various free-stream conditions. The cases, which

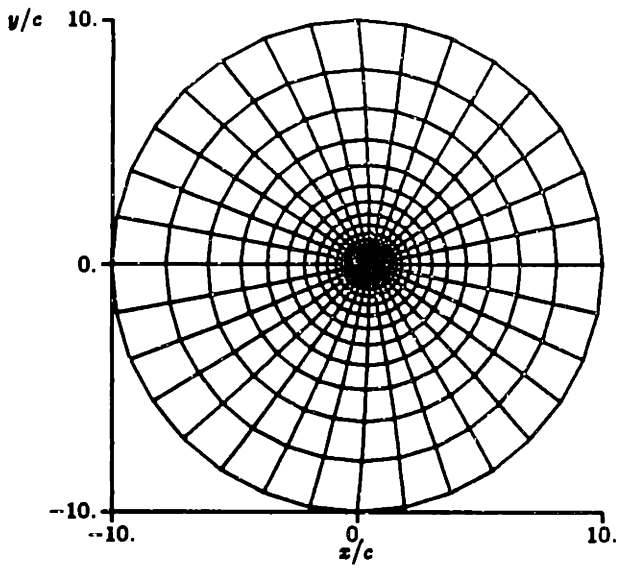


Figure 8.60: Global computational grid (grid 0) for test cases AGARD-01 through AGARD-05.

correspond to test cases 1 through 5 of the AGARD test cases[3], serve to demonstrate the truly adaptive nature of the MITOSIS scheme.

For each case, the computation begins with the global grid shown in figure 8.60, which is a 32×16 O-type mesh. The airfoil node distribution is defined such that the nodes are clustered at the leading and trailing edges; the far-field boundary nodes are evenly spaced along a circle with a ten chord radius which is centered at the airfoil leading edge. In each computation, refinement singularities are defined at the leading and trailing edge nodes so as to enforce refinement there.

In the following sections, the evolution of the adapted grid and its corresponding solution for each of the test cases is given in graphical form. The final solution is then compared with the "best" AGARD solution. Care should be taken with this comparison however, since the "best" is not nec-

essarily correct, and furthermore does not necessarily represent the typical solution of those received; nevertheless the comparison does provide a relevant reference.

8.6.1 AGARD-01

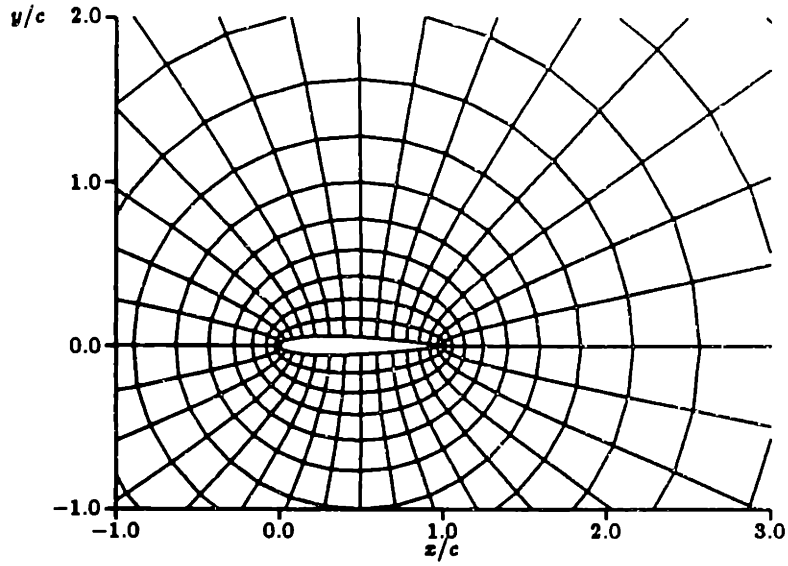
The first test case, AGARD-01, has a free-stream Mach number $M_\infty = 0.80$ and an angle of attack $\alpha_\infty = 1.25°$. The topology of the resulting flow-field is very similar to that of case AGARD-06.

The evolving grids and solutions are shown in figures 8.61 through 8.64, where part a shows the computational grid in the vicinity of the airfoil and part b shows the current and final surface Mach number distributions. Notice that as the grid is successively refined in the vicinity of the shock, the width of the computed shock decreases as expected.

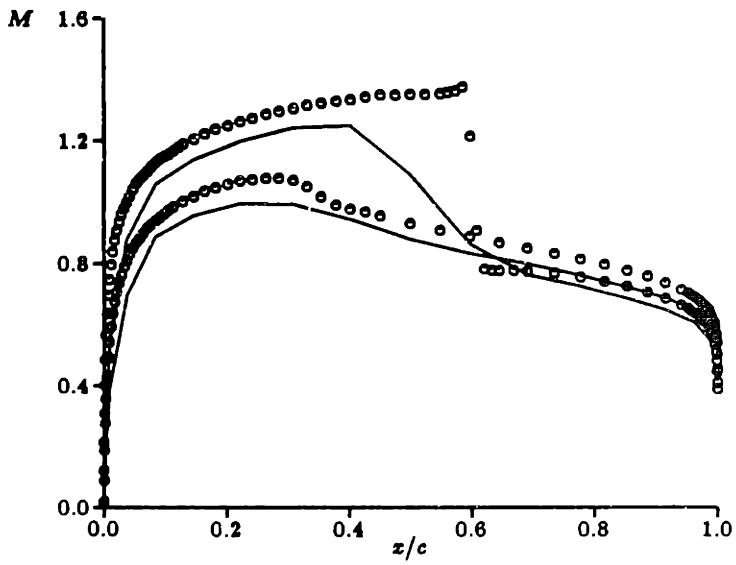
The final surface Mach number distribution is compared in figure 8.65 with the "best" AGARD solution[2]. There are two discrepancies which are immediately obvious from the figures. First, the upper surface shock position computed by the present scheme is about 3.7%-chord ahead of the shock in the published solution. While this is somewhat disturbing, it should be noted that the solutions which AGARD received for this case had an approximately 3%-chord scatter in the predicted shock positions. Second, the present scheme does not predict a lower surface shock, unlike the plotted published solution. The presence of such a shock is somewhat in doubt however based upon the fact that of the seven solutions that AGARD received, more than half also did not predict such a shock.

The computed Mach number contours are shown in figure 8.66. As in section 8.3, one can see that the contours pass smoothly through the edge of the embedded regions. This figure also serves to demonstrate the crispness of the computed shock.

The convergence histories for this case are shown in figure 8.67, where the

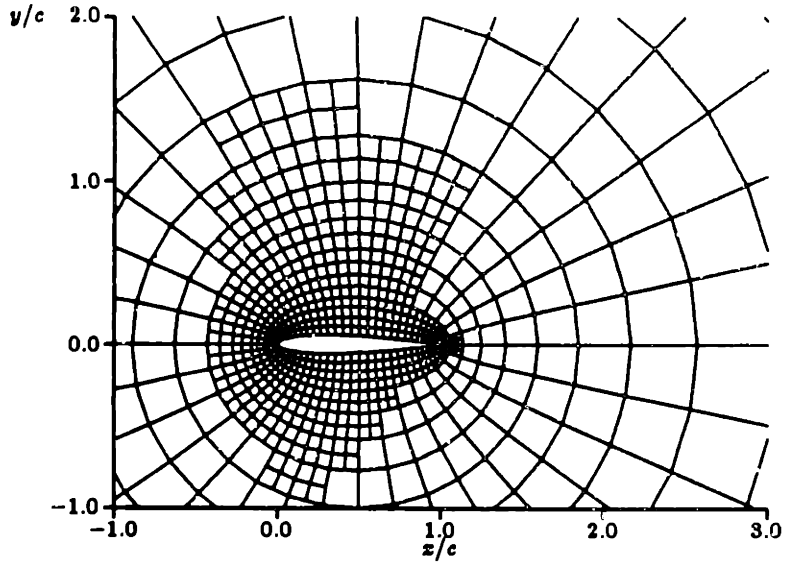


a: computational grid near airfoil

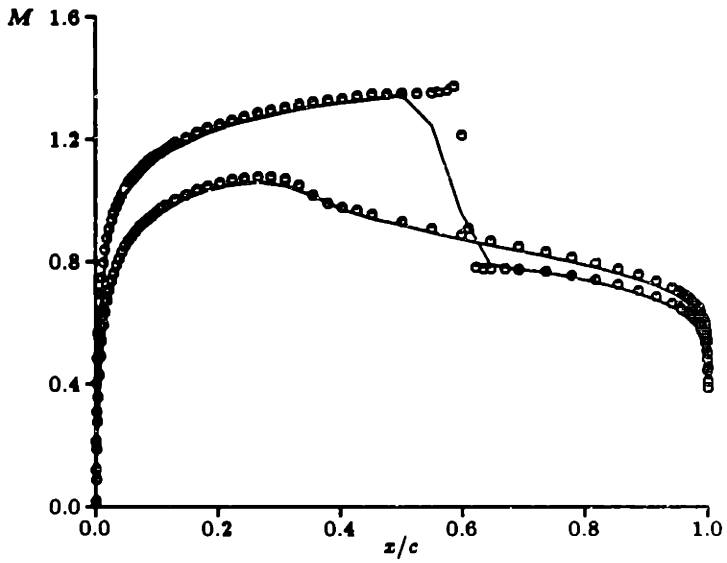


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.61: Grid 0 solution for AGARD-01

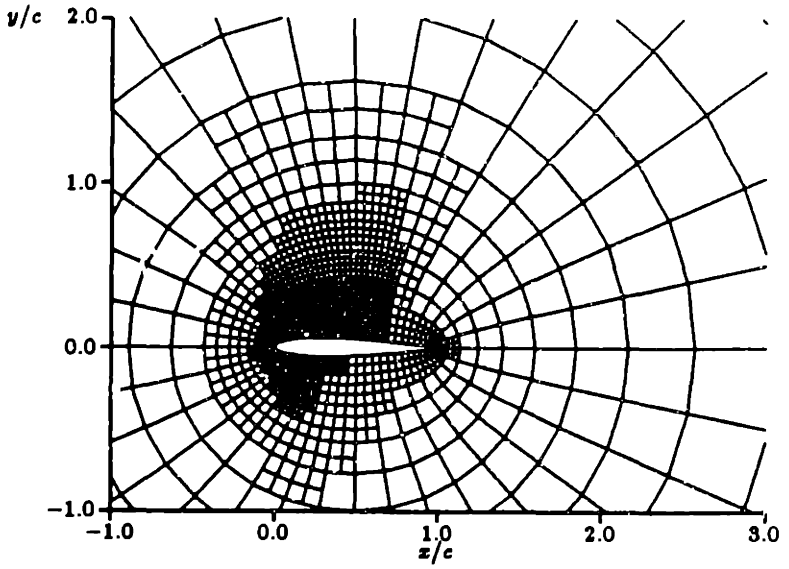


a: computational grid near airfoil

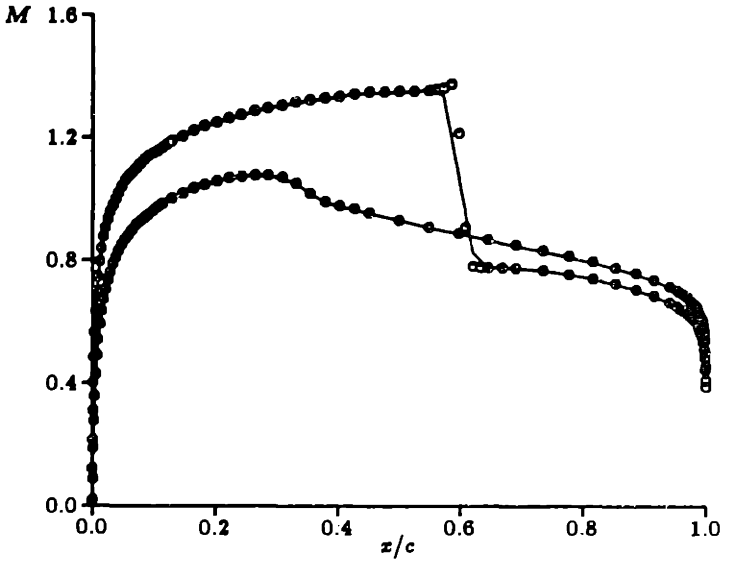


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.62: Grid 1 solution for AGARD-01

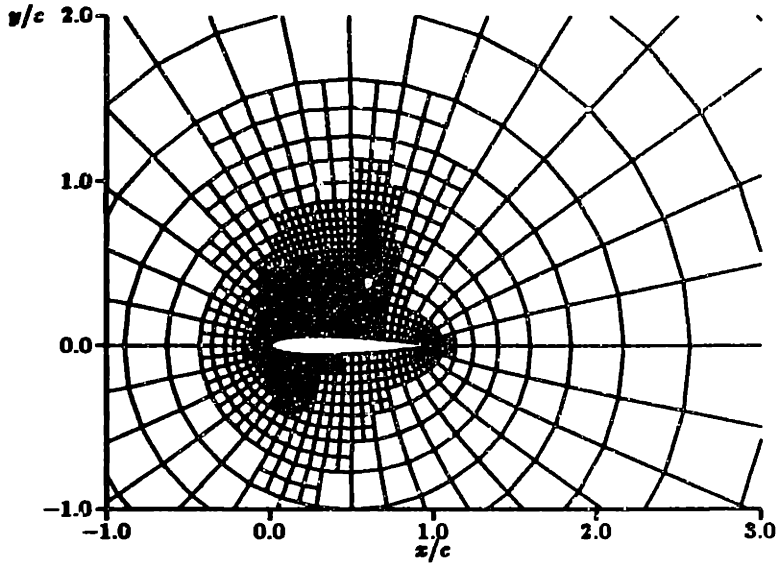


a: computational grid near airfoil

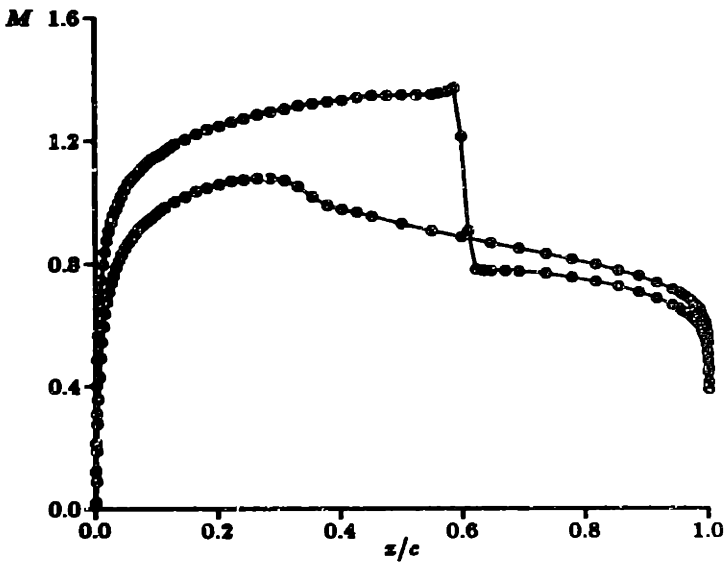


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.63: Grid 2 solution for AGARD-01



a: computational grid near airfoil



b: surface Mach number distribution.
Line and symbols are current (final) solution.

Figure 8.64: Grid 3 (final) solution for AGARD-01

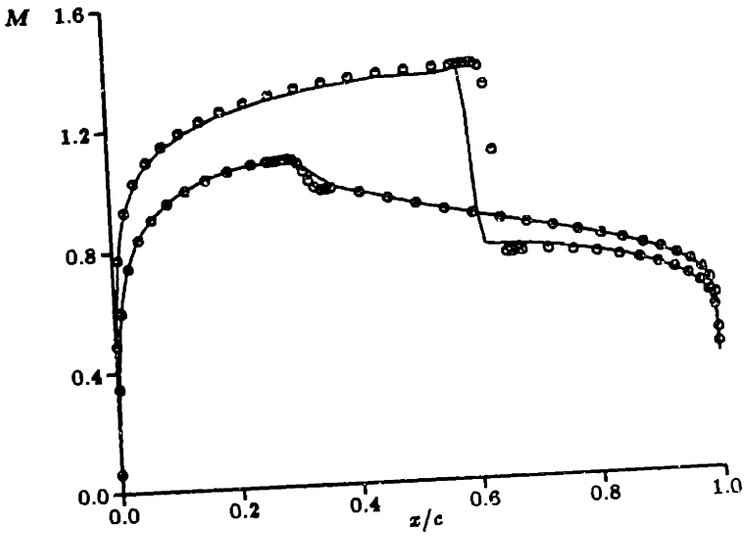


Figure 8.65: Surface Mach number distributions for case AGARD-01. Line is present solution, symbols are reference solution.

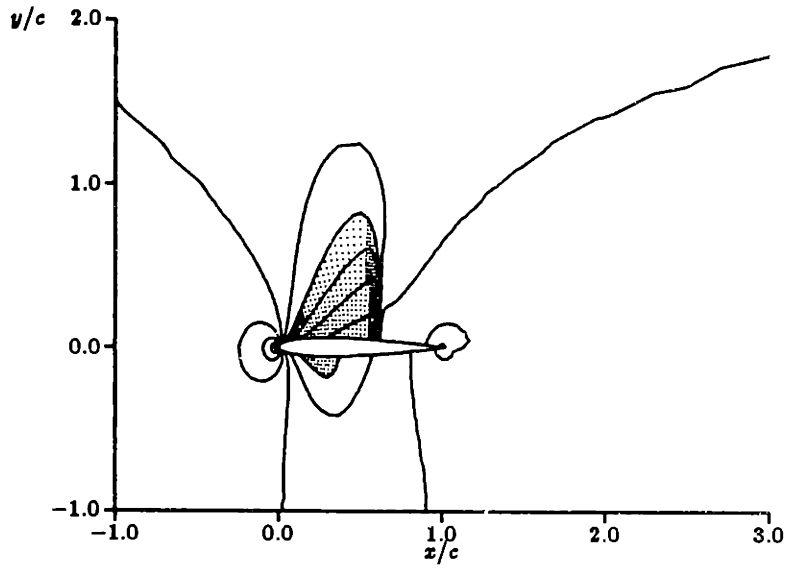
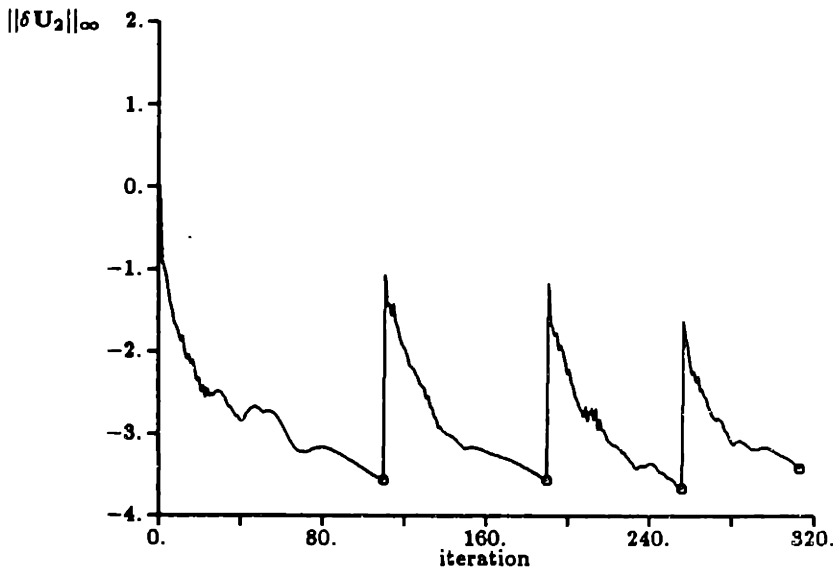
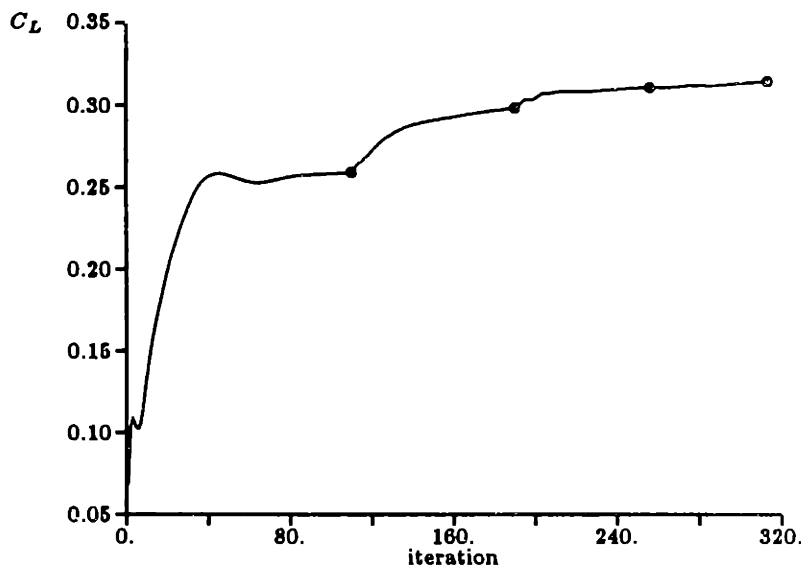


Figure 8.66: Mach number contours for AGARD-01, $\Delta M = 0.10$. Supersonic nodes are dotted.



a: L_∞ norm of $\delta \mathbf{U}_2$.



b: Computed lift distribution.

Figure 8.67: Convergence histories for AGARD-01

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
0	544	672	0.2594	0.0247	110	299
1	1022	1256	0.2991	0.0205	190	708
2	1941	2396	0.3115	0.0203	256	1349
3	3157	3892	0.3152	0.0203	313	2248
average AGARD value			0.3587	0.0228		
scatter			7.6%	9.9%		

Table 8.6: Summary of solution evolution for AGARD-01.

effect of the adaptation can be seen. As noted in section 8.3, the convergence rate of the scheme seems to be independent of the number of embedded regions which are present.

The solution evolution is summarized in Table 8.6. Because of the error in the location of the upper surface shock, the computed lift coefficient is about 12% low as compared with the published AGARD result. A globally refined solution for this case resulted in grid 2 force coefficients of $C_L = 0.3119$ and $C_D = 0.0202$ which are in excellent agreement with the adaptively refined solution. Hence it does not seem that the adaptation is responsible for the lift discrepancy in this case either.

As a final note for this case, it must be pointed out that the adaptation strategy (knowledge base) developed for case AGARD-06 was applied without difficulty.

8.6.2 AGARD-02

The second NACA-0012 test case, AGARD-02, is for a free-stream Mach number $M_\infty = 0.85$ and an angle of attack $\alpha_\infty = 1.0°$. The flow-field topology is somewhat different from the above two cases in that strong shocks are present on both upper and lower surfaces.

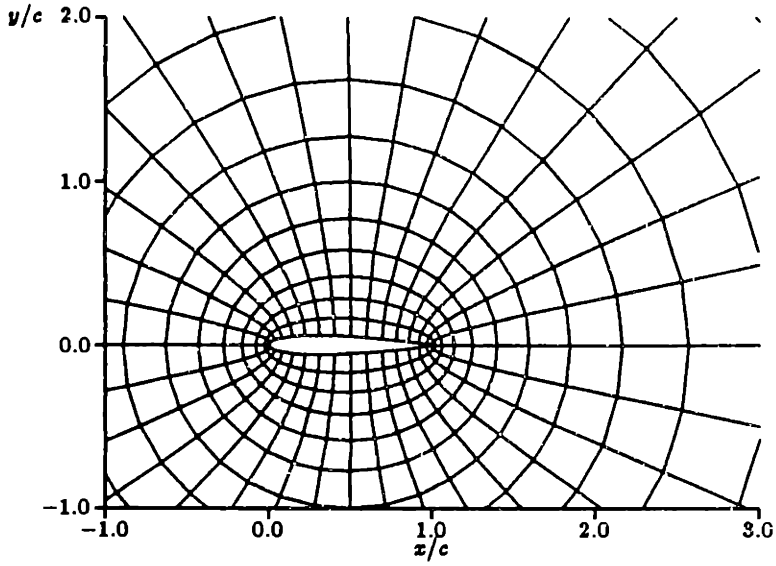
The evolving grids and solutions for the first three adaptation cycles are shown in figures 8.68 through 8.71. Again, as the grid is successively refined in the vicinity of the shocks, the width of the computed shock decreases as expected.

Unfortunately, the flow integration on the fourth adaptive grid (figure 8.72) diverged. Careful examination of the convergence histories and flow-field contour plots indicated that the problem was related to the closeness of the edge of the newly embedded region to the shock location. (Recall from chapter 4 that smoothing, which is large in the vicinity of the shocks, creates interface errors which can eventually result in divergence.)

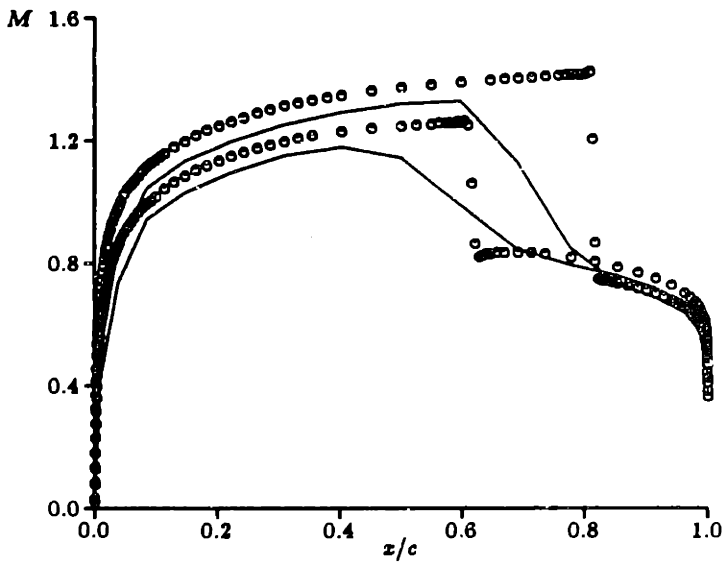
Because of the above, the adaptation strategy described in section 8.1 has to be modified to handle this case. The appropriate change involves adding a buffer zone around adapted regions if divergence is detected.

Implementation of the change is made extremely simple by the hybrid system approach. Rules can be added to the system which state exactly the change identified in the previous paragraph.

The rule which was added is:

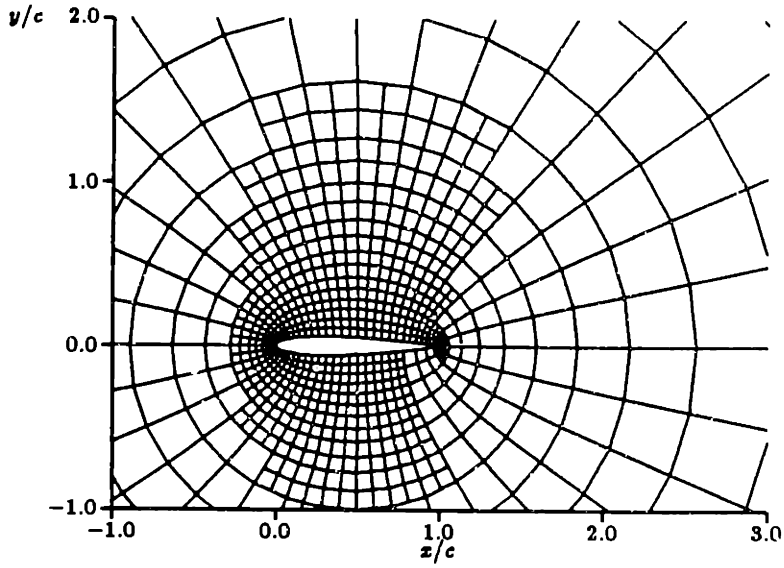


a: computational grid near airfoil

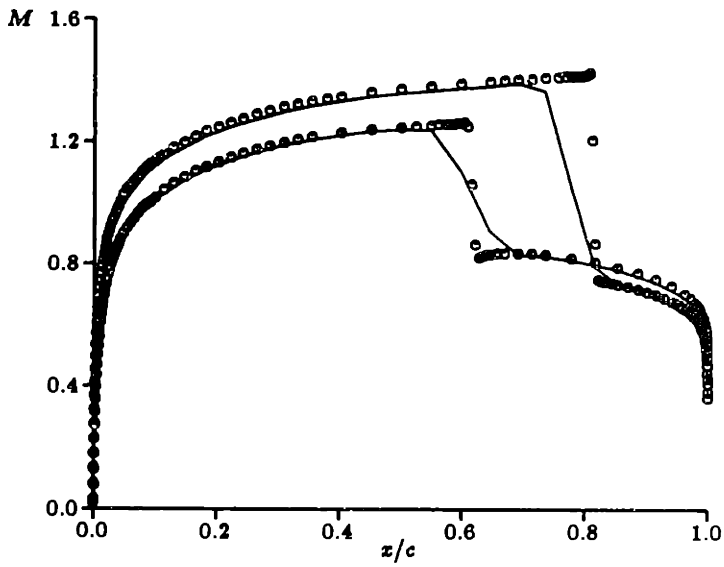


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.68: Grid 0 solution for AGARD-02

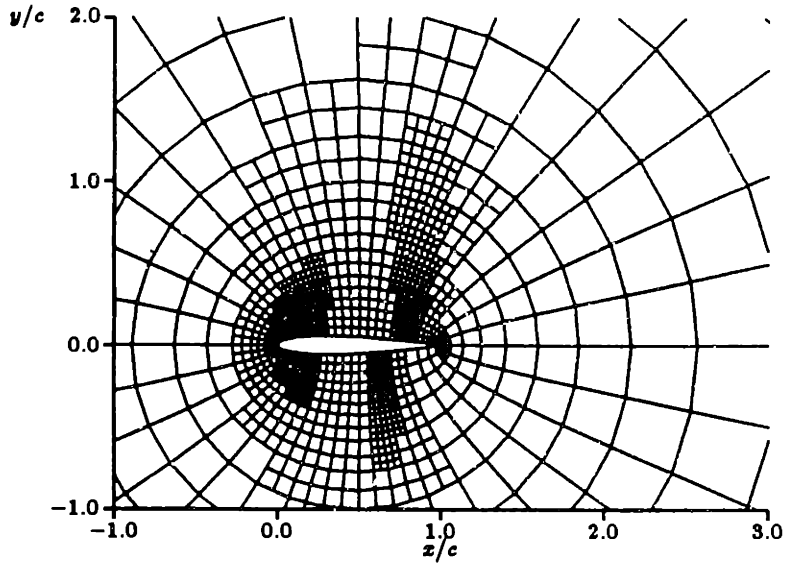


a: computational grid near airfoil

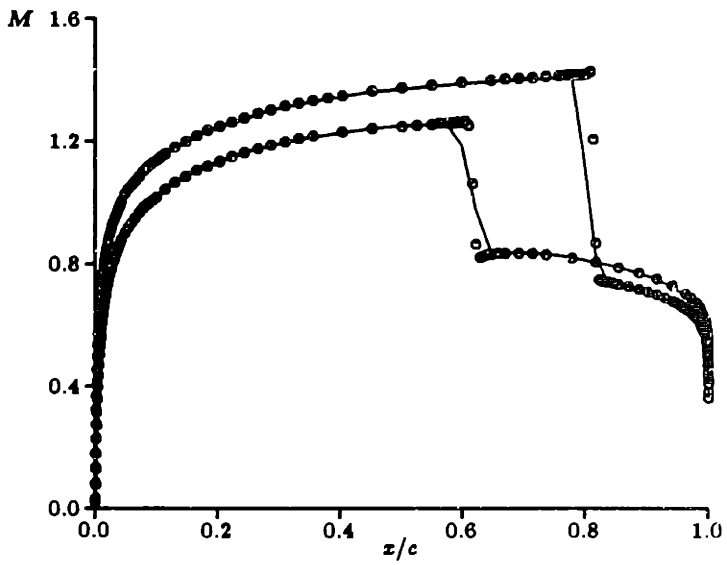


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.69: Grid 1 solution for AGARD-02

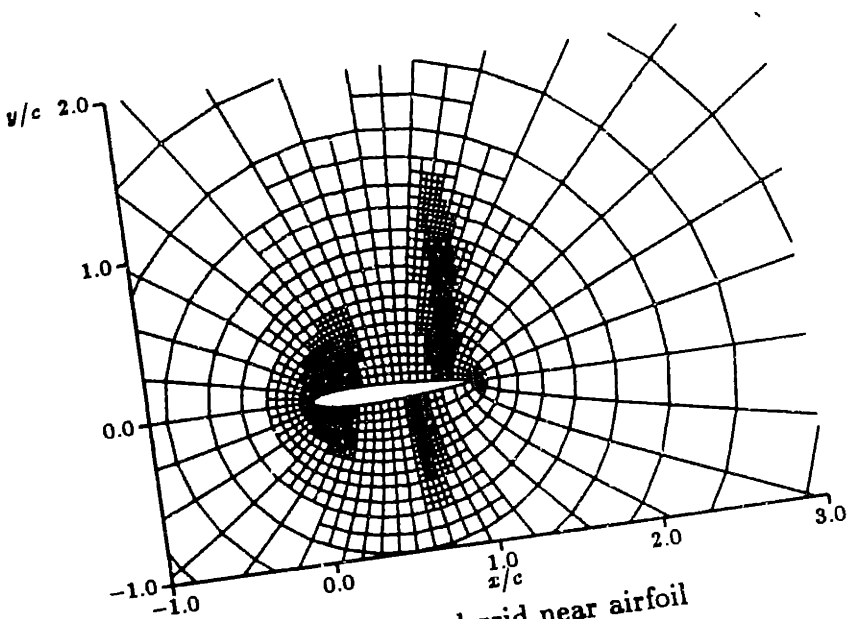


a: computational grid near airfoil

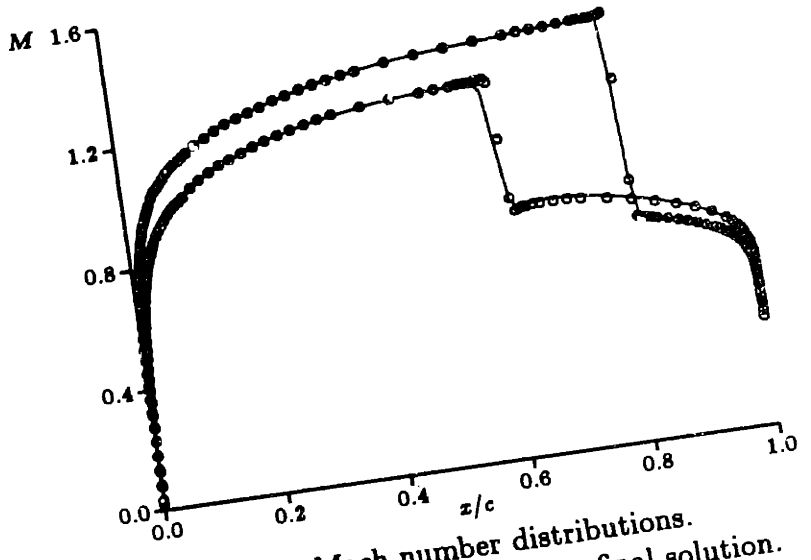


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.70: Grid 2 solution for AGARD-02



a: computational grid near airfoil



b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.71: Grid 3 solution for AGARD-02

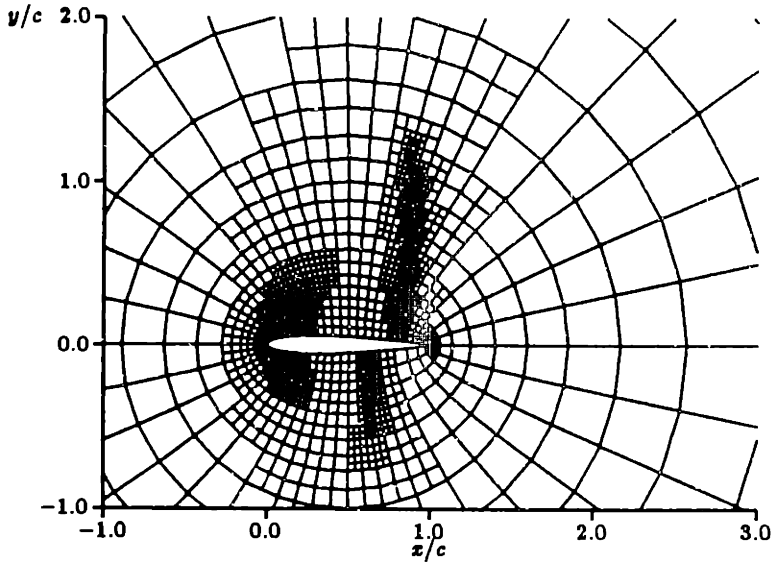


Figure 8.72: Grid 4 for which integration scheme failed, test case AGARD-02

```
RULE      DIVERGENCE RECOVERY
CONTEXT   diverged
IF        * NOP
THEN      E2HSPR
ANDTHEN   E2TIME printer
ANDTHEN   G2NBOR nbor_in
ANDTHEN   G2READ out_file printer
ANDTHEN   ADD %cycle cycle -1.0
ANDTHEN   ADD %nbuffer nbuffer +1.0
ANDTHEN   CONSIDER adapt_i
```

which simply states that if the context is `diverged`, then the appropriate action is to read in the previous converged solution (G2READ), decrement the adaptation cycle number `cycle` so that the iterations which diverged do not count as a valid cycle, and increase the buffer size parameter `nbuffer`. The rule concludes by switching the context to `adapt_i`, which then performs an new adaptation with a large buffer size.

In order to ensure that the buffer size remains bounded, another rule is added:

```
RULE      MAXIMUM BUFFER SIZE REACHED
CONTEXT   diverged
IF        nbuffer .GT. max_buffer
THEN      CONSIDER return
```

which causes the program to stop (set context to `return`) when the buffer size has exceeded a limit; `max_buffer` is set to +2.0 in the current version of the knowledge base.

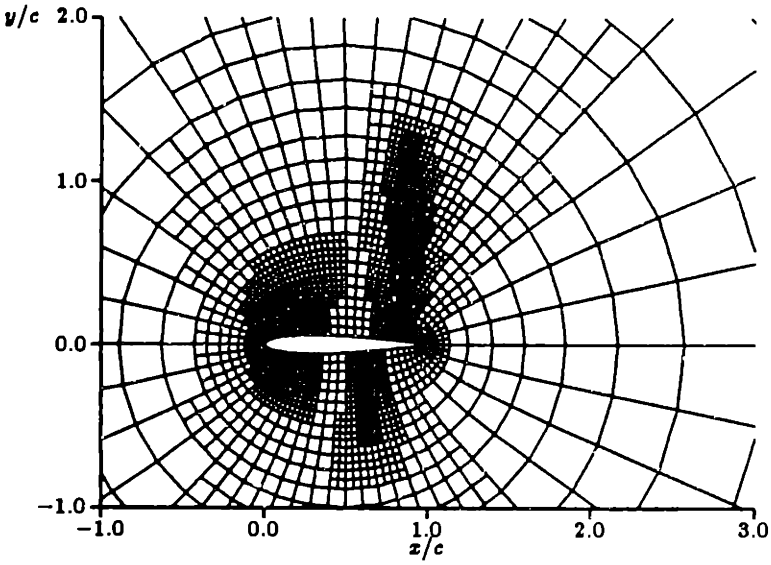
In addition, it is appropriate to add another rule to protect against the case of divergence on the initial global grid. In this case, there would not be a converged solution to read in and hence the program should stop. The rule which accomplishes this is:

```
RULE      INITIAL GRID DIVERGENCE
CONTEXT   diverged
IF        cycle .LE. +0.0
THEN      CONSIDER return
```

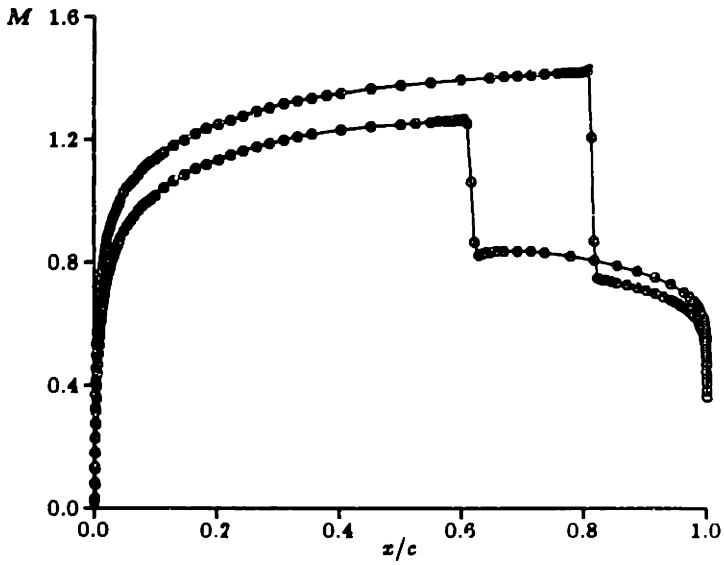
Because the order in which the rules are defined in an expert system is unimportant, these three rules simply were added to the end of the knowledge base.

With these three rules added to the knowledge base, this test case was recomputed, resulting in exactly the same solutions for the first four grids (grid 0 through grid 3) as before, as expected. Again, adaptation for grid 4 yielded the grid shown in figure 8.72, on which the integrator diverged. The program then automatically invoked the appropriate new rule, yielding the grid and subsequently the converged solution shown in figure 8.73.

The final surface Mach number distribution is compared in figure 8.74 with the solution published by AGARD[2]. Again the shock are predicted to be forward of the published results. Figure 8.75 shows the computed Mach



a: computational grid near airfoil



b: surface Mach number distributions.
Line and symbols are current (final) solution.

Figure 8.73: Grid 4 with buffer zone added, test case AGARD-02

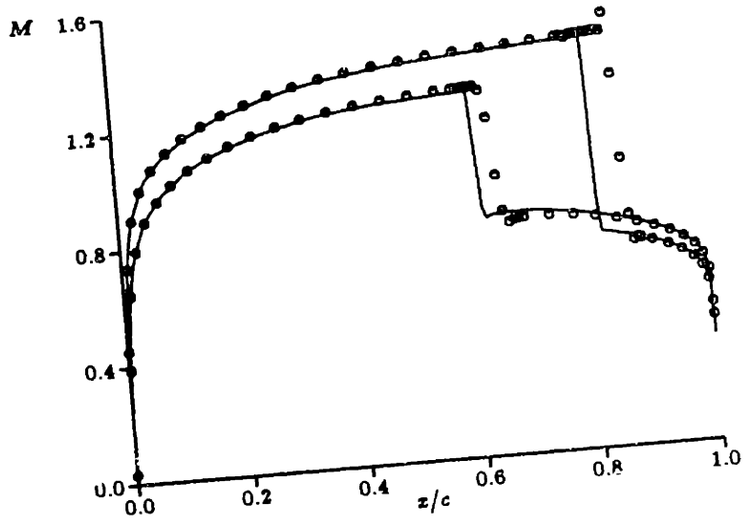


Figure 8.74: Surface Mach number distributions for case AGARD-02. Line is present solution, symbols are reference solution.

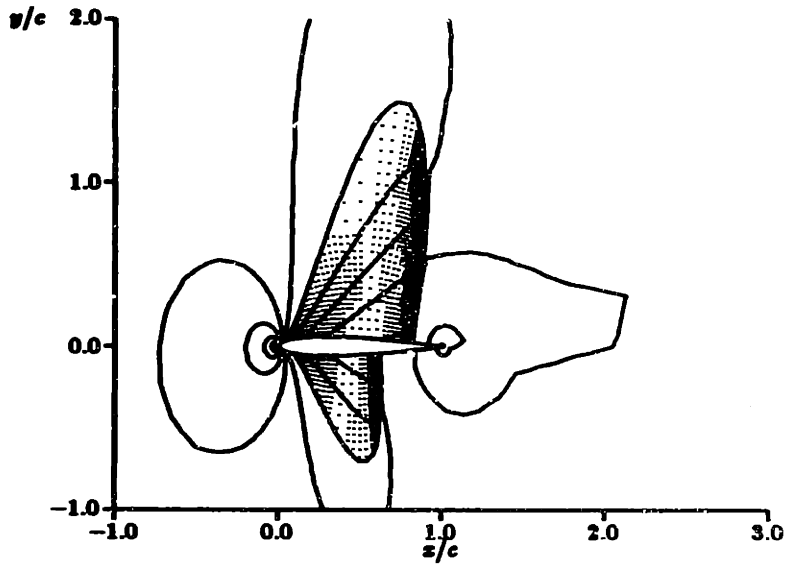


Figure 8.75: Mach number contours for AGARD-02, $\Delta M = 0.10$. Supersonic nodes are dotted.

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
0	544	672	0.2324	0.0474	118	321
1	1037	1276	0.2780	0.0471	207	782
2	1822	2216	0.3072	0.0486	284	1484
3	2968	3600	0.3224	0.0494	348	2434
4	5630	6896	diverged		435	4883
4	7746	9672	0.3217	0.0496	505	7594
average AGARD value			0.3532	0.0545		
scatter			16.4%	23.9%		

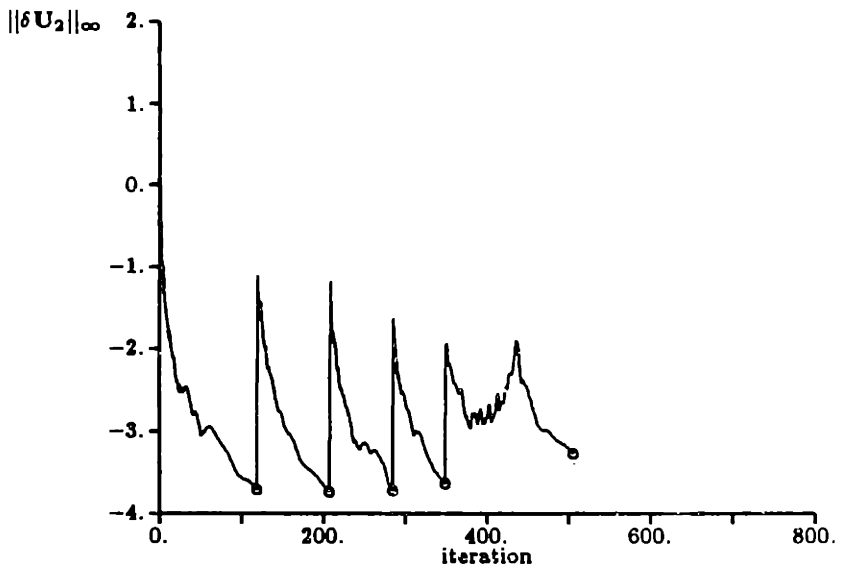
Table 8.7: Summary of solution evolution for AGARD-02.

number contours and the supersonic regions.

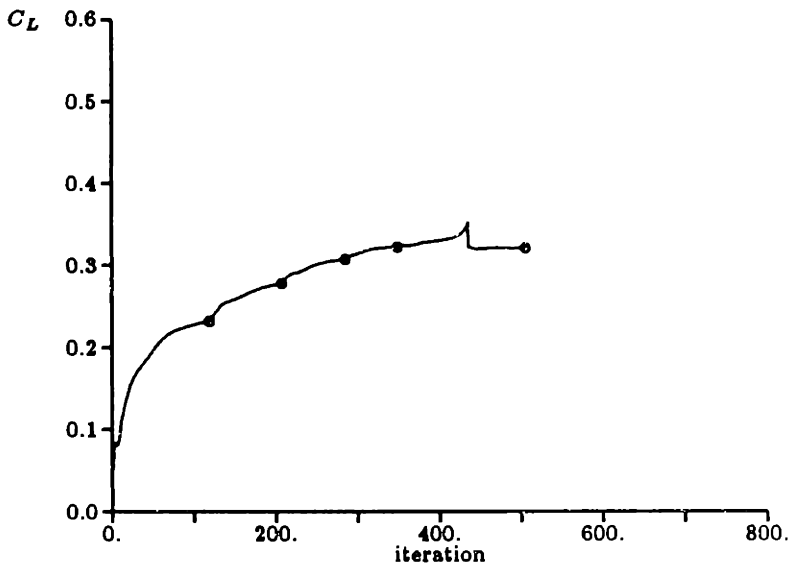
The convergence histories for this case are shown in figure 8.76, where the effect of the adaptation can be seen. Notice the divergence on the original grid 4, which is seen in both plots.

The results of this case are summarized in Table 8.7. The final predicted lift coefficient is within 9.8% of the published values, which is better than in case AGARD-01; unfortunately this better agreement is due principally to the shock position errors on the upper and lower surfaces which tend to cancel each other in the lift coefficient.

In summary for this case, the final computed result was obtained only after modifying the basic grid adaptation strategy described in section 8.1. This modification involved the addition of three new knowledge base rules. Although it is clear that such a modification is possible in typical procedural programs (such as standard FORTRAN), the modification in MITOSIS was



a: L_∞ norm of δU_2.



b: Computed lift distribution.

Figure 8.76: Convergence histories for AGARD-02

greatly facilitated by the simplicity with which rules can be added to the knowledge base.

8.6.3 AGARD-03

Test case AGARD-03 has a very different flow-field topology from any of the above. Here, the free-stream Mach number is $M_\infty = 0.95$ and the angle of attack $\alpha_\infty = 0.0°$, resulting in a fish-tail shock arrangement.

The evolving grids and solutions for the first two adaptation cycles are shown in figures 8.77 through 8.79. Using the knowledge base described above, the current solution is said to be grid-converged since the lift and drag coefficients are the same (within the specified tolerance) after the first two adaptation cycles. This is to be expected for a non-lifting section, and hence using lift and drag coefficients to determine final convergence is inappropriate.

To circumvent the problem, it was decided to ensure that non-lifting solutions employ at least three adaptation cycles. Again, new rules can be added to the knowledge base to reflect this modification of the strategy.

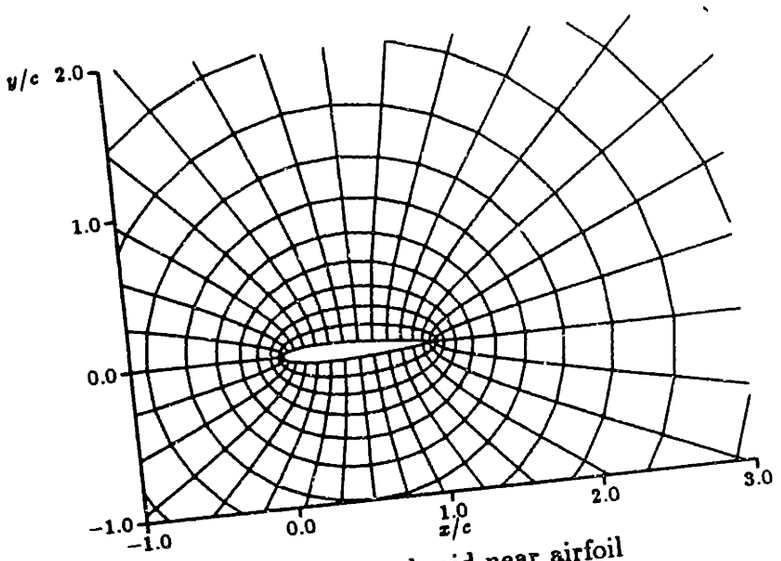
The first rules to be added involve determining if the problem is a lifting or non-lifting type. They take the form:

```
RULE      NON-LIFTING PROBLEM
CONTEXT   adapt_i
IF        clift .LE. clift_tol
ANDIF     prob_type .EQ. +0.0
THEN      SET %prob_type +1.0
```

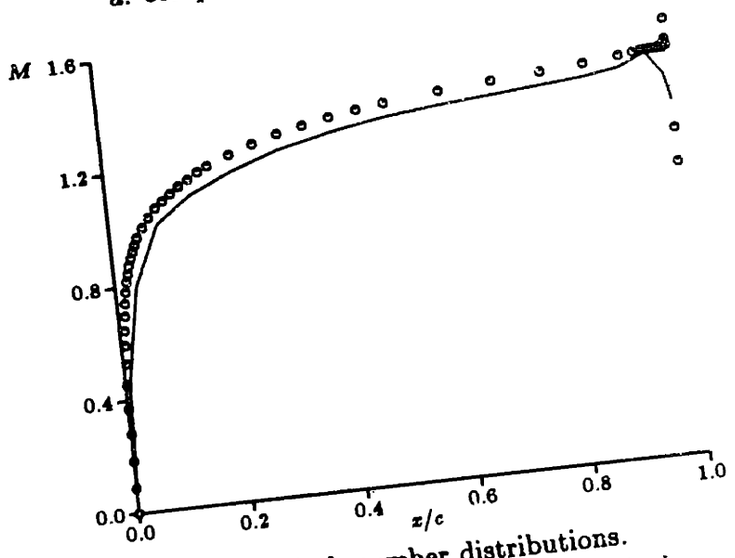
and:

```
RULE      LIFTING PROBLEM
CONTEXT   adapt_i
IF        clift .GT. clift_tol
ANDIF     prob_type .EQ. +1.0
THEN      SET %prob_type +0.0
```

In each case, prob_type is toggled if its current setting and the current value of the lift coefficient are inconsistent.

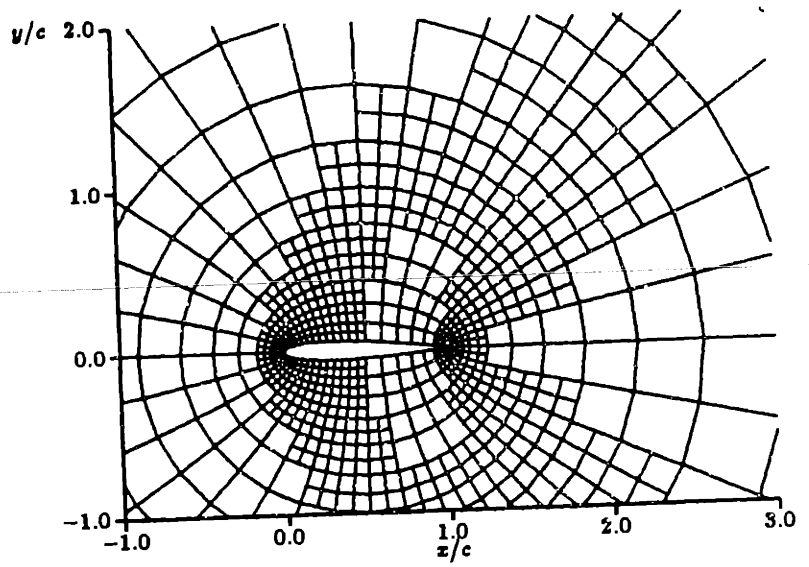


a: computational grid near airfoil

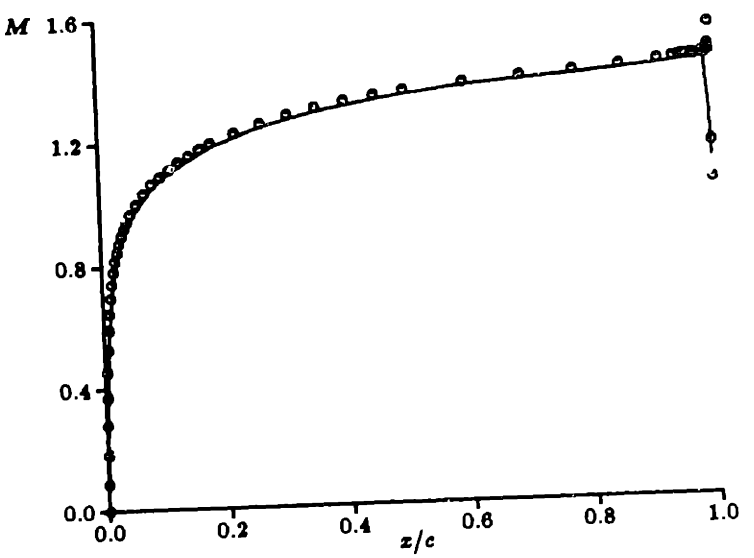


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.77: Grid 0 solution for AGARD-03

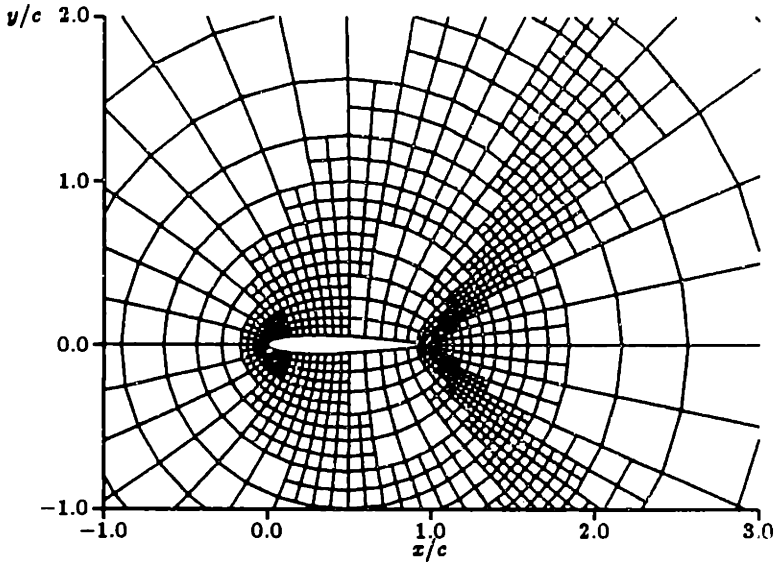


a: computational grid near airfoil

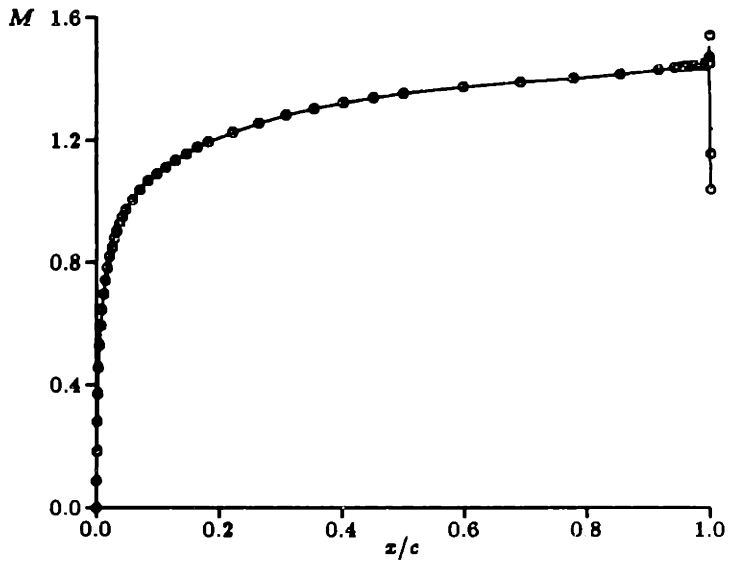


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.78: Grid 1 solution for AGARD-03



a: computational grid near airfoil



b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.79: Grid 2 solution for AGARD-03

The other rule which must be added is the one which requires that the minimum number of adaptation cycles `min_cycle` have been taken, or:

```
RULE      FINAL CONVERGENCE REACHED (NL)
CONTEXT   adapt_i
IF        delta_cl .LE. clift_tol
ANDIF     delta_cd .LE. cdrag_tol
ANDIF     prob_type .EQ. +1.0
ANDIF     cycle .GT. min_cycle
THEN      E2PRNT
ANDTHEN   CONSIDER return
```

The attribute `min_cycle` is initialized to +2.0. If this rule does not fire, then the rule `INITIALIZE ADAPTATION` fires and another grid adaptation cycle ensues.

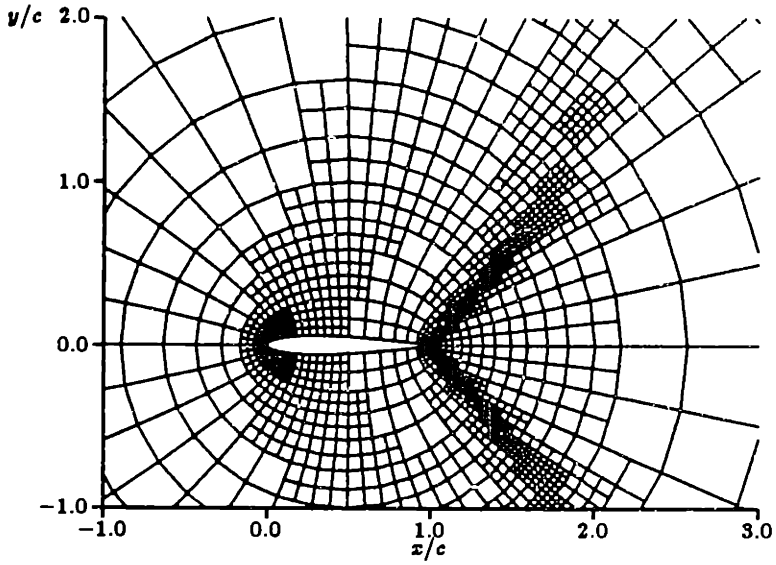
When applied to test case AGARD-03, the current rules detect that the case is non-lifting and require a third adaptation cycle, resulting in the grid and solution shown in figure 8.80. Since this is the third cycle of a non-lifting case, the adaptation process terminates.

Again the final surface Mach number distribution is compared in figure 8.81 with the solution published by AGARD[2]; the Mach number contours are shown in figure 8.82. Notice here how the adapted region follows the two shocks which emanate from the trailing edge.

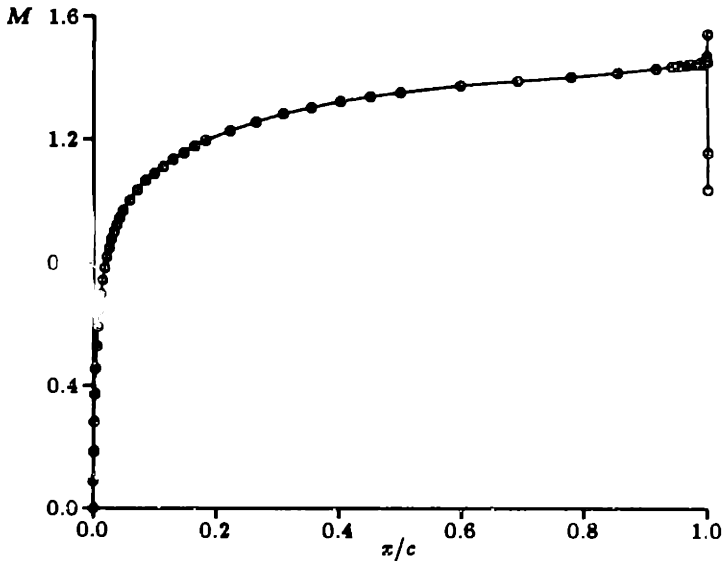
An additional measure of the accuracy of the present solution is obtained by determining the location of the sonic point downstream of the airfoil on the symmetry line. The plot of the Mach number along this downstream stagnation streamline is shown in figure 8.83, from which one can see that the sonic point is predicted to be at $x/c = 2.08$, which is within the band $(1.4 < x/c < 3.1)$ generated by the published solutions.

The convergence histories for this case are shown in figure 8.84, where the effect of the adaptation can be seen.

Table 8.8 contains a summary of the solution evolution for this case.



a: computational grid near airfoil



b: surface Mach number distributions.
Line and symbols are current (final) solution.

Figure 8.80: Grid 3 solution for AGARD-03

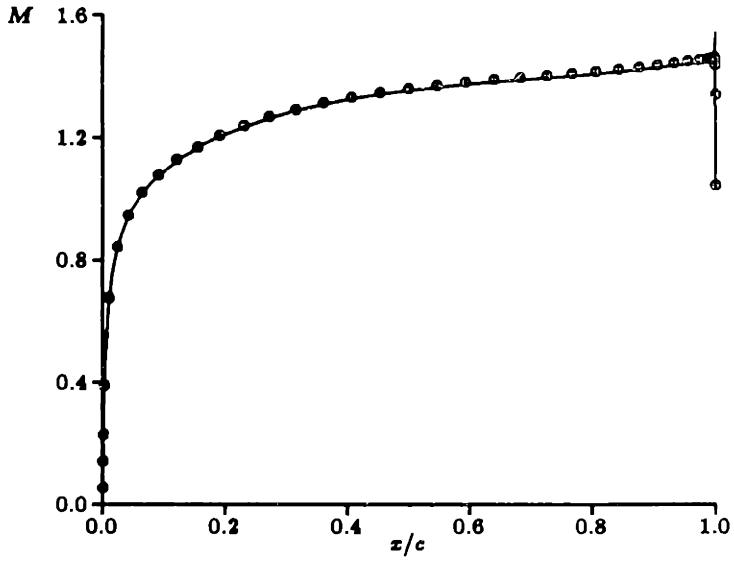


Figure 8.81: Surface Mach number distributions for case AGARD-03. Line is present solution, symbols are reference solution.

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
0	544	672	0.0000	0.1114	110	299
1	1093	1328	0.0000	0.1078	168	616
2	1823	2192	0.0000	0.1076	244	1309
3	3092	3704	-.0001	0.1076	343	2839
average AGARD value			0.0000	0.1082		
scatter				0.7%		

Table 8.8: Summary of solution evolution for AGARD-03.

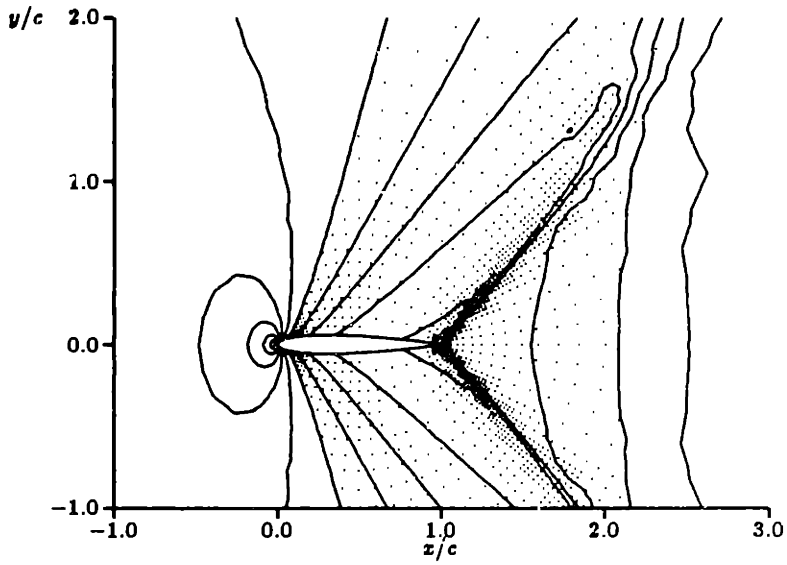


Figure 8.82: Mach number contours for AGARD-03, $\Delta M = 0.10$. Supersonic nodes are dotted.

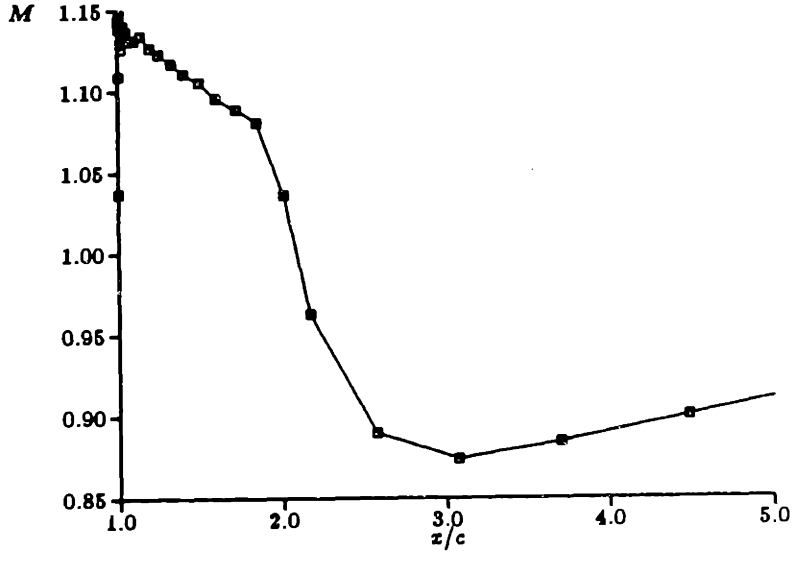
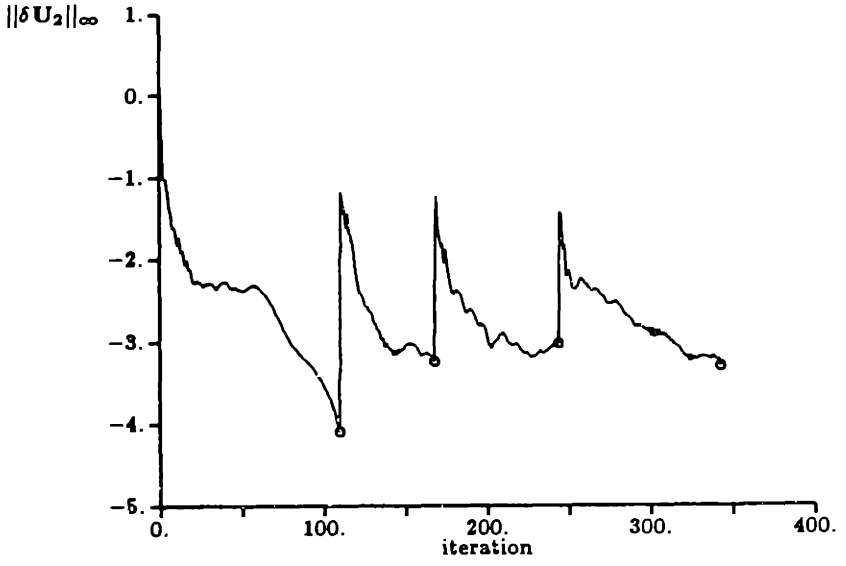
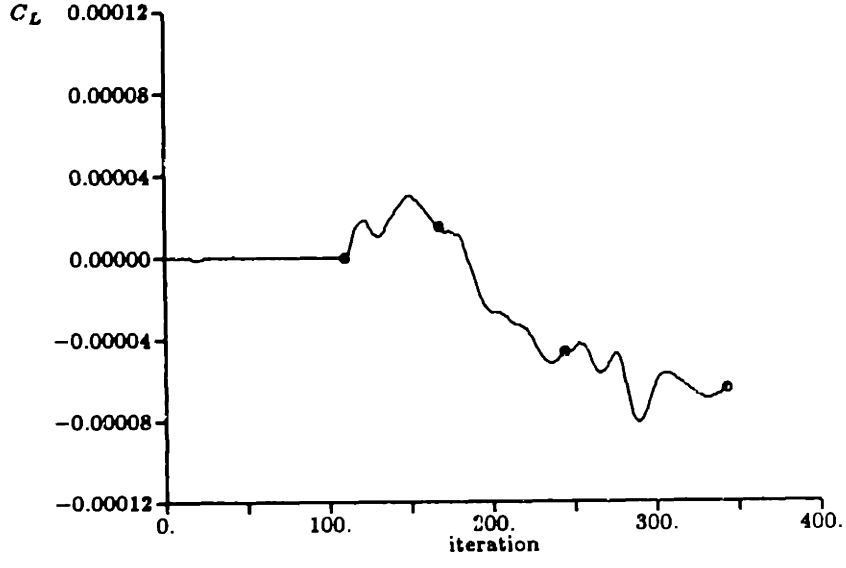


Figure 8.83: Mach number distribution along the upstream stagnation streamline (symmetry line), test case AGARD-03.



a: L_∞ norm of δU_2.



b: Computed lift distribution.

Figure 8.84: Convergence histories for AGARD-03

The predicted lift and drag coefficients are in excellent agreement with the published values. Notice that without grid refinement, the agreement would have been substantially worse.

In summary, again the adaptation strategy had to be modified to properly handle this case. And again, the fact that the adaptation strategy was encoded in a knowledge base greatly simplified the implementation of this modification.

8.6.4 AGARD-04

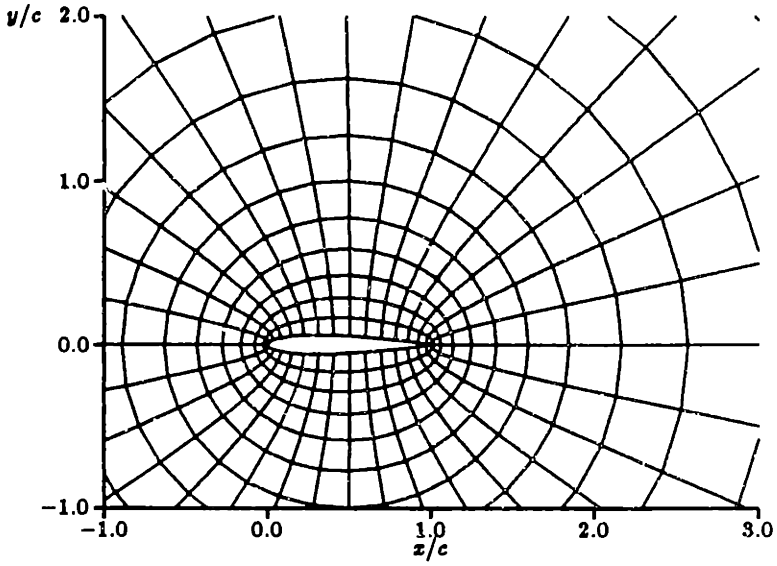
The computation for test case AGARD-04 was uneventful due to the strategy modifications already developed in the previous two sections. The flow topology for this case contains a symmetric bow shock due to the supersonic free-stream conditions $M_\infty = 1.20$ and $\alpha_\infty = 0.0°$.

The evolving grids and solutions for the first two adaptation cycles are shown in figures 8.85 through 8.87. On grid 3, shown in figure 8.88, the integrator again diverged. This automatically invoked the divergence recovery rules, which subsequently resulted in the grid and solution shown in figure 8.89. Again, because this is a non-lifting case, the computation halted after three adaptation cycles.

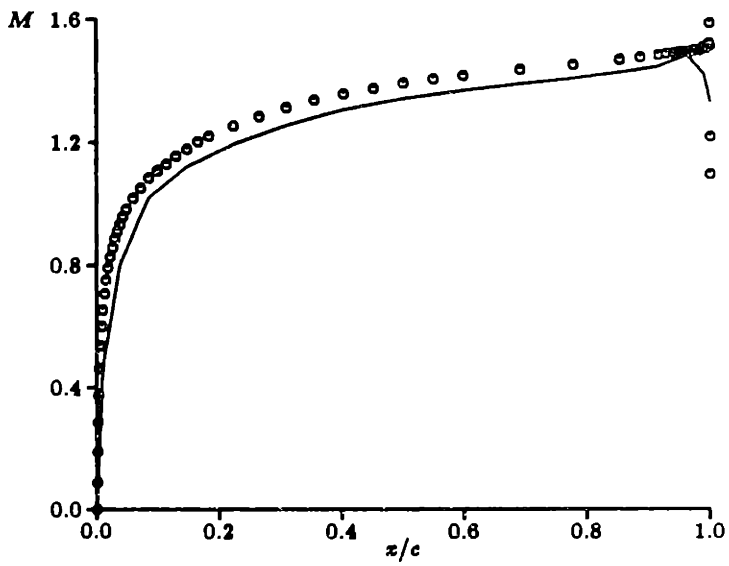
The final surface Mach number distribution is shown in figure 8.90 along with the solution published by AGARD[2]; the Mach number contours are shown in figure 8.91. In the latter, the bow shock is clearly identified. The exact location of the bow shock can be determined by plotting the Mach number distribution along the upstream stagnation streamline, as in figure 8.92. Here one can see that the shock is located at $x/c = -0.48$ which is in excellent agreement with the average published value of $x/c = -0.45$.

The convergence histories for this case are shown in figure 8.93, where the effect of the adaptation can be seen. Again the divergence on the grid 3 is clearly visible in both plots.

A summary is again provided in Table 8.9, which shows the excellent agreement between the current solution and the published solutions.

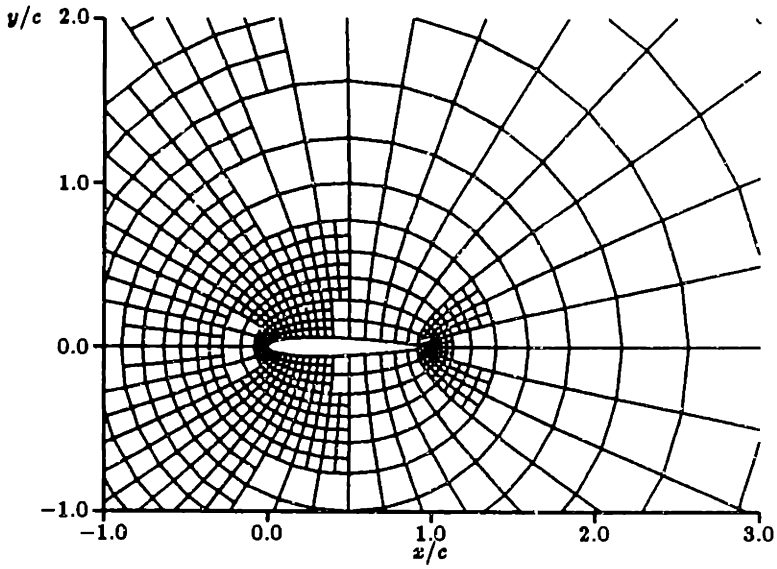


a: computational grid near airfoil

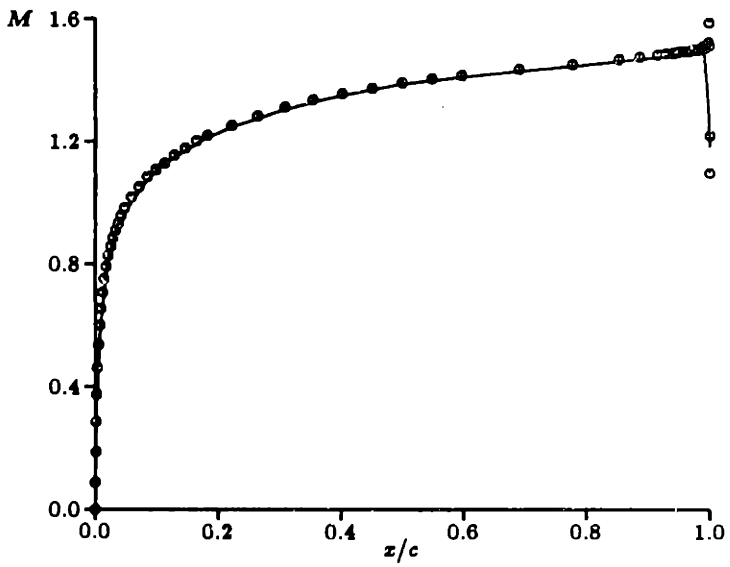


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.85: Grid 0 solution for AGARD-04

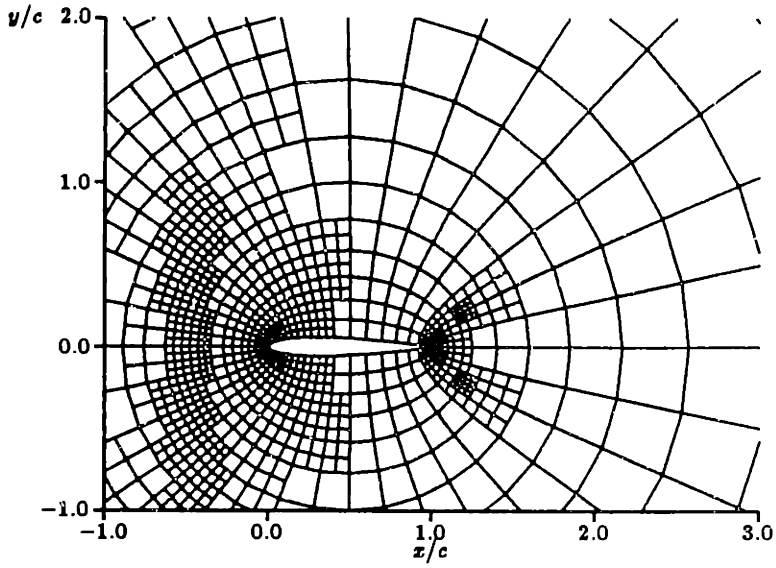


a: computational grid near airfoil

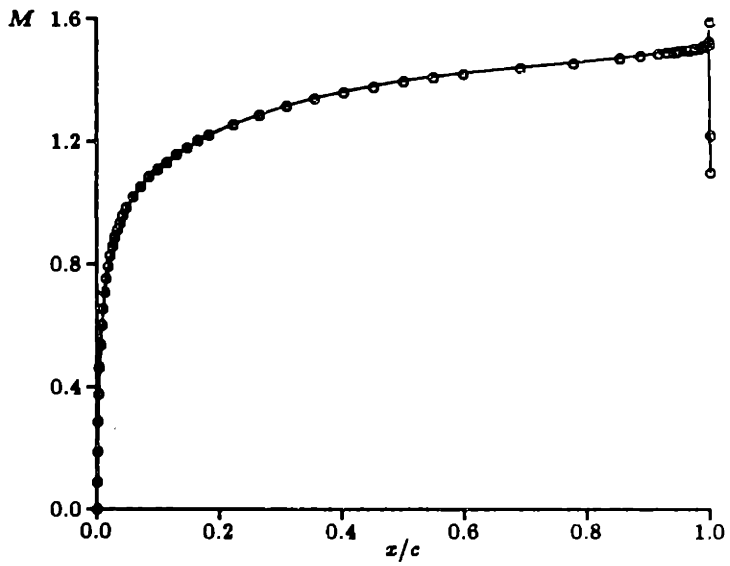


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.86: Grid 1 solution for AGARD-04



a: computational grid near airfoil



b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.87: Grid 2 solution for AGARD-04

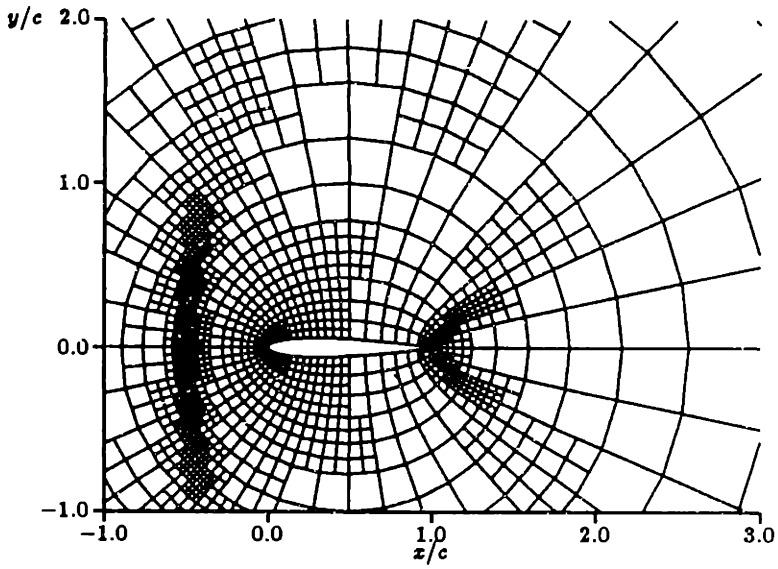
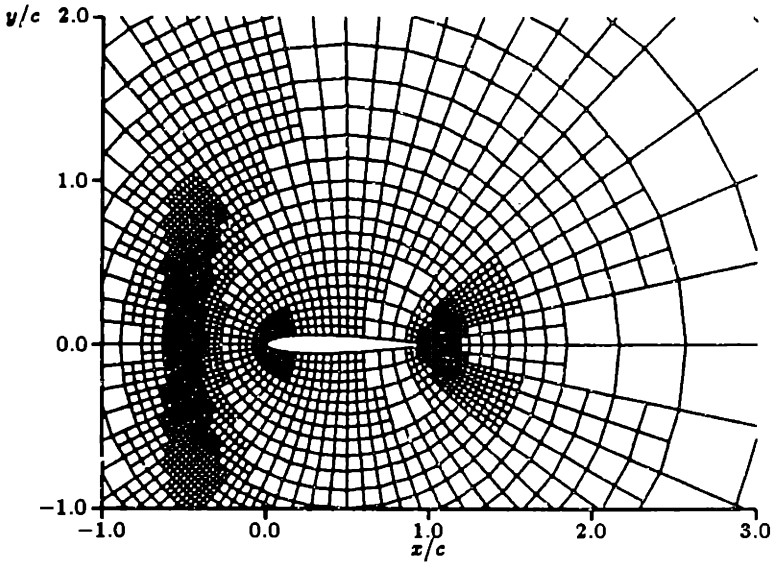
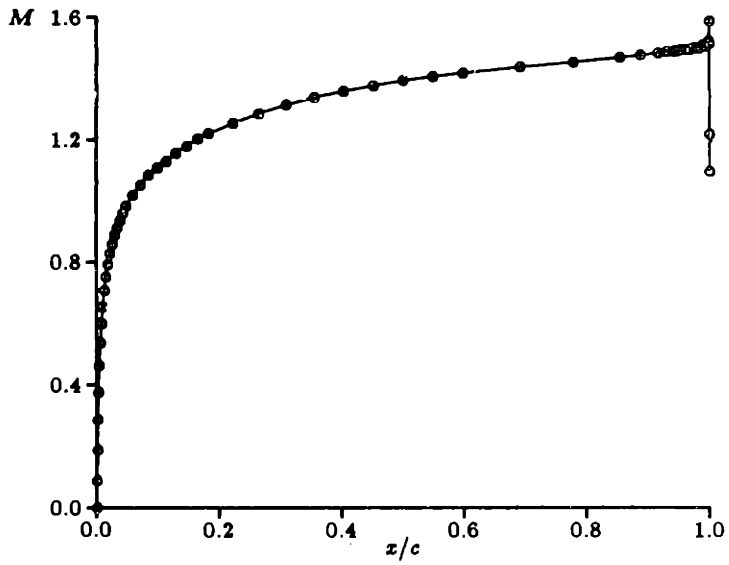


Figure 8.88: Grid 3 for which integration scheme failed, test case AGARD-04



a: computational grid near airfoil



b: surface Mach number distributions.
Line and symbols are current (final) solution.

Figure 8.89: Grid 3 with buffer zone added, test case AGARD-04

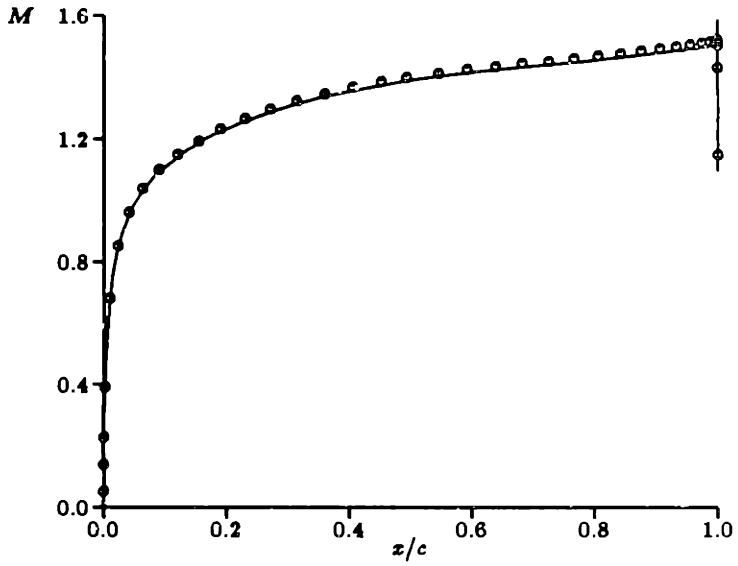


Figure 8.90: Surface Mach number distributions for case AGARD-04. Line is present solution, symbols are reference solution.

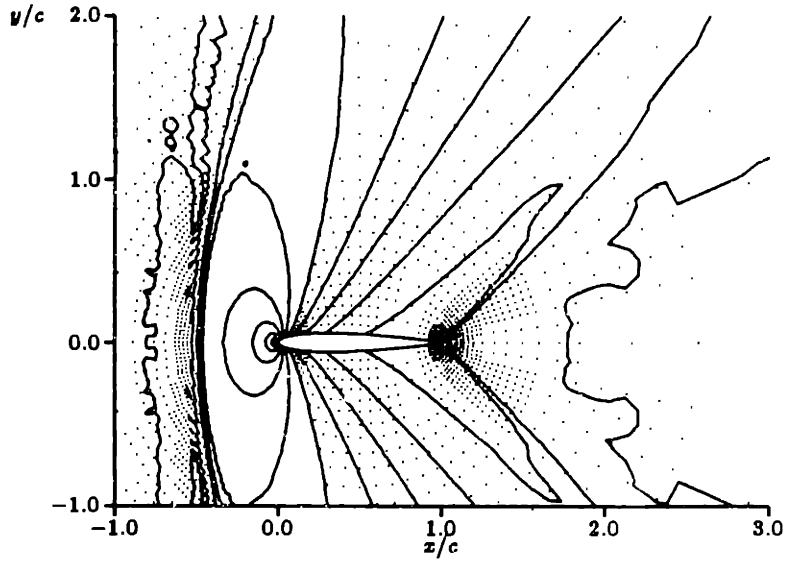


Figure 8.91: Mach number contours for AGARD-04, $\Delta M = 0.10$. Supersonic nodes are dotted.

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
0	544	672	0.0000	0.0979	78	212
1	1004	1216	0.0001	0.0949	139	518
2	1597	1920	0.0000	0.0952	262	1501
3	2911	3488	diverged		370	3072
3	4251	5272	0.0000	0.0948	476	5326
average AGARD value			0.0000	0.0954		
scatter				1.5%		

Table 8.9: Summary of solution evolution for AGARD-04.

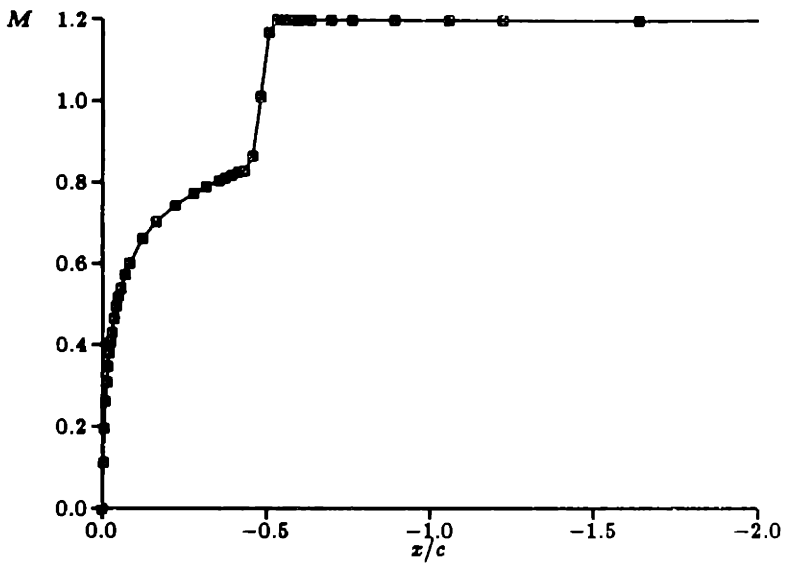
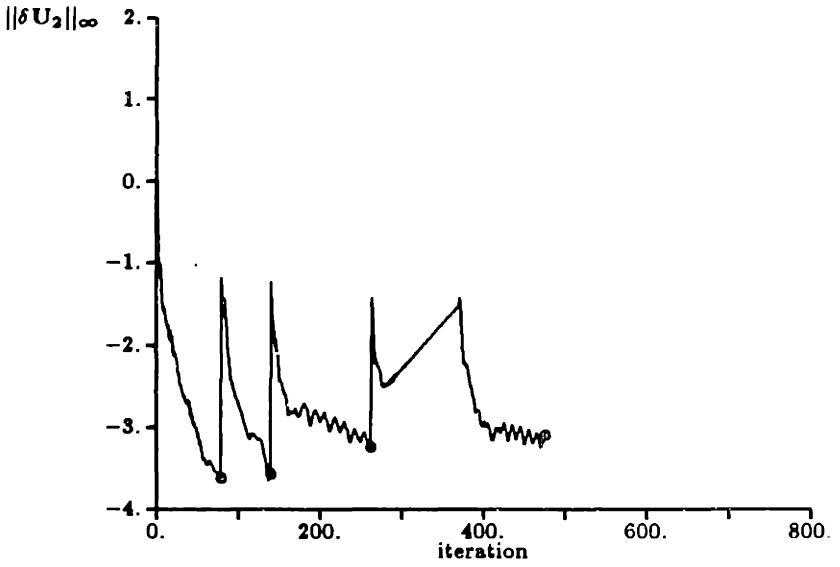
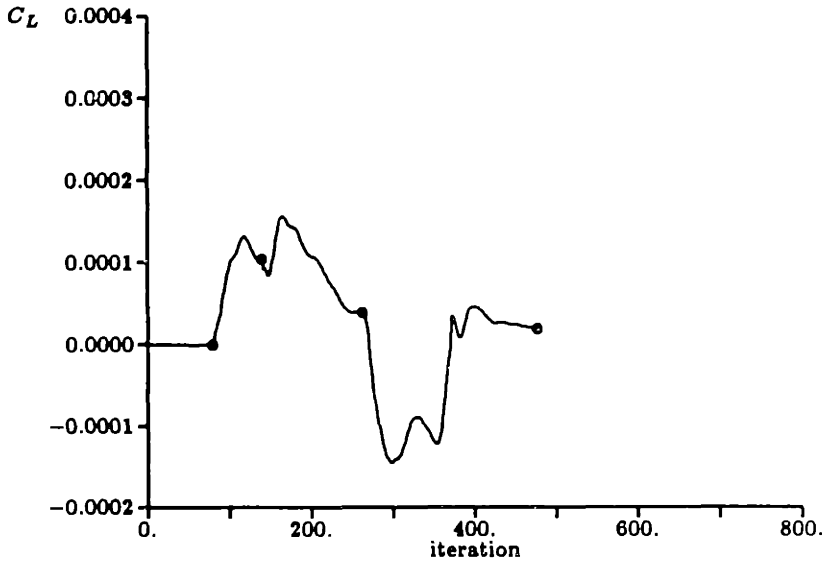


Figure 8.92: Mach number distribution along the upstream stagnation streamline (symmetry line), test case AGARD-04.



a: L_∞ norm of $\delta \mathbf{U}_2$.



b: Computed lift distribution.

Figure 8.93: Convergence histories for AGARD-04

8.6.5 AGARD-05

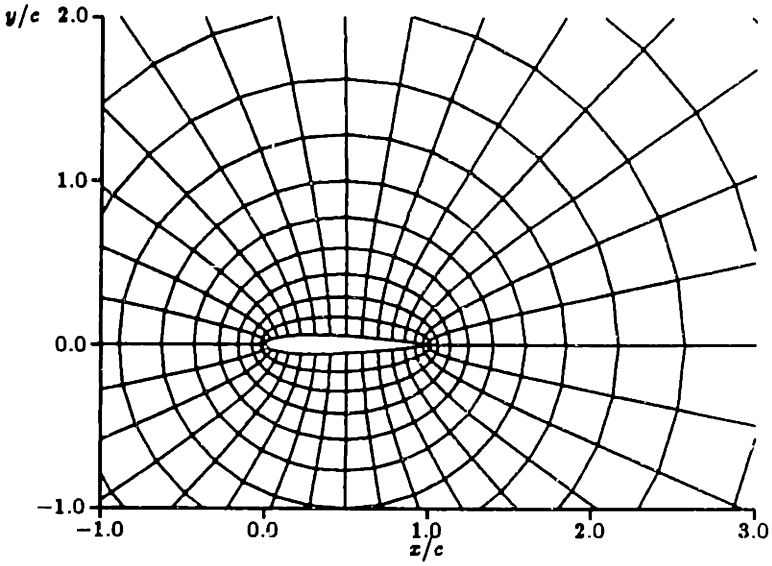
The final test case in this series, case AGARD-05, is topologically very similar to AGARD-04, except that the angle of attack is $\alpha_\infty = 7.0°$. Again, using the knowledge base above the scheme converged automatically.

The evolving grids and solutions for the first two adaptation cycles are shown in figures 8.94 through 8.97. Notice that figure 8.96 shows a grid on which the integration was not successful.

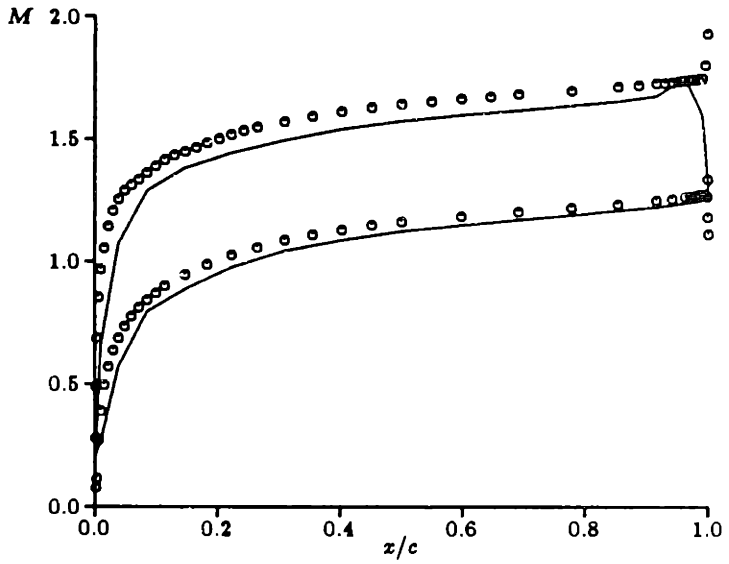
The final surface Mach number distribution is shown in figure 8.98 along with the solution published by AGARD[2]; the Mach number contours are shown in figure 8.99. In the latter, the bow shock is clearly identified. The exact location of the bow shock can be determined by plotting the Mach number distribution along the upstream x-axis, yielding $x/c = -0.62$ which agrees well with the average published value of $x/c = -0.60$

The convergence histories for this case are shown in figure 8.101, where the effect of the adaptation can be seen. Again the divergence on the grid 2 is clearly visible in both plots.

This solution evolution is summarized in Table 8.10, from which one can note that only two successful levels of refinement were required. Since this is a lifting case, the rule which sets a minimum number of adaptation cycles did not apply.

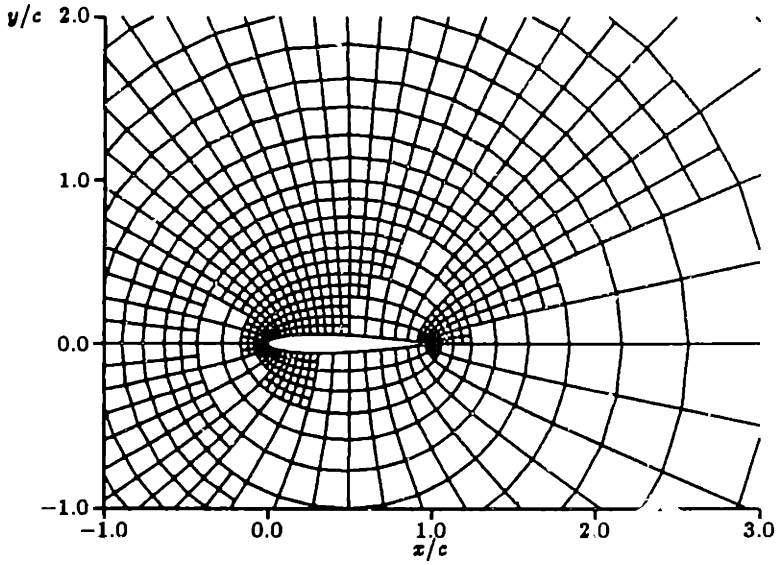


a: computational grid near airfoil

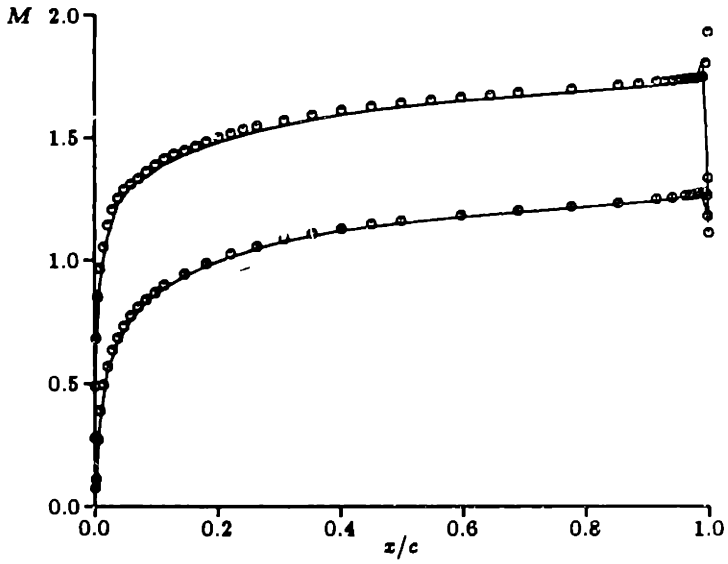


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.94: Grid 0 solution for AGARD-05



a: computational grid near airfoil



b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.95: Grid 1 solution for AGARD-05

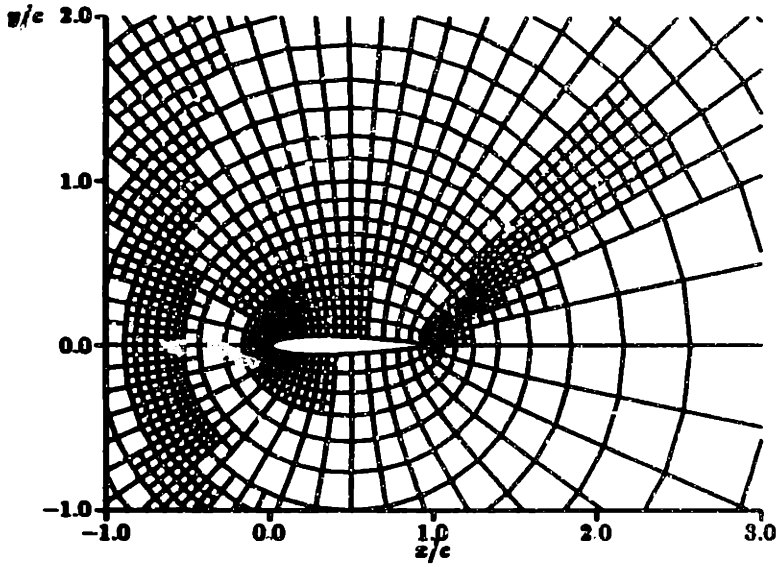
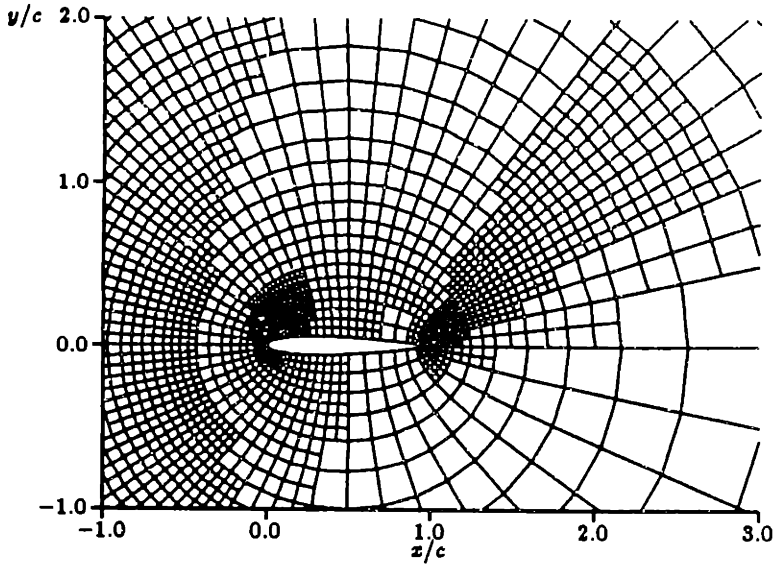
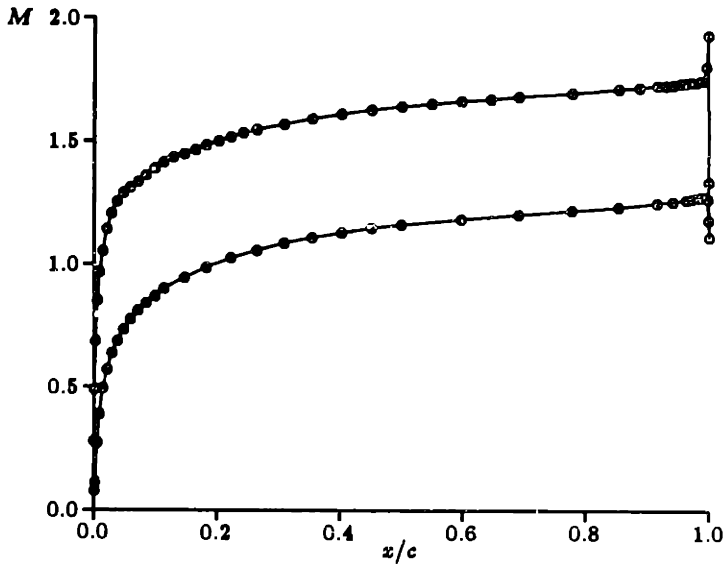


Figure 8.96: Grid 2 for which integration scheme failed, test case AGARD-05



a: computational grid near airfoil



b: surface Mach number distributions.
Line and symbols are current (final) solution.

Figure 8.97: Grid 2 with buffer zone added, test case AGARD-05

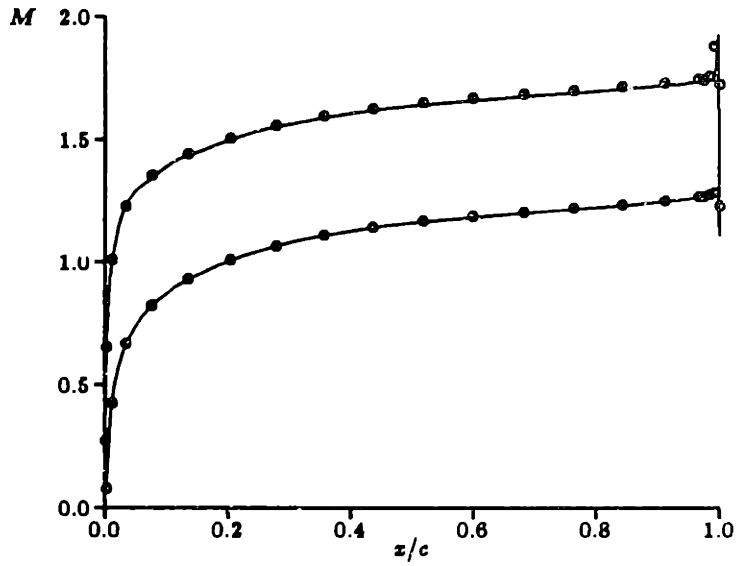


Figure 8.98: Surface Mach number distributions for case AGARD-05. Line is present solution, symbols are reference solution.

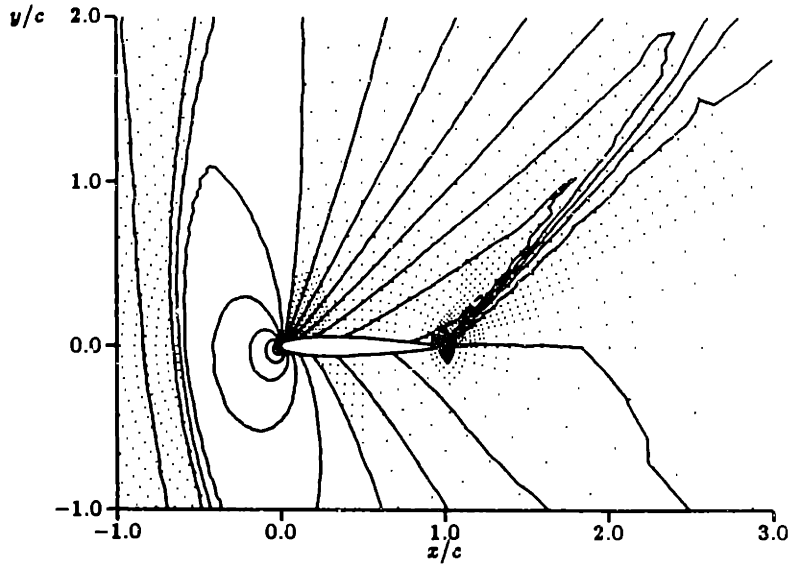


Figure 8.99: Mach number contours for AGARD-05, $\Delta M = 0.10$. Supersonic nodes are dotted.

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
0	544	672	0.5098	0.1568	82	223
1	1144	1396	0.5250	0.1536	160	669
2	2021	2428	diverged		226	1336
2	2871	3560	0.5285	0.1539	281	2126
average AGARD value			0.5211	0.1538		
scatter			2.7%	0.8%		

Table 8.10: Summary of solution evolution for AGARD-05.

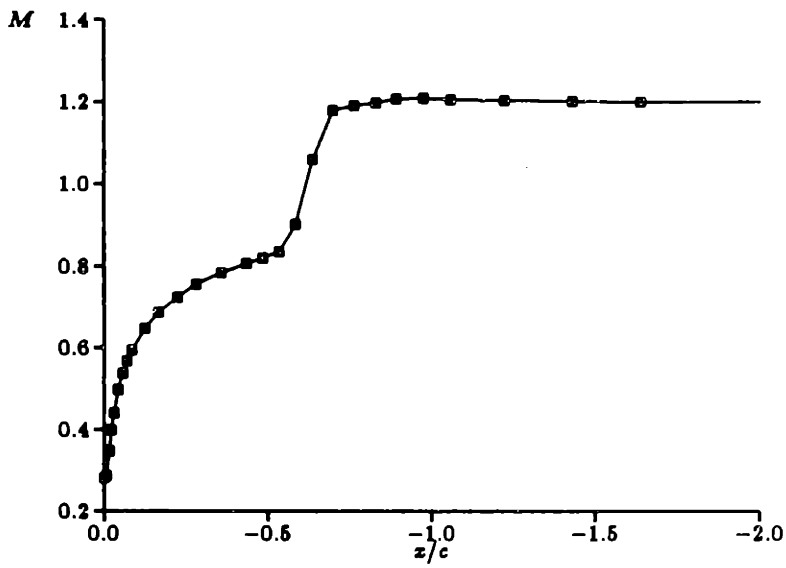
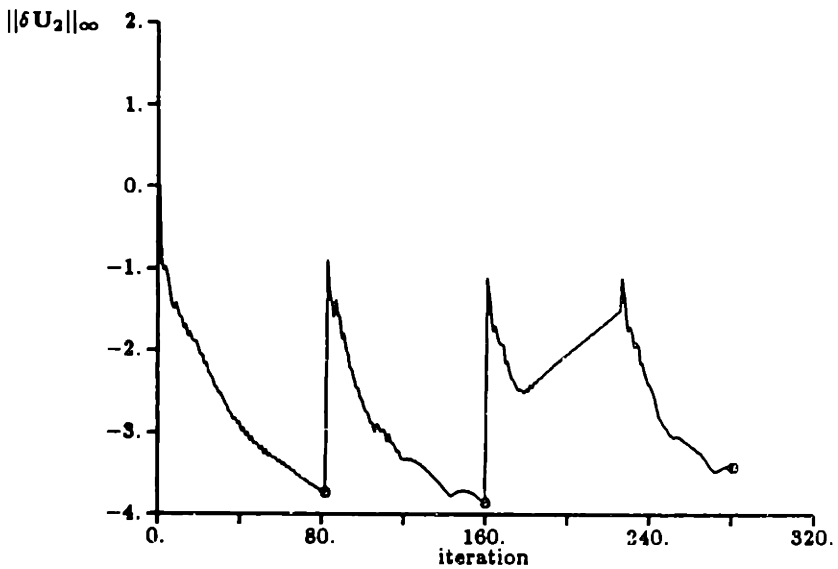
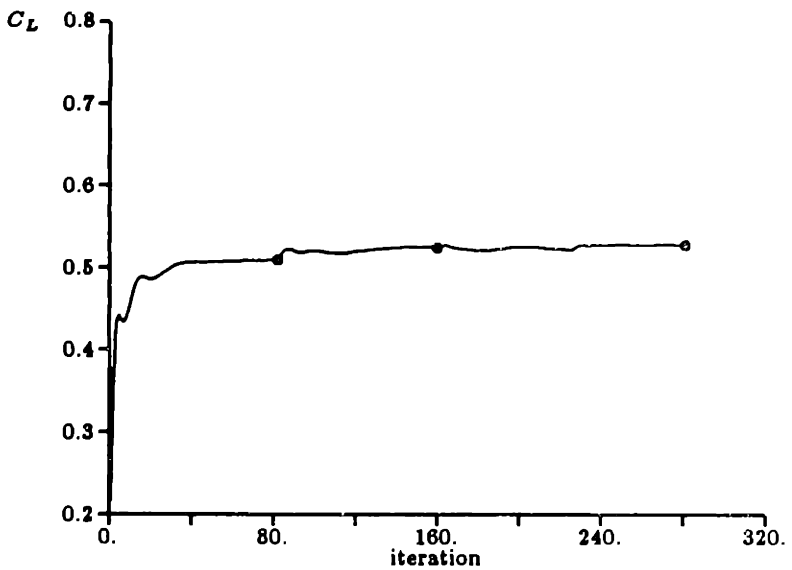


Figure 8.100: Mach number distribution along the upstream x-axis, test case AGARD-05.



a: L_∞ norm of $\delta \mathbf{U}_2$.



b: Computed lift distribution.

Figure 8.101: Convergence histories for AGARD-05

8.6.6 Summary

In the last five sections, the grid adaptation strategy was applied to five test cases with differing flow-field topologies even though they shared a common geometry and initial grid. Although some of the cases were not successfully computed the first time, appropriate adjustments in the adaptation strategy were made by simple rule additions to the knowledge base. The new knowledge base then resulted in successful calculations in all five cases.

Incidentally, the solution for AGARD-06 was re-computed with the final knowledge base, yielding the same solution as discussed in section 8.3.

8.7 O-type and H-type Mesh Solutions

In addition to the O-type mesh solutions computed above, the six AGARD test cases were also computed using H-type meshes. These latter solutions were made possible both by the ability to pre-embed grids in the vicinity of the airfoil for the initial solution as well as by high leading- and trailing-edge resolutions which resulted from adaptation.

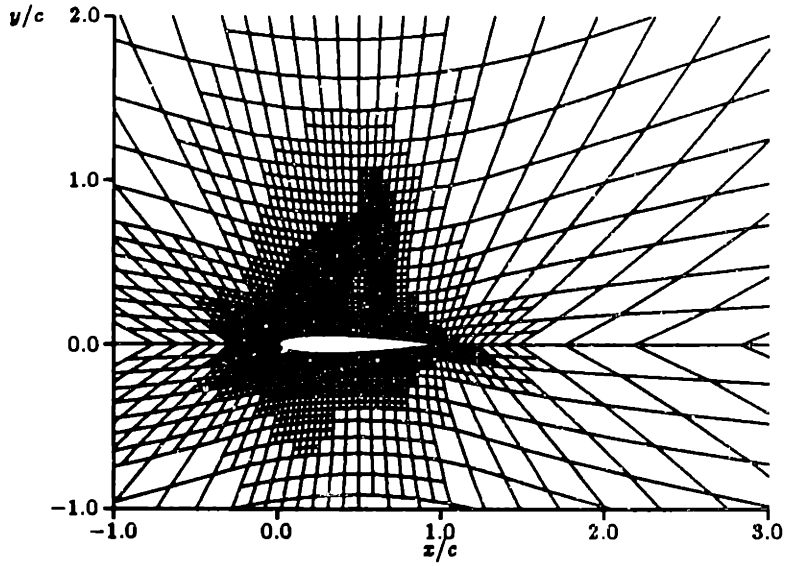
In the following discussion, the final H-type mesh solution is compared with the final O-type mesh solution (from above). For each test case, the final mesh shapes as well as a comparison of the surface Mach number distributions with the published AGARD solution is presented. Finally, a table comparing the evolution of both solutions is presented for each test case.

Solutions for the first test case (NACA-0012 at free-stream conditions $M_\infty = 0.80$ and $\alpha_\infty = 1.25°$) are presented in figures 8.102 and 8.103. Note that the H-mesh grid contains many more grid points, especially at the expansion fan emanating from the leading edge. Figure 8.103 clearly shows that the H-mesh solution agrees much better with the published solution than does the O-mesh solution. This better agreement is also evident in Table 8.11.

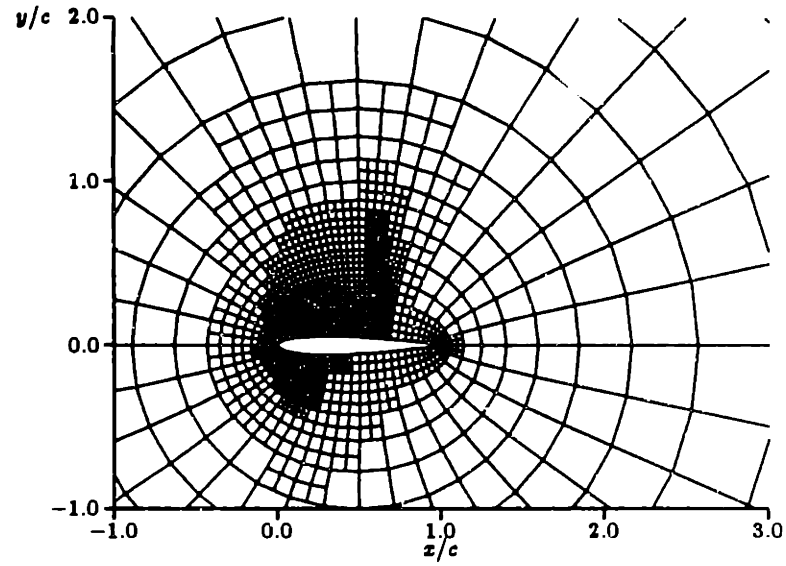
adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
			H-type mesh			
0	1512	1856	0.2804	0.0059	163	1232
1	2616	3232	0.3327	0.0146	276	2710
2	4527	5616	0.3539	0.0200	360	4612
3	6747	8384	0.3617	0.0022	434	7108
4	10117	12592	0.3628	0.0227	500	10447
			O-type mesh			
0	544	672	0.2594	0.0247	110	299
1	1022	1256	0.2991	0.0205	190	708
2	1941	2396	0.3115	0.0203	256	1349
3	3157	3892	0.3152	0.0203	313	2248
average AGARD value			0.3587	0.0228		
scatter			7.6%	9.9%		

Table 8.11: Summary of solution evolution for AGARD-01

Solutions for the second test case (NACA-0012 at free-stream conditions $M_\infty = 0.85$ and $\alpha_\infty = 1.00°$) are presented in figures 8.104 and 8.105. Again the H-mesh grid contains many more grid points, especially at the expansion fan emanating from the leading edge. However in this case, the Mach number distributions for the two cases are nearly identical, both disagreeing with the published solution as shown in figure 8.105 and Table 8.12.

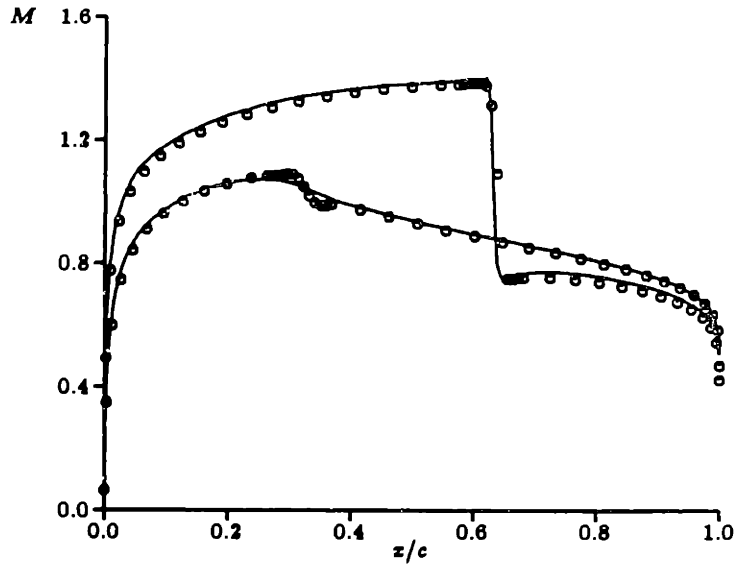


a: H-type mesh

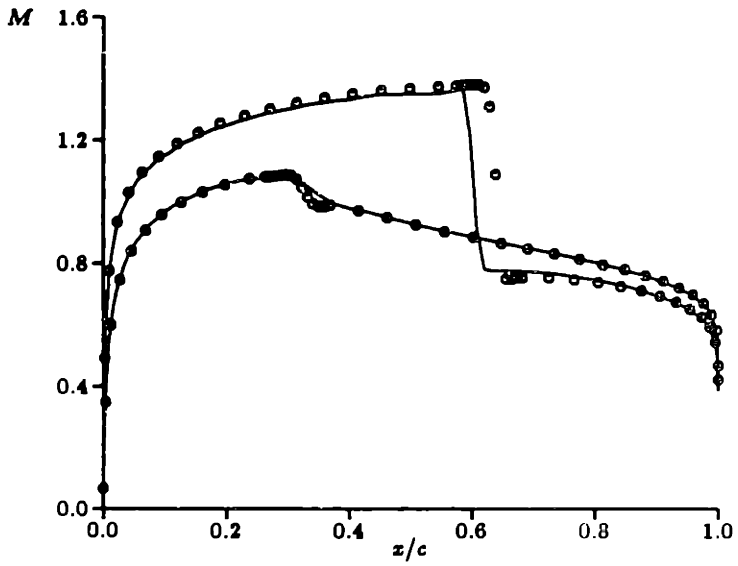


b: O-type mesh

Figure 8.102: Final computational grids for AGARD-01

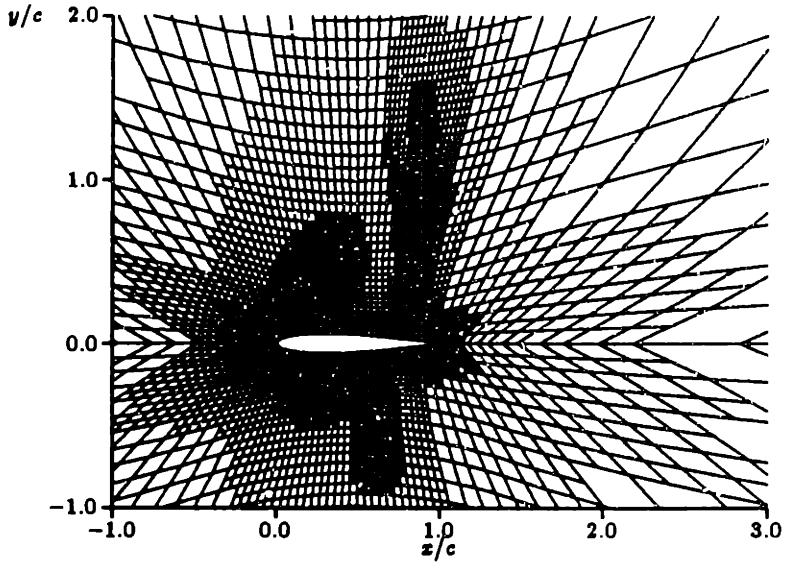


a: H-type mesh

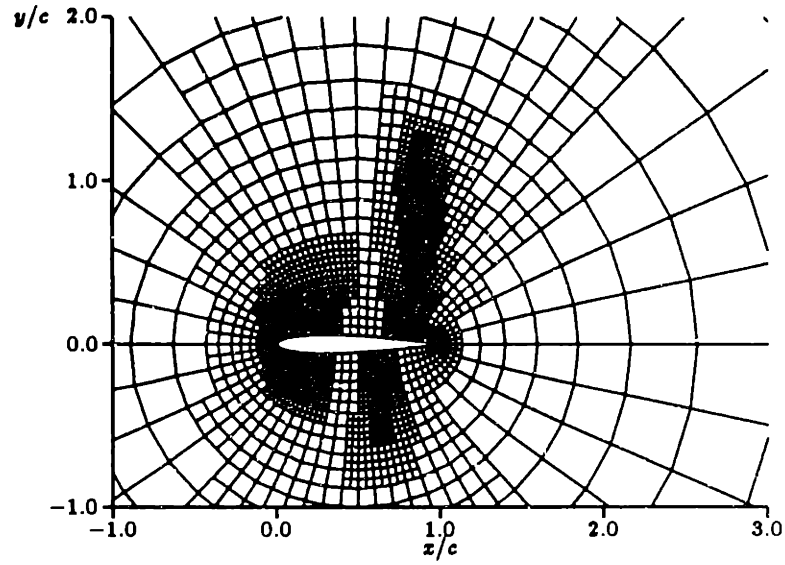


b: O-type mesh

Figure 8.103: Surface Mach number distributions for AGARD-01, symbols are AGARD published solution

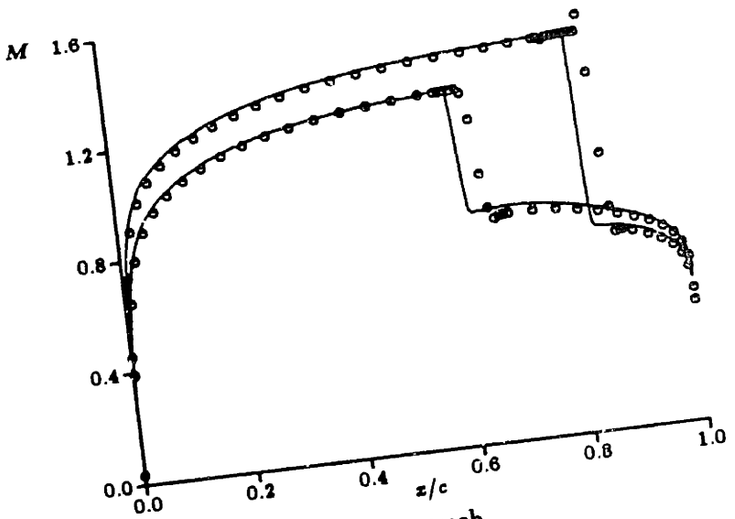


a: H-type mesh

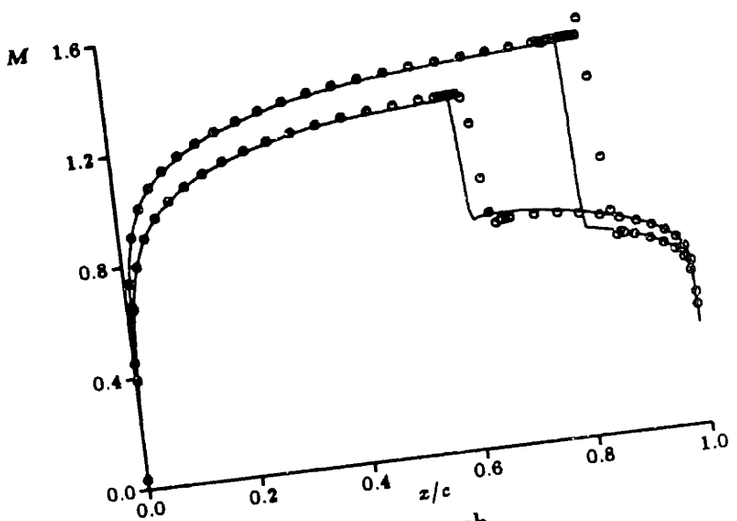


b: O-type mesh

Figure 8.104: Final computational grids for AGARD-02



a: H-type mesh



b: O-type mesh

Figure 8.105: Surface Mach number distributions for AGARD-02, symbols are AGARD published solution

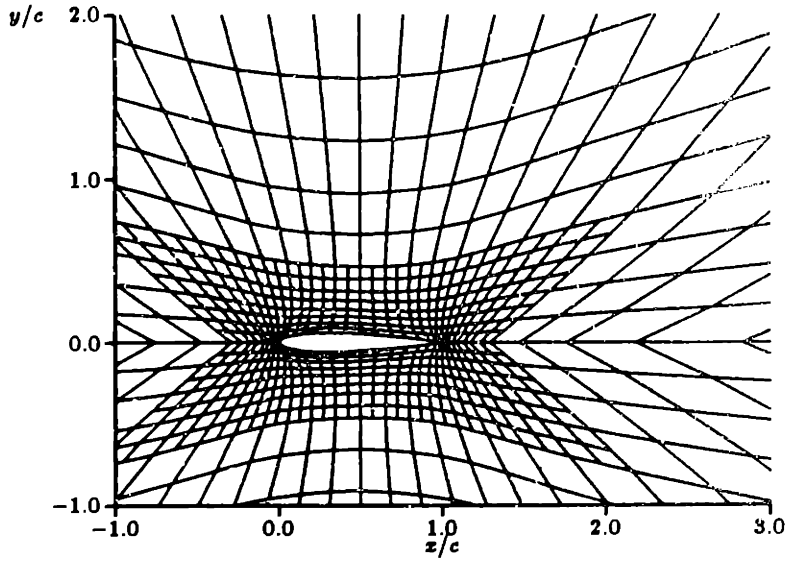
adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
			H-type mesh			
0	1512	1856	0.2734	0.0281	141	1066
1	2595	3208	0.3188	0.0416	260	2610
2	4599	5704	0.3408	0.0480	362	4955
3	6743	8304	diverged		449	7889
3	9045	11336	0.3443	0.0506	509	10602
4	18081	22880	0.3447	0.0513	568	15936
			O-type mesh			
0	544	672	0.2324	0.0474	118	321
1	1037	1276	0.2780	0.0471	207	782
2	1822	2216	0.3072	0.0486	284	1484
3	2968	3600	0.3224	0.0494	348	2434
4	5630	6896	diverged		435	4883
4	7746	9672	0.3217	0.0496	505	7594
average AGARD value			0.3532	0.0545		
scatter			16.4%	23.9%		

Table 8.12: Summary of solution evolution for AGARD-02

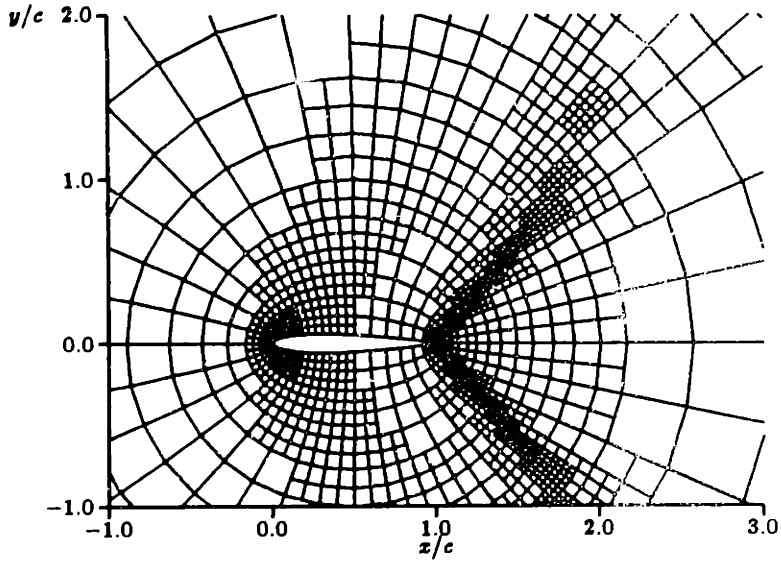
adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
			H-type mesh			
0	1512	1856	0.0001	0.0933	119	900
1	2823	3488	diverged		230	2466
1	3484	4352	diverged		333	4261
1	4255	5360	diverged		362	4878
1	5023	6384	diverged		391	5606
			O-type mesh			
0	544	672	0.0000	0.1114	110	299
1	1093	1328	0.0000	0.1078	168	616
2	1823	2192	0.0000	0.1076	244	1309
3	3092	3704	-.0001	0.1076	343	2839
average AGARD value			0.0000	0.1082		
scatter				0.7%		

Table 8.13: Summary of solution evolution for AGARD-03

The third test case (NACA-0012 at $M_\infty = 0.95$ and $\alpha_\infty = 0.00°$) resulted in a failure of the adaptation scheme. No attempt was made to determine the cause of the problem and hence the solution strategy was not modified to properly handle this case. As a result, the final solutions, which are presented in figures 8.106 and 8.107, do not accurately represent the differences between H-mesh and O-mesh solutions for this case. It should be noted that in this case, the O-mesh solution agrees quite well with the published AGARD solution.

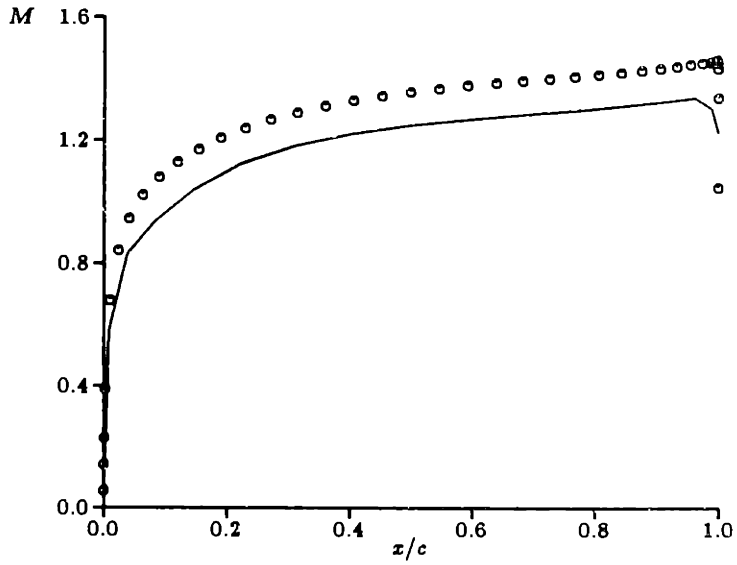


a: H-type mesh

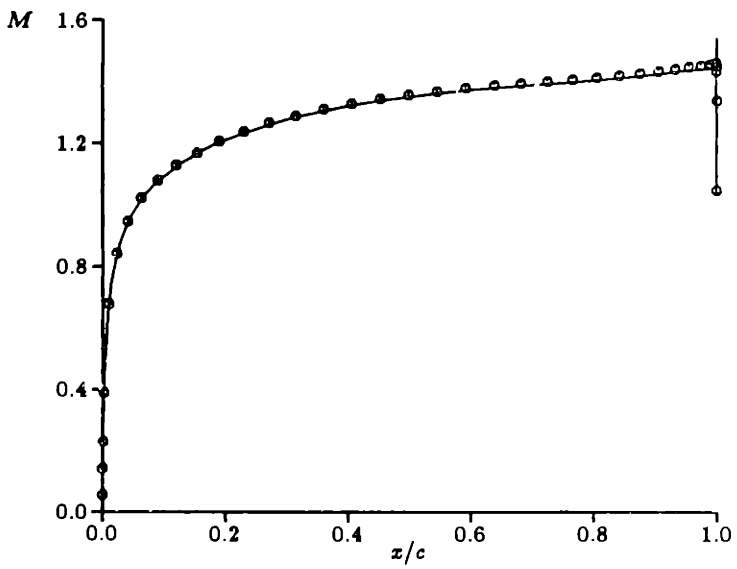


b: O-type mesh

Figure 8.106: Final computational grids for AGARD-03

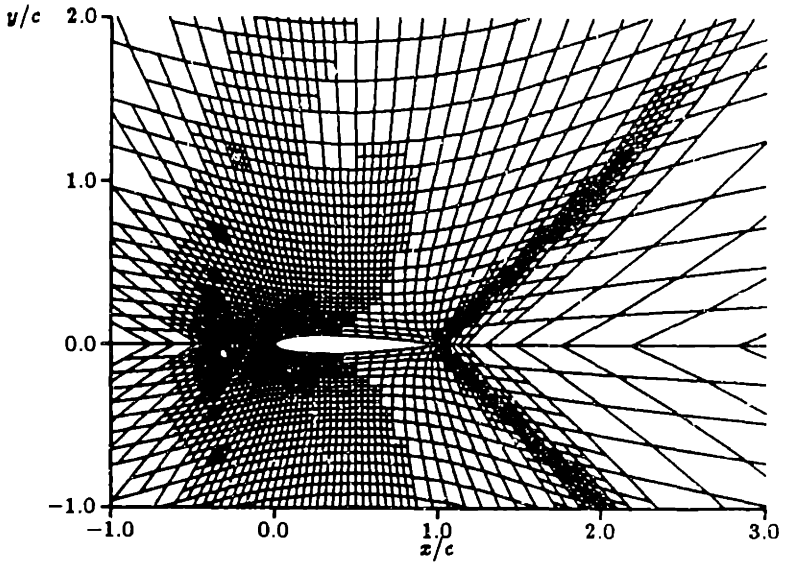


a: H-type mesh

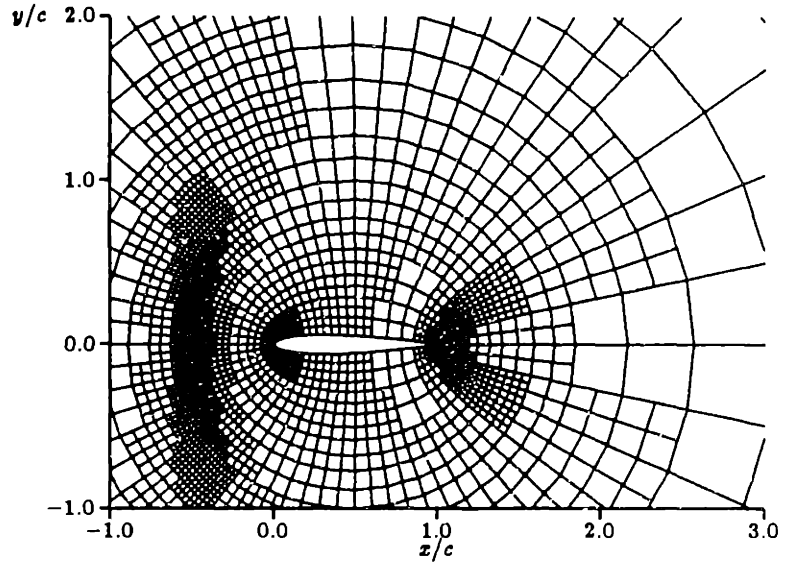


b: O-type mesh

Figure 8.107: Surface Mach number distributions for AGARD-03, symbols are AGARD published solution

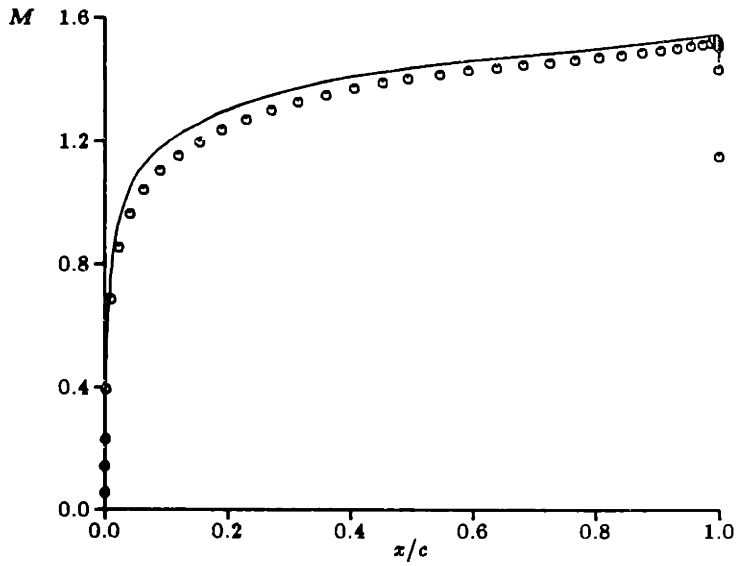


a: H-type mesh

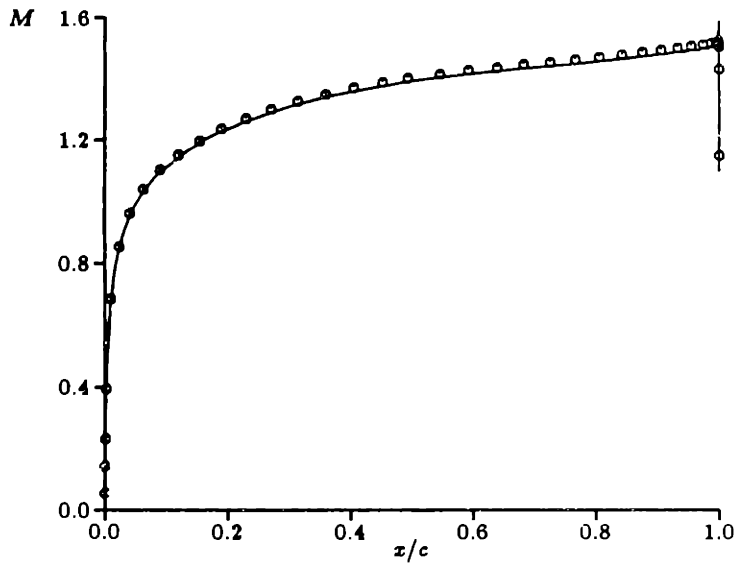


b: O-type mesh

Figure 8.108: Final computational grids for AGARD-04



a: H-type mesh



b: O-type mesh

Figure 8.109: Surface Mach number distributions for AGARD-04, symbols are AGARD published solution

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
			H-type mesh			
0	1512	1856	0.0001	0.0824	92	696
1	2853	3512	0.0000	0.0889	159	1651
2	4675	5680	0.0000	0.0929	224	3171
3	7539	9064	0.0000	0.0955	371	8712
4	10462	12788	0.0002	0.0942	553	18232
			O-type mesh			
0	544	672	0.0000	0.0979	78	212
1	1004	1216	0.0001	0.0949	139	518
2	1597	1920	0.0000	0.0952	262	1501
3	2911	3488	diverged		370	3072
3	4251	5272	0.0000	0.0948	476	5326
average AGARD value			0.0000	0.0954		
scatter				1.5%		

Table 8.14: Summary of solution evolution for AGARD-04

Solutions for the fourth test case (NACA-0012 at free-stream conditions $M_\infty = 1.20$ and $\alpha_\infty = 0.00°$) are presented in figures 8.108 and 8.109. Once again, the H-mesh grid contains many more grid points, especially at the expansion fan emanating from the trailing edge. But unlike the cases above, the H-mesh solution does not agree with the published solutions as well as the O-mesh solution does; this is in direct contrast with the observations for the first test case. Table 8.14 contains the summary of the evolution of the two solutions.

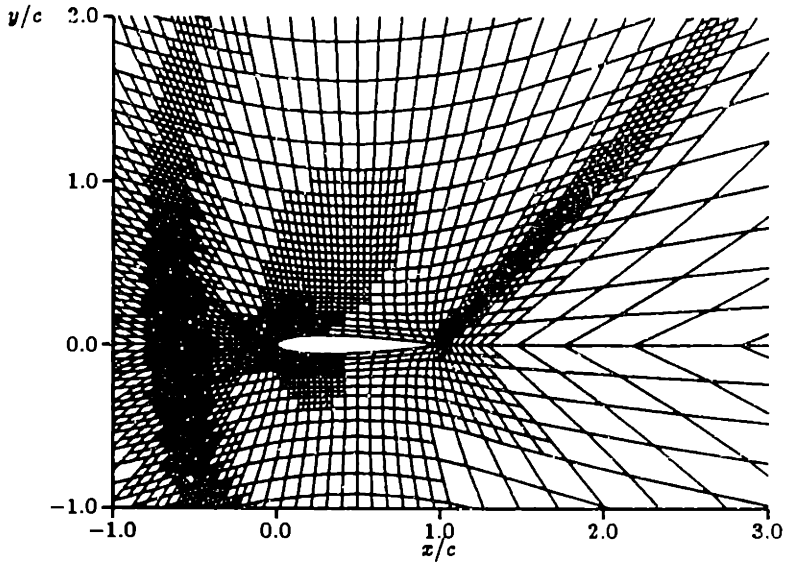
adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
			H-type mesh			
0	1512	1856	0.5083	0.1423	103	779
1	2818	3464	0.5204	0.1492	176	1807
2	4775	5816	0.5178	0.1522	282	4338
3	7357	8884	0.5200	0.1535	435	9966
			O-type mesh			
0	544	672	0.5098	0.1568	82	223
1	1144	1396	0.5250	0.1536	160	669
2	2021	2428	diverged		226	1336
2	2871	3560	0.5285	0.1539	281	2126
average AGARD value			0.5211	0.1538		
scatter			2.7%	0.8%		

Table 8.15: Summary of solution evolution for AGARD-05

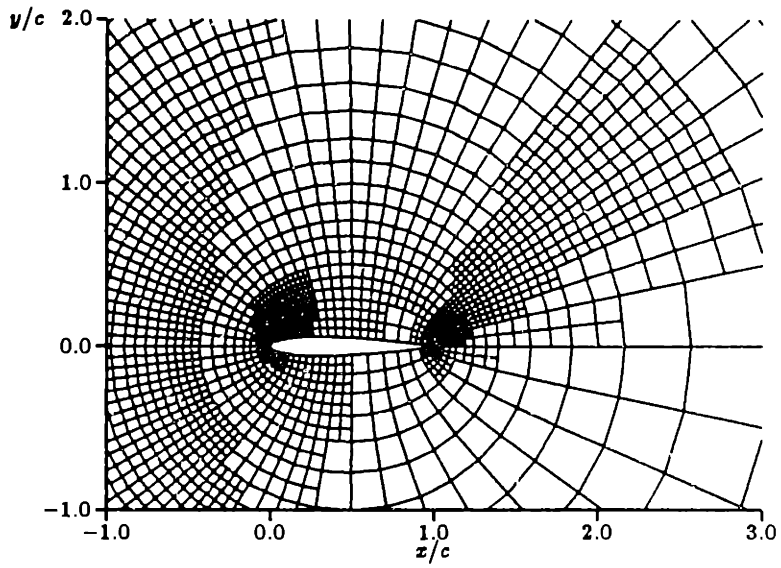
Solutions for the fifth test case (NACA-0012 at free-stream conditions $M_\infty = 1.20$ and $\alpha_\infty = 7.00°$) are presented in figures 8.108 and 8.109. In this case, both O-mesh and H-mesh solutions agree well with the published solution, as also shown by figure 8.111 and Table 8.15.

Solutions for the final test case (RAE-2822 at free-stream conditions $M_\infty = 0.75$ and $\alpha_\infty = 3.00°$) are shown in figures 8.112 and 8.113. As in the first test case, the H-type mesh solution agrees better with the published solution than does the O-mesh solution. Again, Table 8.16 bears this out.

In summary, comparison of the O-mesh and H-mesh solutions for the six AGARD test cases shows that the solutions are often different, and

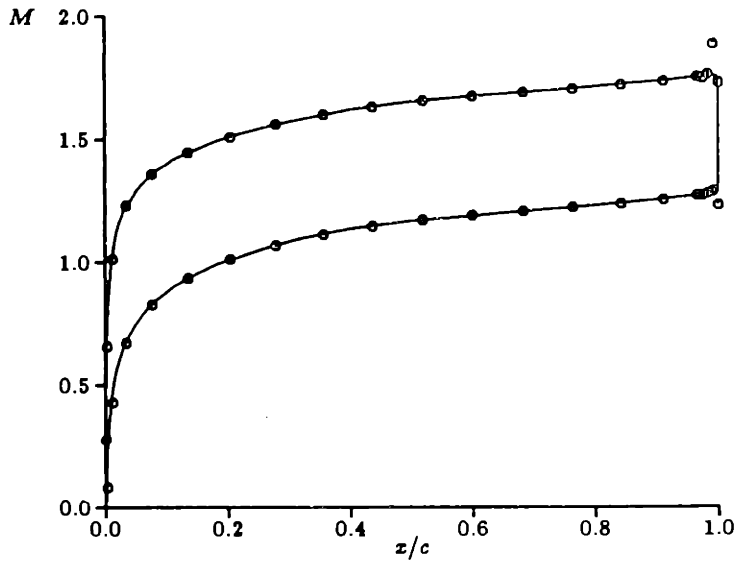


a: H-type mesh

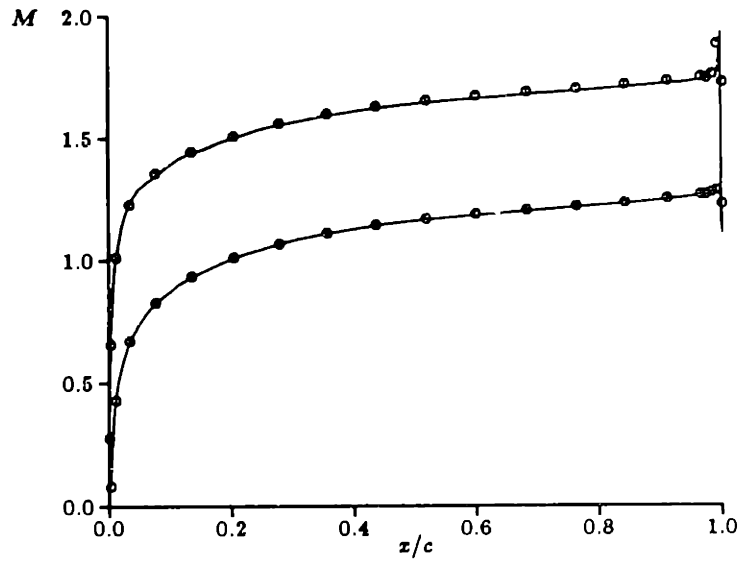


b: O-type mesh

Figure 8.110: Final computational grids for AGARD-05

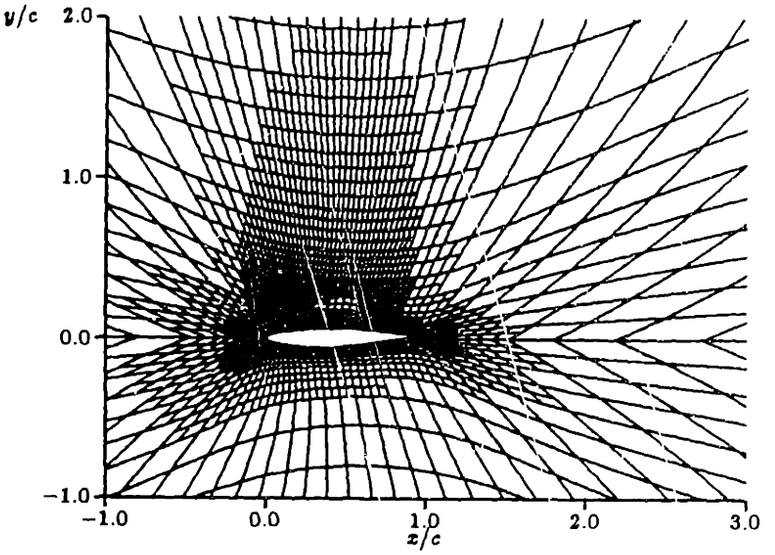


a: H-type mesh

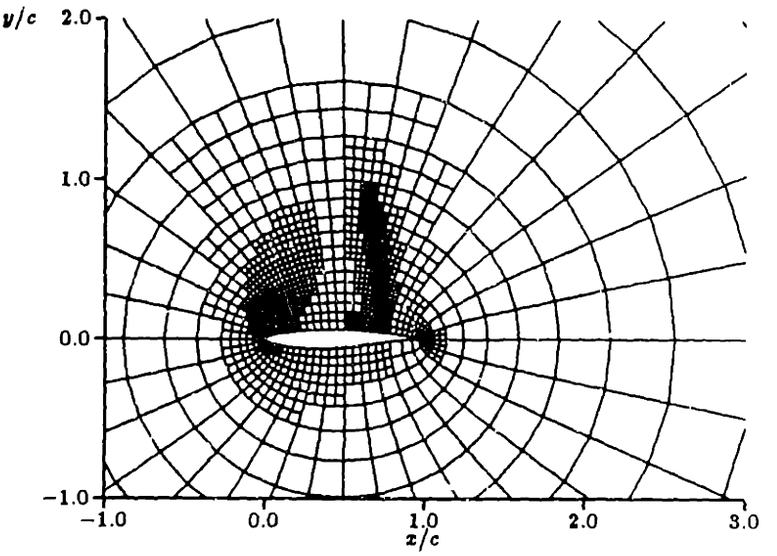


b: O-type mesh

Figure 8.111: Surface Mach number distributions for AGARD-05, symbols are AGARD published solution

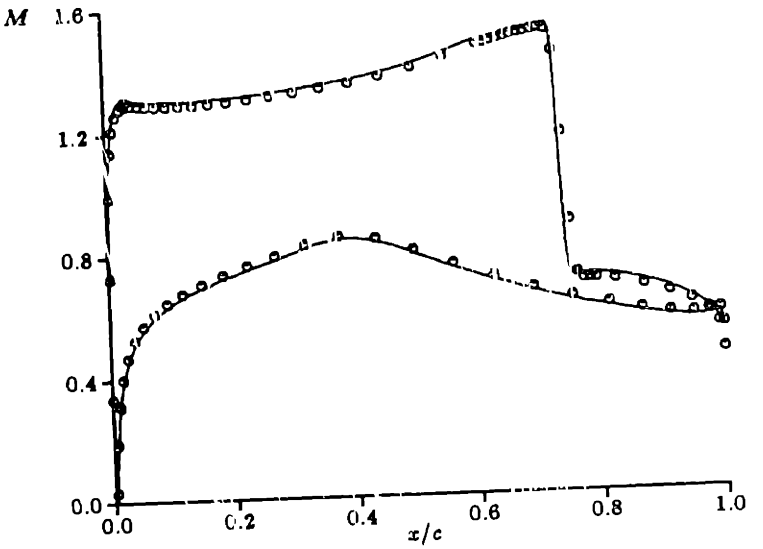


a: H-type mesh

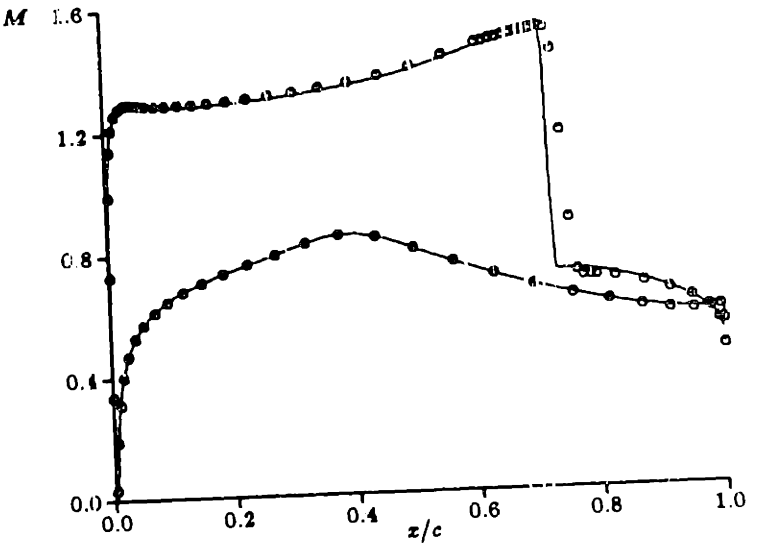


b: O-type mesh

Figure 8.112: Final computational grids for AGARD-06



a: H-type mesh



b: O-type mesh

Figure 8.113: Surface Mach number distributions for AGARD-06, symbols are AGARD published solution

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
			H-type mesh			
0	2808	3456	1.0684	0.0408	232	3257
1	4338	5360	1.1062	0.0442	524	9591
2	7955	9940	1.1063	0.0439	772	19455
			O-type mesh			
0	544	672	0.9026	0.0364	133	362
1	978	1200	0.9977	0.0357	235	861
2	1565	1912	1.0449	0.0385	327	1580
3	2250	2744	1.0574	0.0393	397	2368
4	3225	3924	1.0602	0.0394	458	3352
average AGARD value			1.1058	0.0467		
scatter			2.9%	11.8%		

Table 8.16: Summary of solution evolution for AGARD-06

furthermore shows that one type of topology is not clearly better in all cases. This is somewhat disturbing, since one would hope that solutions which were independent of mesh resolution would agree with each other. Since this is not the case here, it implies some sort of inconsistency in the formulation. It should be pointed out that the inconsistency is not a product of the adaptation process, as indicated by globally-refined solutions which yield different results for different topologies.

There still remain questions about the solution differences which result from different grid topologies. But the essential point here is that the current unstructured grid algorithm is capable of producing H-type mesh solutions which are comparable to O-type mesh solutions because of the enhanced resolution at the airfoil surface, and more specifically at the leading and trailing edges.

8.8 Application of MITOSIS to Complex Geometric Topologies

In this section, the adaptive grid refinement scheme is applied to geometric topologies which are considerably more complex than the isolated airfoils discussed above. These solutions have been computed not especially to show the effectiveness of the adaptation scheme but rather to demonstrate the flexibility of the semi-structured grid.

In both cases which follow, the initial computational grid has an H-type topology for each of the bodies in the flow field. This greatly simplifies the grid generation since one family of grid line is streamline-like and the other is nearly orthogonal to the first. Such H-type grid are normally avoided because they yield inaccurate solutions in the vicinity of the grid singularities at the leading and trailing edges. But with the adaptive refinement in the present scheme, such errors are controlled automatically.

8.8.1 Flapped Airfoil

The first test case in this series is for the subsonic flow over an airfoil with a trailing edge flap. This geometry, which was developed by the National Aerospace Laboratory (NLR) in the Netherlands, consists of a modified NLR-7301 airfoil with a flap placed 2.6% chord from the main airfoil.

The initial grid for this case is shown in figure 8.114, where part a consists of a view in the vicinity of the airfoil and part b shows the grid near the flap. Notice that this H-type mesh on both components has been pre-embedded so as to obtain reasonable airfoil/flap resolution.

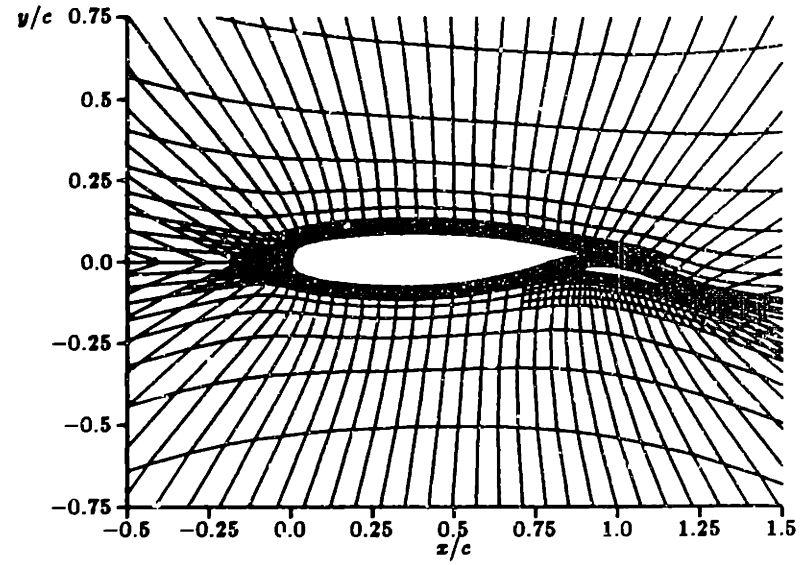
The computed pressure coefficient distribution on the initial grid is compared in figure 8.115a with the experimentally measured data of van den Berg[87]. Note that the agreement is quite good, except on the flap upper surface where the viscous wake (from the experiment) from the main airfoil probably plays a role. The pressure coefficient contours are shown in figure 8.115b.

Unfortunately grid adaptation failed for this geometry. Careful examination showed that the integrator's divergence was not due to embedding but rather to a deficiency in the current implementation of the basic integration scheme, that is a globally-refined case showed exactly the same divergent behavior. Most probably this is caused either by the low subsonic free-stream Mach number or the extremely blunt leading edge.

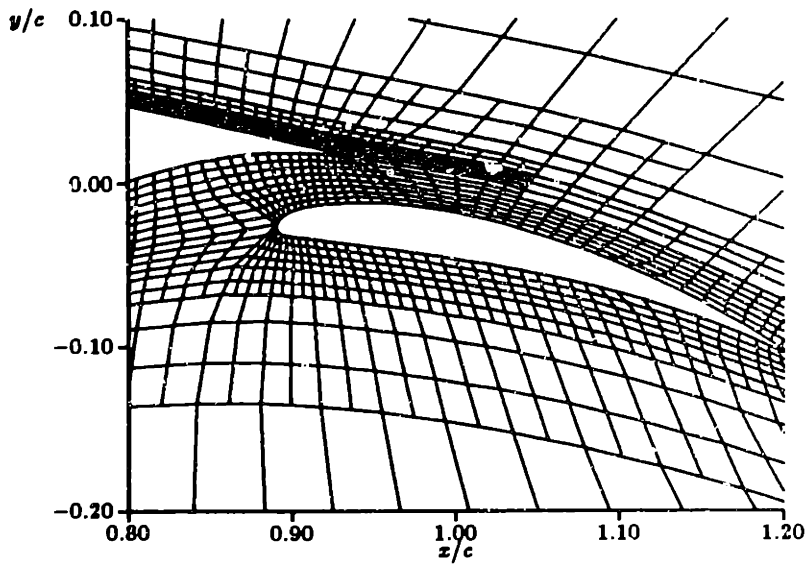
In any case, the computation of the solution about this airfoil/flap clearly shows the utility of employing H-type meshes about complex topologies. There is no reason to expect that the current formulation would not be applicable to three- of four-part airfoils as well.

8.8.2 Biplane

The last test case to be presented in this chapter is for a biplane of non-staggered NACA-0012 airfoils, with a gap of $y_{gap}/c = 0.50$. This biplane

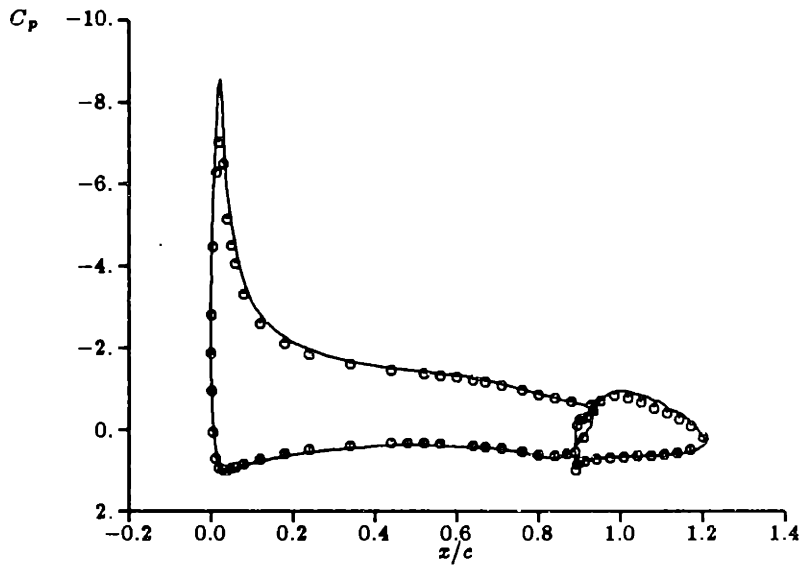


a: near airfoil

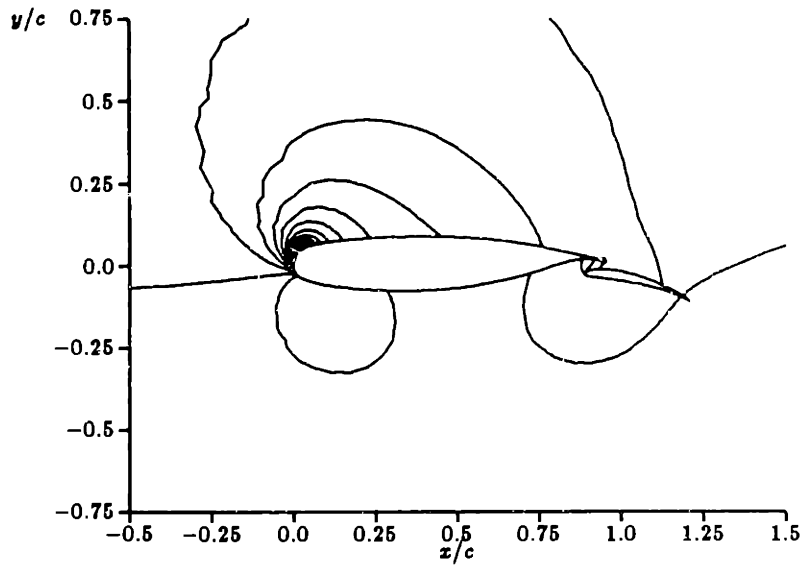


b: near flap

Figure 8.11 H-type computational grid for airfoil with flap



a: surface pressure coefficient disrtibution
Line is computed, symbols are experimentally measured[87]



b: pressure coefficient contours, $\Delta C_p = 0.50$

Figure 8.115: Computed pressure coefficients for airfoil with flap

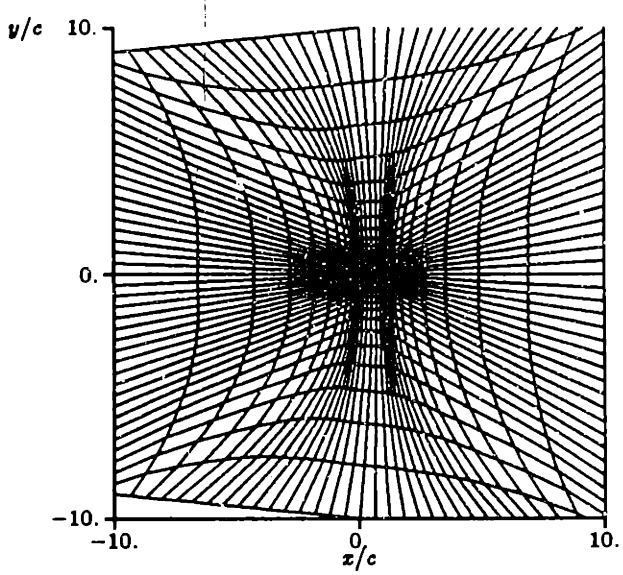


Figure 8.116: Grid 0 for biplane test case

operates at a free-stream Mach number of $M_\infty = 0.55$ and an angle of attack of $\alpha_\infty = 6.0°$. There are no comparison solutions available for this case.

The global computational grid, which is shown in figure 8.116, is an H-type grid, with 32 × 40 cells in the streamwise and crossflow directions respectively. The far-field boundary is a rhombus, with the boundaries placed 10 chord lengths upstream and 9 chord lengths downstream of the airfoils; the lateral extent is an average of 10 chords above and below the center of the biplane. Refinement singularities are located at the leading and trailing edges of both airfoils.

In this case as in the flapped airfoil case, in view of the coarse global grid, two levels of pre-embedding were used adjacent to airfoil surfaces. This action was taken because otherwise large total pressure losses are created at the poorly defined leading edges, resulting in almost certain divergence of the integrator. While it is true that a rule could be added to the knowledge base

adaptation cycle	number of nodes	number of cells	C_L	C_D	iterations	pseudo-CPU time
0	2263	2704	0.9535	0.0687	192	2172
1	3652	4396	1.1654	0.0154	346	4985
2	5906	7236	1.2200	0.0105	496	9414
3	10758	13404	1.2352	0.0086	604	15223
4	21187	26768	1.2335	0.0090	655	20626

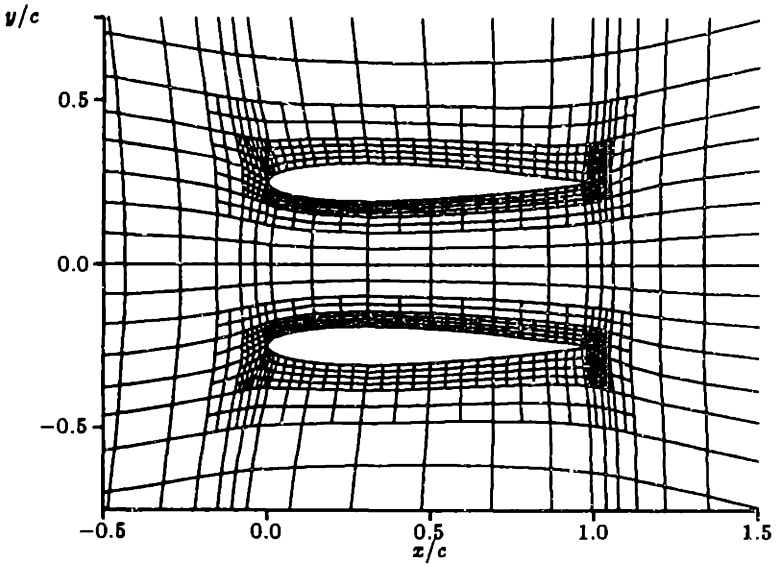
Table 8.17: Summary of solution evolution for biplane test case

to recover from that divergence on the initial grid, it seems more reasonable to prevent such a divergence by pre-embedding. It should be noted that if the pre-embedding turns out to be unnecessary, the grid adaptation procedures will automatically fuse (un-refine) the unwanted cells.

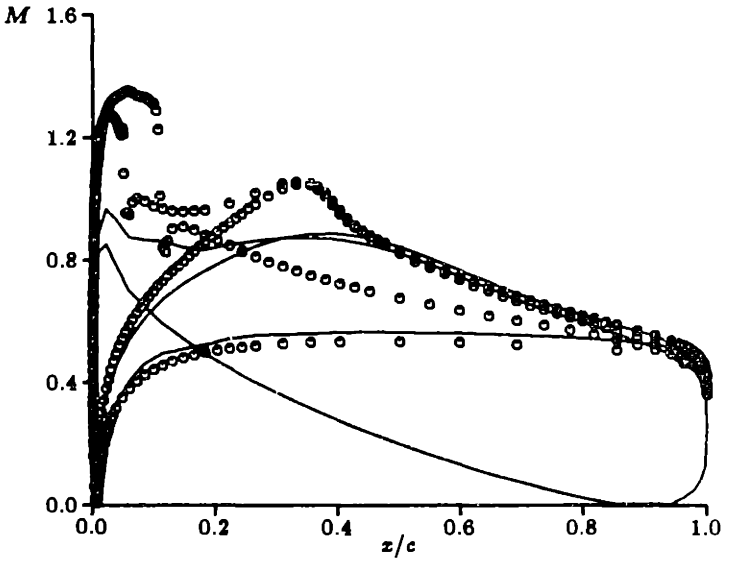
The evolving grids and solutions for this case are shown in figures 8.117 through 8.121. The solution for this case proceeded in a straightforward manner for this case, with no diverged solution encountered.

The Mach number contours for this final solution are shown in figure 8.122, where again the supersonic nodes are represented by dots. Note that the channel for this solution is nearly choked.

The convergence histories for this case are shown in figure 8.123, where the effect of the adaptation can be seen. The results from this test case are tabulated in Table 8.17, where one can note that the lift increases significantly with increasing refinement. One can also note that the pseudo-CPU time required for this case is much greater than the time required for any of the isolated airfoil cases.

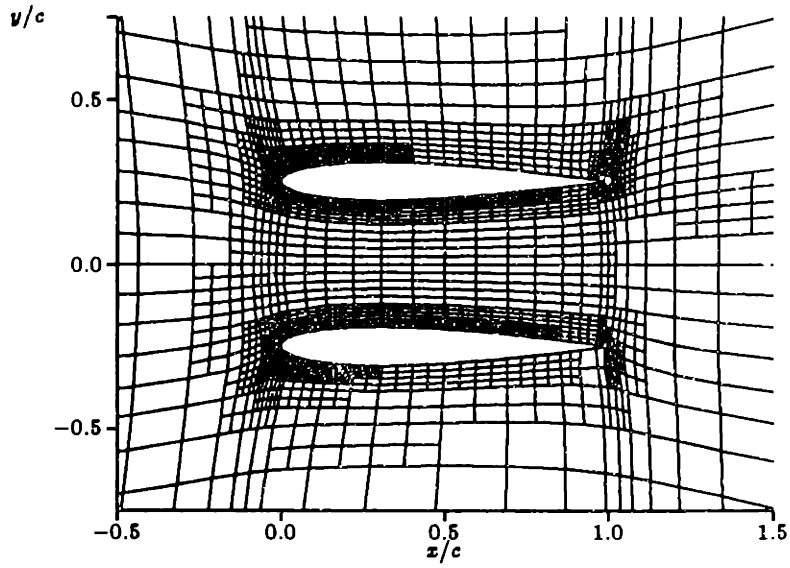


a: computational grid near airfoils

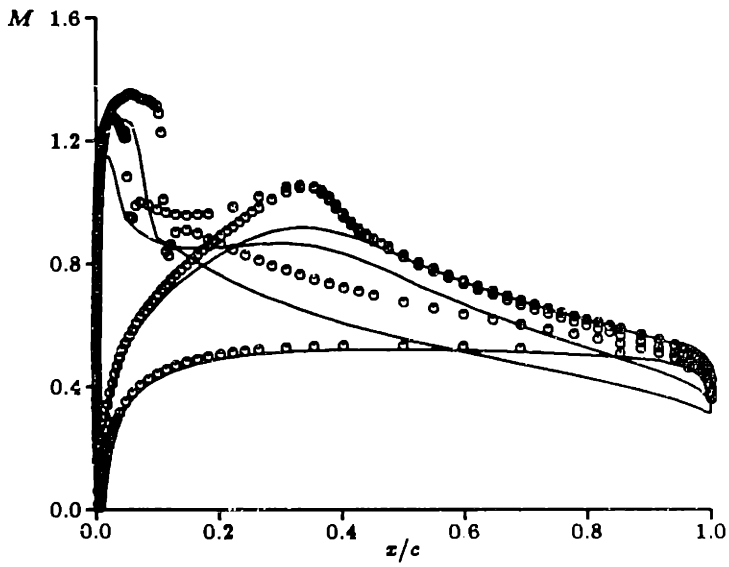


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.117: Grid 0 solution for biplane

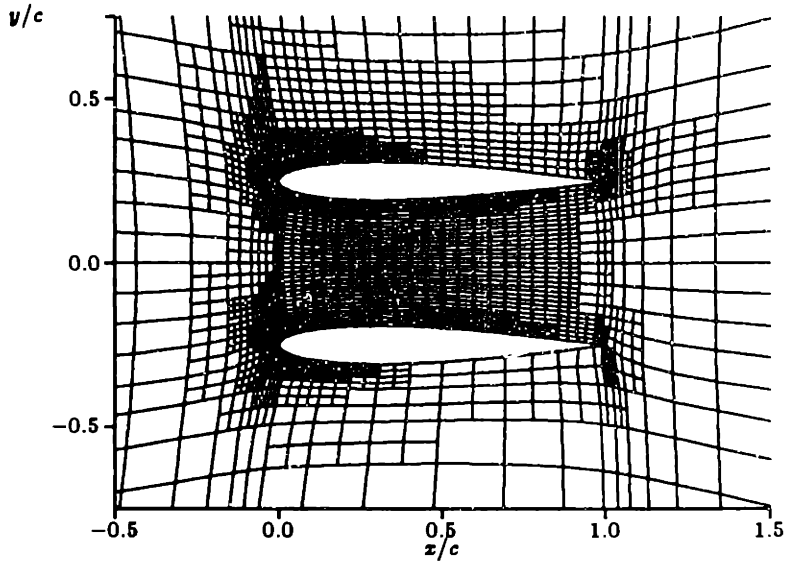


a: computational grid near airfoils

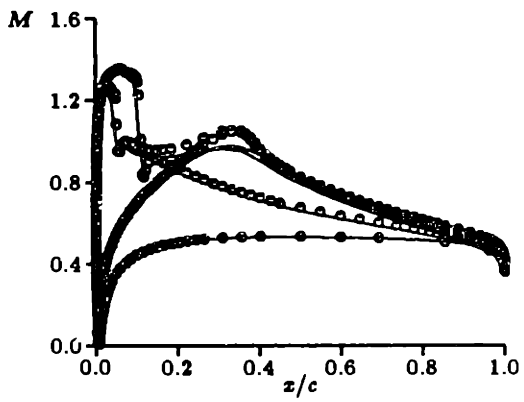


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.118: Grid 1 solution for biplane

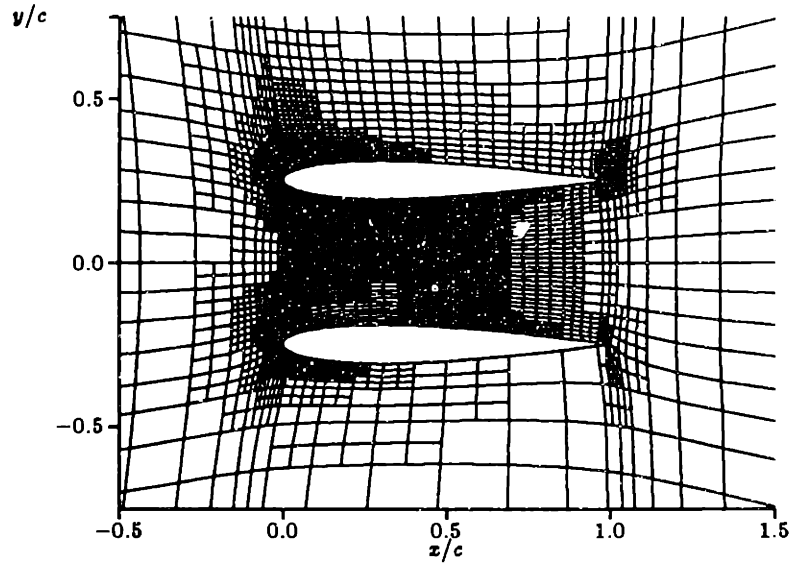


a: computational grid near airfoils

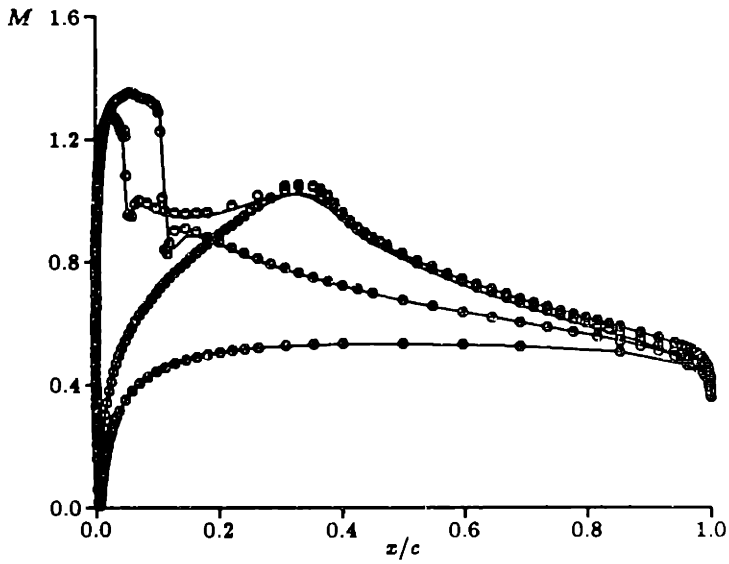


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.119: Grid 2 solution for biplane

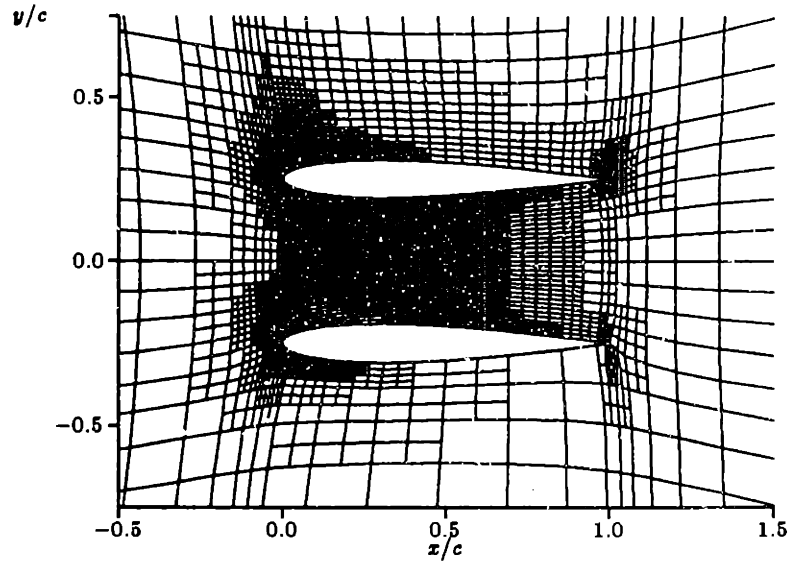


a: computational grid near airfoils

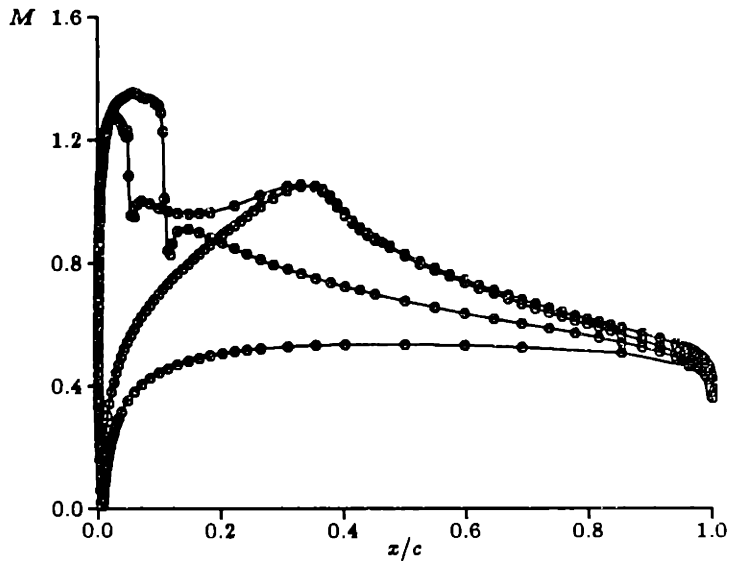


b: surface Mach number distributions.
Line is current solution, symbols are final solution.

Figure 8.120: Grid 3 solution for biplane



a: computational grid near airfoil



b: surface Mach number distributions.
Line and symbols are current (final) solution.

Figure 8.121: Grid 4 (final) solution for biplane

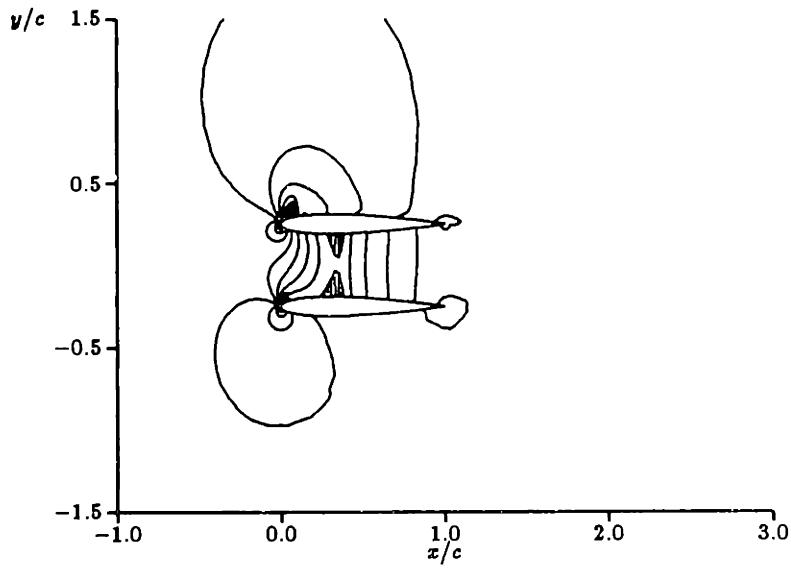
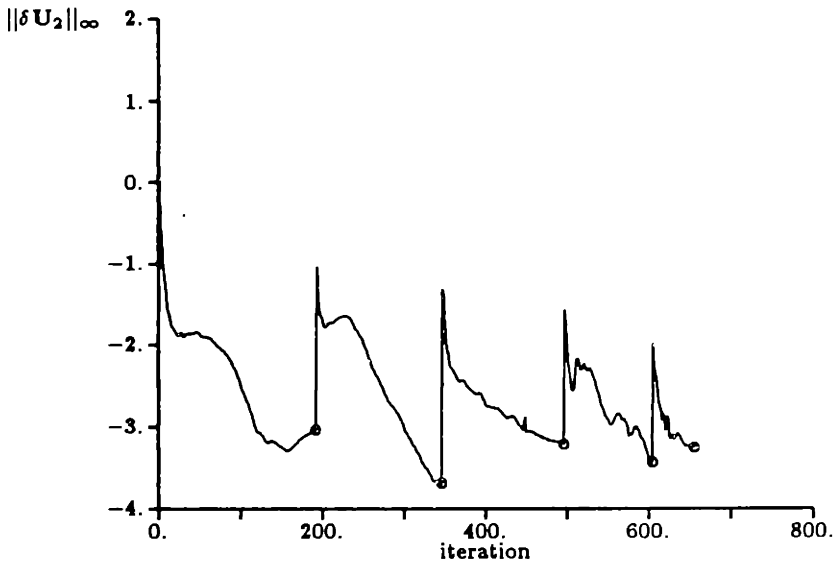
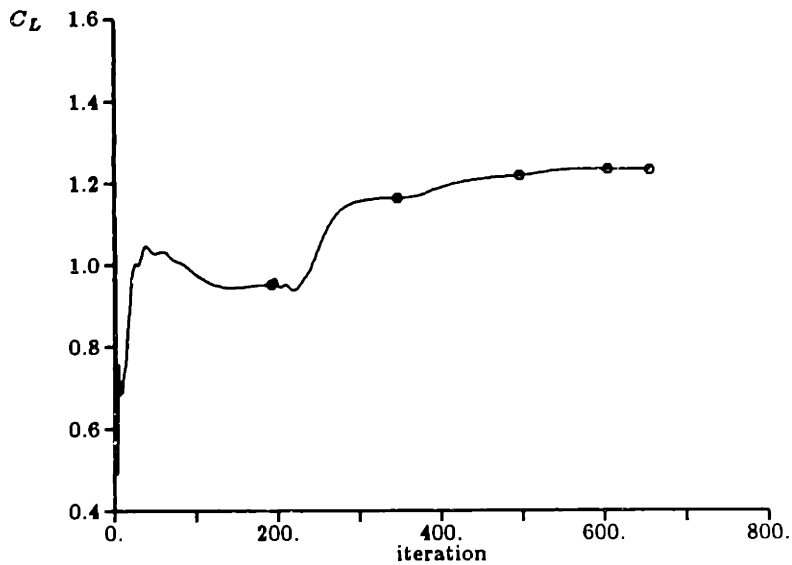


Figure 8.122: Mach number contours for biplane test case, $\Delta M = 0.10$. Supersonic nodes are dotted.



a: L_∞ norm of $\delta \mathbf{U}_2$.



b: Computed lift distribution.

Figure 8.123: Convergence histories for biplane test case

Chapter 9

Concluding Remarks

This final chapter of the thesis begins by providing a summary of the work. After a brief description of the adaptation scheme, the *new* algorithms and *new* computational techniques which have been developed in order to compute adaptive solutions are identified. The summary concludes with a review of the test cases which have been performed and the salient points to be drawn from each.

The second section describes some extensions to the adapted grid methodology which this author feels should be fruitful; this section is followed immediately by a discussion of some of the limitations of the techniques developed herein.

The final section presents a concise statement of the major conclusions that have been reached on the basis of the study.

9.1 Summary

A grid adaptation methodology for complex transonic flow-fields has been developed. The basic approach is to integrate the unsteady Euler equations to steady state on successive grids until solutions on successive grids are "close" to each other, indicating grid independence of the final solution. Typically fewer than five successive grids are required to achieve force coefficients which are grid-converged to within one percent.

The grids all share a common, fixed global grid but contain successive levels of irregularly-shaped embedded regions which are automatically generated so as to be refined in the vicinity of detected features. The Euler integration is performed using a one-step Lax Wendroff-type integration scheme and the multiple-grid accelerator developed by Ni[67]. An extension of the multiple-grid accelerator is used to couple the solutions on the global and embedded regions, in much the same way as was previously developed by Usab[84]. The locations of the features are determined automatically by computing an appropriate refinement parameter throughout the domain, and then setting thresholds to separate the features from the background physics.

Because of the small number of successive grids employed (typically less than 5), and the large number of iterations required to achieve convergence on each grid (typically more than 100), the CPU-time required to detect features and modify the grid and its data structure is typically less than five percent of the total.

The creation of this adapted grid technique required the development of a number of new concepts and algorithms. The most significant of them are described in the following paragraphs:

- Far-field boundary condition treatment — A new far-field boundary condition treatment has been developed for general hyperbolic equations which is based upon the principle of characteristic propagation

and prescribed physical conditions. This new technique, which when applied, is based only upon information at the boundary node (no extrapolation), has been applied to the Euler equations.

- Multiple-grid transport operator — Because the multiple-grid accelerator plays a central role in the coupling between the embedded and global grids, the operators used to transfer information amongst the various grid levels were carefully examined. This lead to a new transport operator which results in convergence rates which are more dependent on the coarse rather than the fine mesh resolution. This new operator turns out to be very similar to the one recently published by Ni[66].

- Smoothing coefficient equation — Most of the current Euler integration schemes employ a fourth-order difference smoothing operator to damp out oscillations while introducing only a third order error. Unfortunately, such an operator is difficult to implement on unstructured grids. As a compromise, a spatially varying second-order difference operator is employed here; the coefficient of which is computed using a newly developed smoothing coefficient equation. The latter has the property of resulting in a smooth distribution of the coefficient while remaining efficient to compute.

- Convergence checking algorithm — In most time-marching solutions of the Euler equations, the scheme is said to converge when the residual falls below some arbitrarily specified level. Fortunately, the force coefficients, which are the primary output of the calculation, converge to their final value much sooner than the residual converges. This observation gave rise to a new convergence checking scheme which detects convergence of the force coefficients rather than at some specified level of residual. The savings obtained with such a technique can be

substantial, especially at intermediate levels of refinement.

- Grid generation — The global grids are generated in this work using a new technique which is a modification of the standard elliptic grid generation techniques. Here, rather than controlling the grid spacing at boundaries through the use of forcing functions (as in the Poisson generation techniques), the spacing is controlled by solving Laplace's equation on a stretched mesh as suggested by Drela[30].

- Threshold selection — Separating the flow features from the background physics requires that one be able to determine where the refinement parameter is "large". The technique used here to define "large" is a newly developed threshold selection algorithm which uses the shape of the cumulative distribution function. This technique has been shown to be reliable in all of the cases examined.

- Expert system — A new expert system, with both forward- and backward-chaining inference engines has been developed. This expert system is different from most others in two respects: it is written in FORTRAN to enhance its portability to all scientific computers, including supercomputers; and it is designed on the assumption that most of the actions which the expert system is to perform are external procedures.

The adapted grid program MITOSIS was developed and tuned based primarily on the results of two test cases. Both consisted of channel flows, in which either a circular arc or a sine-squared bump was positioned on the lower channel wall. The channels were operated at subsonic, transonic, and fully supersonic flow conditions. These cases were used primarily to test the various new components developed for the Euler integration scheme, including its embedded mesh properties.

The next test case which was employed was the AGARD Working Group 7 test case 06, that is an RAE-2822 airfoil operating at $M_\infty = 0.75$ and $\alpha_\infty = 3.0°$. This case, which possesses a strong normal shock on the airfoil upper surface was used to develop the basic adaptation strategy.

Following that, a series of five test cases were employed to test the robustness of the adaptation strategy. These test cases, AGARD cases 01 through 05, are all based upon the same geometry (an NACA-0012 airfoil) but for various free-stream conditions. Various deficiencies in the original adaptation strategy which were subsequently remedied were uncovered by consideration of these cases.

The final two test cases were aimed at proving that the unstructured grid employed for irregularly-shaped embedded regions introduced a side benefit: calculations over complex geometric topologies can be simply performed. To show this, the test cases employed were an airfoil with a trailing edge flap and a biplane.

9.2 Possible Extensions of the Adaptation Methodology

The technique described above is very general and can be extended to new capabilities in a number of ways. The following are believed to be some of the more fruitful.

The first is an application of the grid adaptation technique to other flow domains that are describable by the Euler equations. Examples include incompressible and hypersonic flows. The modifications to the technique which are required fall into two broad categories. First, some of the numerical models in the Euler integrator may need to be slightly adjusted to apply to the new speed regime, such as the multiple-grid accelerator or the smoothing operator and its coefficients. Second, the adaptation strategy (which is contained in the knowledge base), may need to be modified to

use different refinement parameters or other strategy changes. An obvious example is incompressible flow, where using the first difference of density as the refinement parameter is clearly unsatisfactory.

The second extension involves changing the set of governing equations. For example, it is possible to modify the present scheme to use the Navier-Stokes equations[29,50]. Such a change involves primarily modifications to the integration scheme; the feature detection, and grid and data structure handling, and expert system parts of the program should remain unaltered. In addition, the knowledge base would probably have to be changed to reflect an adaptation strategy appropriate for the new flow domain types.

An extension to three dimensions also seems fruitful. There are no fundamental limitations in any of the procedures developed here that preclude such an extension. It should be recognized however that such an undertaking is not trivial, especially in light of the number of special cases which arise in three-dimensional unstructured grids.

For any of the above extensions, it may be desirable to modify the grid refinement algorithm such that the refinement is directional. For example, in two dimensions, the cells are divided into quarters in the present technique; directional embedded would divide them in halves instead. This extension may be particularly useful for grid adaptation in boundary layer-like regions or for shock sheets in three dimensions. Kallinderis and Baron[50] have discussed the merits of directional adaptation in connection with their adaptive Navier-Stokes scheme.

Another extension involves the combination of grid embedding and grid redistribution. The concept involves using redistribution to align the computational grid with the flow features wherever possible, and to use refinement to achieve the required grid spacing. In this way, the benefits of the two schemes can be employed simultaneously. Adjerid and Flaherty[1] have developed such a scheme for parabolic partial differential equations.

Finally, the feature detector is equally applicable to flow field descriptions which are experimentally generated. Hence it is possible to employ a modification of the feature detector to extract important flow features from fields of experimental data. But since experimental data is not usually defined on a grid, a triangulation technique may be needed to generate a grid on which the feature detector then can be applied.

9.3 Restrictions of the Adaptation Methodology

Throughout the development of the adaptation strategy a number of assumptions were made, some of which put restrictions on the applicability of the scheme to other situations. In this section the two most important limitations are discussed.

First, the technique has been developed for the computation of steady flow fields; the unsteady form of the governing equations is used solely as a device to simplify the solution procedure. Because only the steady solution was sought, various acceleration techniques, most notably the multiple-grid accelerator, were used in the integrator to hasten convergence to the steady state. If these acceleration devices are eliminated, the embedded mesh procedure as described in chapter 4 would no longer function because the multiple-grid accelerator plays a central role in its formulation. Hence although in principal unsteady flows could be computed, new grid coupling procedures would need to be developed.

Second, the semi-structured embedded domain and its associated data structure are ideally suited for explicit integration techniques. In explicit techniques, arbitrarily connected nodes do not cause difficulties because the integration scheme can be written on a cell-by-cell basis. However, application of this semi-structured grid to implicit integration schemes poses a significant challenge because of the unstructured matrix which results. Implicit schemes typically require an underlying structure in the matrix to

Case	refinement cycles	pseudo-CPU time		
		adaptively refined	globally refined	savings factor
AGARD-01	3	2248	13301	5.9
AGARD-02	4	7594	61086	8.0
AGARD-03	3	2839	20420	7.2
AGARD-04	3	5326	23476	4.4
AGARD-05	3	2126	12874	6.1
AGARD-06	4	3300	56907	17.2
NLR-26	-	–	–	–
Biplane	4	20626	262236	12.7

Table 9.1: Summary of CPU savings achieved through grid adaptation

solve it efficiently, whether by factoring or splitting. Hence a significant effort would be required to use the current unstructured meshes in an implicit scheme.

9.4 Major Conclusions

Grid adaptation by local grid embedding is an effective technique for obtaining accurate solutions.

The adaptive grid embedding technique developed here has been applied to a variety of transonic test cases. These comprise a number of geometric and flow-field topologies, including nearly normal and oblique shock waves. In each case, the solution generated with the adaptively embedded regions produced results which were within 1.5% of the solution obtained with global refinement.

Tables 9.1 and 9.2 contain a summary of the required CPU-time and

Case	refinement cycles	storage		
		adaptively refined	globally refined	savings factor
AGARD-01	3	3157	33024	10.5
AGARD-02	4	7746	131584	17.0
AGARD-03	3	3092	33024	10.7
AGARD-04	3	4251	33024	7.8
AGARD-05	3	2871	33024	11.5
AGARD-06	4	3251	131584	40.5
NLR-26	-	-	-	-
Biplane	4	21187	579328	27.3

Table 9.2: Summary of storage savings achieved through grid adaptation

storage (number of nodes) for each of the test cases described in chapter 8, both for an adaptively and globally refined grids. The number of cycles of refinement required for each case is based on the requirement that the lift and drag coefficients agree on successive grid to within 0.010 and 0.002 respectively.

The Tables also contain a *savings factor*, that is the ratio of the requirements for the globally refined solution to those of the adaptive grid solution. The savings which result from using adaptation are on the order of ten for CPU-time and approximately twenty for storage. (Note however that the data structure required for semi-structured mesh calculations requires about twice as much storage as typical i, j indexing, so that the effective savings for storage is only about ten.)

The numbers in these tables are highly dependent on the acceptable difference in force coefficients on successive grids. If such tolerances are made

more strict, the number of required adaptation cycles increases, resulting in larger savings. This effect can be seen in the Tables by noting that the solutions with the largest savings factors are those which required the largest number of adaptation cycles.

The hybrid system approach developed herein provides a technique for handling the complexity associated with large, complex computer systems.

Grid adaptation strategies are by their nature heuristic, and hence no single adaptation strategy should be expected to be universally applicable. With this in mind, there needed to be a mechanism through which the strategy could be easily modified as new adaptation rules became known. The mechanism through which this was accomplished was an expert system.

Expert systems are ideally suited to handling heuristic information such as strategy statements but not as well suited to large volumes of numerical data. On the other hand, procedural systems (such as FORTRAN) are well suited to numerical calculations but do not handle heuristics well. The hybrid system approach developed here enables one to treat both types of information simultaneously in one system.

The flexibility which the hybrid system offers is well demonstrated in chapter 8, where the original adaptation strategy needed to be modified in order to handle more complicated cases. The rules which were added to the knowledge base in order to change the strategy were implemented in minutes, without need to change the underlying program.

Also, because the strategy is contained in a knowledge base (which is separate from the program), a single program can be used with different strategies (knowledge bases) for different problems. This makes the program more flexible and applicable to a wider range of problems.

Irregularly shaped grids offer an effective means of computing flows over complex geometric topologies.

The final two sections of chapter 8 showed the computation of flows over complex geometric topologies. The initial grid in both cases was of an H-type topology, with pre-embedding used around the airfoil. The resulting grids, which were not logically rectangular, were treated in the same manner as embedded grids in adaptive computations. The errors introduced at the leading edges by the H-type topology could easily be controlled through local mesh refinement.

The implication is that with semi-structured grids as developed here, and with adaptive grid techniques, flow field evaluations for many types of configurations are possible. This has even more important implications in three dimensions, where the generation of structured grids is nearly impossible for many geometries, as discussed by Atta[8].

Bibliography

[1] S. Adjerid and J. E. Flaherty. A moving-mesh finite element method with local refinement for parabolic partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 55:3–26, 1986.

[2] AGARD Subcommittee C. *Test Cases for Inviscid Flow Field Methods.* AGARD Advisory Report 211, Advisory Group for Aerospace Research & Development, 1986??

[3] AGARD Subcommittee C. *Test Cases for Steady Inviscid Transonic or Supersonic Flows.* AGARD FDP WG-07, Advisory Group for Aerospace Research & Development, January 1983.

[4] D. A. Anderson and N. Rajendran. *Two Approaches Toward Generating Orthogonal Adaptive Grids.* AIAA-84-1610, American Institute of Aeronautics and Astronautics, June 1984.

[5] D. A. Anderson and J. Steinbrenner. *Generating Adaptive Grids with a Conventional Grid Scheme.* AIAA-86-0427, American Institute of Aeronautics and Astronautics, January 1986.

[6] A. E. Andrews. *Progress and Challenges in the Application of Artificial Intelligence to Computational Fluid Dynamics.* AIAA-87-0593, American Institute of Aeronautics and Astronautics, January 1987.

[7] D. C. Arney and J. E. Flaherty. *A Two-Dimensional Mesh Moving*

Technique for Time Dependent Partial Differential Equations. Technical Report 85-9, Department of Computer Science, Rensselaer Polytechnic Institute, 1985.

[8] E. H. Atta, L. Birckelbaw, and K. A. Hall. *A Zonal Grid Generation Method for Complex Configurations.* AIAA-87-0276, American Institute of Aeronautics and Astronautics, January 1987.

[9] I. Babuska and W. C. Rheinbolt. Error estimates for adaptive finite element computations. *SIAM Journal of Numerical Analysis*, 15(4):736–754, August 1978.

[10] M. J. Berger. *Adaptive Finite Difference Methods in Fluid Dynamics.* DOE/ER/03077-277, Courant Mathematics and Computing Laboratory, New York University, September 1984.

[11] M. J. Berger. *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations.* STAN-CS-82-924, Department of Computer Science, Stanford University, August 1982.

[12] M. J. Berger. *Stability of Interfaces with Mesh Refinement.* Report No. 83-42, Institute for Computer Applications in Science and Engineering, August 1983.

[13] J. W. Boerstoel. *Combining Characteristic Forms of Boundary Conditions and Conservation Equations at Boundaries of Cell-Centered Euler-Flow Calculations.* NLR MP 87024 U, National Aerospace Laboratory NLR, The Netherlands, 1987.

[14] A. Brandt. Multilevel adaptive computations in fluid dynamics. *AIAA Journal*, 18(10):1165–1172, October 1980.

[15] J. J. Brown. *A Multigrid Mesh-Embedding Technique for Three-Dimensional Transonic Potential Flow Analysis.* AIAA-82-0107,

American Institute of Aeronautics and Astronautics, January 1982.

[16] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5.* Addison-Wesley Publishing, Co., Reading, MA, 1986.

[17] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiment of the Stanford Hueristic Programming Project.* Addison-Wesley Publishing, Co., Reading, MA, 1984.

[18] R. Carcaillet, G. S. Dulikravich, and S. R. Kennon. Generation of solution-adaptive computational grids using optimization. *Computer Methods in Applied Mechanics and Engineering,* 57:279–295, 1986.

[19] J. E. Carter. *A New Boundary-Layer Inviscid Iteration Technique for Separated Flows.* AIAA-79-1450, American Institute of Aeronautics and Astronautics, July 1979.

[20] J. E. Carter, D. E. Edwards, and M. J. Werle. Coordinate transformation for laminar and turbulent boundary layers. *AIAA Journal,* 20(2):282–284, February 1982.

[21] S. C. Caruso, J. H. Ferziger, and J. Oliger. *Adaptive Grid Techniques for Elliptic Fluid Flow Problems.* AIAA-86-0498, American Institute of Aeronautics and Astronautics, January 1986.

[22] S. R. Chakravarthy. *Euler Equations — Implicit Schemes and Implicit Boundary Conditions.* AIAA-82-0228, American Institute of Aeronautics and Astronautics, January 1982.

[23] P. Cheeseman and W. Gevarter. *Introduction to Artificial Intelligence.* AIAA-86-0163, American Institute of Aeronautics and Astronautics, January 1986.

[24] J. D. Cole and L. P. Cook. *Transonic Aerodynamics.* North-Holland, 1986.

[25] A. Collins and M. R. Quillian. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8:240–247, 1969.

[26] R. S. Conner and D. J. Purdon. *PAN AIR Knowledge System.* AIAA-86-0239, American Institute of Aeronautics and Astronautics, January 1986.

[27] W. C. Connett, R. K. Agarwal, and A. L. Schwartz. *An Adaptive Grid-Generation Scheme for Flowfield Calculations.* AIAA-87-0199, American Institute of Aeronautics and Astronautics, January 1987.

[28] G. Dahlquist, A. Björck, and N. Anderson. *Numerical Methods.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

[29] R. L. Davis. Personal communication.

[30] M. Drela. Personal communication.

[31] M. Drela. *Two-Dimensional Transonic Aerodynamic Design and Analysis Using the Euler Equations.* PhD thesis, Massachusetts Institute of Technology, December 1985.

[32] H. A. Dwyer. *A Discussion of Some Criteria for the Use of Adaptive Gridding.* AIAA-83-1932, American Institute of Aeronautics and Astronautics, July 1983.

[33] H. A. Dwyer, R. J. Kee, and B. R. Sanders. Adaptive grid method for problems in fluid mechanics and heat transfer. *AIAA Journal*, 18(10):1205–1212, October 1980.

[34] A. Ecer and H. U. Akay. *Investigation of Transonic Flow in a Cascade Using an Adaptive Mesh.* AIAA-80-1430, American Institute of Aeronautics and Astronautics, July 1980.

[35] P. R. Eiseman. Adaptive grid generation by mean value relaxation. *Journal of Fluids Engineering*, 107:477–483, December 1985.

[36] L. Fuchs. *Adaptive Construction of Grid-Systems for Flow Simulations.* , Royal Institute of Technology, Stockholm, Sweden, 1986.

[37] P. R. Garabedian. Computation of wave drag for transonic flow. *Journal D'Analyse Mathematique*, 30:164–171, 1976.

[38] N. K. Ghia, U. Ghia, and C. T. Shin. Adaptive grid generation for flows with local high gradient regions. *Advances in Grid Generation, AMSE*, 5:35–47, February 1983.

[39] M. B. Giles. *Eigenmode Analysis of Unsteady One-Dimensional Euler Equations.* NASA CR 172217, Institute for Computer Applications in Science and Engineering, August 1983.

[40] A. Goodall. *The Guide to Expert Systems.* Learned Information, Oxford and New York, 1985.

[41] M. G. Hall. *Cell-Vertex Multigrid Schemes for Solution of the Euler Equations.* Tech Memo Aero 2029, Royal Aircraft Establishment, March 1985.

[42] P. Harmon and D. King. *Expert Systems — Artificial Intelligence in Business.* John Wiley and Sons, Inc., New York, 1985.

[43] A. Harten and J. M. Hyman. Self adjusting grid methods for one-dimensional hyperbolic conservation laws. *Journal of Computational Physics*, 50:235–269, 1983.

[44] D. F. Hawken. *Review of Adaptive-Grid Techniques for Solution of Partial Differential Equations.* UTIAS Review No. 46, Institute for Aerospace Studies, December 1985.

[45] F. B. Hildebrand. *Advanced Calculus for Applications.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1962.

[46] R. G. Hindman and J. Spencer. *A New Approach to Truly Adaptive Grid Generation.* AIAA-83-0450, American Institute of Aeronautics and Astronautics, January 1983.

[47] J. E. Holcomb and R. G. Hindman. *Development of a Dynamically Adaptive Grid Method for Multidimensional Problems.* AIAA-84-1668, American Institute of Aeronautics and Astronautics, June 1984.

[48] T. L. Holst and D. Brown. Transonic airfoil calculations using solution-adaptive grids. *AIAA Journal*, 21(2):304–306, February 1983.

[49] A. Jameson, T. J. Baker, and N. P. Weatherill. *Calculation of Inviscid Transonic Flow over a Complete Aircraft.* AIAA-86-0103, American Institute of Aeronautics and Astronautics, January 1986.

[50] J. G. Kallinderis and J. R. Baron. *Adaptation Methods for a New Navier-Stokes Algorithm.* AIAA-87-1167-CP, American Institute of Aeronautics and Astronautics, June 1987.

[51] P. Kutler and U. Mehta. *Computational Aerodynamics and Artificial Intelligence.* AIAA-84-1531, American Institute of Aeronautics and Astronautics, June 1984.

[52] P. Lax. On the stability of difference approximations to solutions of hyperbolic equations with variable coefficients. *Communications on Pure and Applied Mathematics*, 14:497–520, 1961.

[53] P. Lax and B. Wendroff. Difference schemes for hyperbolic equations with high order of accuracy. *Communications on Pure and Applied Mathematics*, 17:381–398, 1964.

[54] P. Lax and B. Wendroff. On the stability of difference schemes. *Communications on Pure and Applied Mathematics*, 15:363–371, 1962.

[55] P. Lax and B. Wendroff. Systems of conservation laws. *Communications on Pure and Applied Mathematics*, 13:217–237, 1960.

[56] R. Löhner, K. Morgan, and O. C. Zienkiewicz. Adaptive grid refinement for the compressible euler equations. In *Accuracy Estimates and Adaptivity for Finite Elements*, Wiley, 1984.

[57] R. Löhner, K. Morgan, and O. C. Zienkiewicz. *Adaptive Grid Refinement for the Euler and Compressible Navier Stokes Equations.* C/R/470/83, Institute for Numerical Methods in Engineering, December 1983.

[58] G. S. S. Ludford. The behavior at infinity of the potential function of a two-dimensional subsonic compressible flow. *Journal of Methematics and Physics*, 30:117–130, 1951.

[59] D. L. Marcum and J. D. Hoffman. *Numerical Boundary Condition Procedures for Unsteady Euler Solvers.* AIAA-86-0107, American Institute of Aeronautics and Astronautics, January 1986.

[60] C. W. Mastin. Errors introduced by coordinate systems. In J. F. Thompson, editor, *Numerical Grid Generation*, pages 31–40, Elsevier Science Publishing Co., 1982.

[61] C. W. Mastin and J. F. Thompson. *Adaptive Grids Generated by Elliptic Systems.* AIAA-83-0451, American Institute of Aeronautics and Astronautics, January 1983.

[62] R. E. Melnik, R. Chow, and H. R. Mead. *Theory of Viscous Transonic Flow Over Airfoils at Hign Reynolds Number.* AIAA-77-0680, American Institute of Aeronautics and Astronautics, June 1977.

[63] G. Moretti. *Experiments in Multi-Dimensional Floating Shock-Fitting.* PIBAL Report 73-18, August 1973.

[64] K. Nakahashi and G. S. Deiwert. *A Practical Adaptive-Grid Method for Complex Fluid-Flow Problems.* TM 85989, National Aeronautics and Space Administration, June 1984.

[65] A. Newell and H. Simon. *Human Problem Solving.* Prentice Hall, Englewood Cliffs, NJ, 1972.

[66] R.H. Ni. *Multigrid Convergence Acceleration Techniques for Explicit Euler Solvers and Applications to Navier- Stokes Calculations.* , Von Karman Institute, 1986.

[67] R-H. Ni. A multiple-grid scheme for solving the Euler equations. *AIAA Journal,* 20(11):1565–1571, November 1982.

[68] R-H. Ni. Personal communication.

[69] K. G. Powell. *Vortical Solutions of the Conical Euler Equations.* PhD thesis, Massachusetts Institute of Technology, June 1987.

[70] T. H. Pulliam. *Artificial Dissipation Models for the Euler Equations.* AIAA-85-0438, American Institute of Aeronautics and Astronautics, January 1985.

[71] T. H. Pulliam and J. T. Barton. *Euler Computations of AGARD Working Group 07 Airfoil Test Cases.* AIAA-85-0018, American Institute of Aeronautics and Astronautics, January 1985.

[72] M. M. Rai and D. A. Anderson. *Application of Adaptive Grids to Fluid Flow Problems with Asymptotic Solutions.* AIAA-81-0114, American Institute of Aeronautics and Astronautics, January 1981.

[73] R. D. Richtmyer and K. W. Mortin. *Difference Methods for Initial-Value Problems.* Interscience Publishers, New York, 1967.

[74] P. J. Roache. *Computational Fluid Dynamics.* Hermosa Publishers, Albuquerque, NM, 1982.

[75] J. Saltzman and J. Brackbill. Applications and generalizations of variational methods for generating adaptive meshes. In J. F. Thompson, editor, *Numerical Grid Generation*, pages 865–884, Elsevier Science Publishing Co., 1982.

[76] W. Schmidt, A. Jameson, and D. Whitfield. *Finite Volume Solution for the Euler Equation for Transonic Flow over Airfoils and Wings Including Viscous Effects.* AIAA-81-1265, American Institute of Aeronautics and Astronautics, June 1981.

[77] R. L. Sorenson. *A Computer Program to Generate Two-Dimensional Grids About Airfoils and Other Shapes by the Use of Poisson's Equation.* TM 81198, National Aeronautics and Space Administration, May 1980.

[78] J. L. Steger and R. L. Sorenson. Automatic mesh-point clustering near a boundary in grid generation with elliptic partial differential equations. *Journal of Computational Physics*, 33:405–410, 1979.

[79] J. L. Thomas and M. D. Salas. Far-field boundary conditions for transonic lifting solutions to the euler equations. *AIAA Journal*, 24(7):1074–1080, July 1986.

[80] J. F. Thompson. *A Survey of Dynamically-Adaptive Grids in the Numerical Solution of Partial Differential Equations.* AIAA-84-1606, American Institute of Aeronautics and Astronautics, June 1984.

[81] J. F. Thompson, F. C. Thames, and C. W. Mastin. Automatic numerical generation of body-fitted curvilinear coordinate system for field

containing any number of arbitrary two-dimensional bodies. *Journal of Computational Physics*, 15:299–319, 1974.

[82] S. S. Tong. *Coupling Artificial Intelligence and Numerical Computation for Engineering Design.* AIAA-86-0242, American Institute of Aeronautics and Astronautics, January 1986.

[83] S. S. Tong. *Design of Aerodynamic Bodies Using Artificial Intelligence/Expert System Technique.* AIAA-85-0112, American Institute of Aeronautics and Astronautics, January 1985.

[84] W. J. Usab. *Embedded Mesh Solutions of the Euler Equation Using a Multiple-Grid Method.* PhD thesis, Massachusetts Institute of Technology, December 1983.

[85] W. J. Usab and E. M. Murman. Embedded mesh solutions of the Euler equation using a multi-grid method. In W. G. Habashi, editor, *Advances in Computational Transonics*, pages 447–472, Pineridge Press, 1985.

[86] K. Ushimaru. *Development and Application of Adaptive Grids in Two-Dimensional Transonic Calculations.* AIAA-82-1016, American Institute of Aeronautics and Astronautics, June 1982.

[87] B. van den Berg. *Boundary Layer Measurements on a Two-Dimensional Wing with Flap.* NLR TR 79009 U, National Aerospace Laboratory NLR, The Netherlands, January 1979.

[88] P. H. Winston. *Artificial Intelligence.* Addison-Wesley Publishing, Co., Reading, MA, 1984.

[89] P. H. Winston and K. A. Prendergast, editors. *XCON: An Expert Configuration System at Digital Equipment Corp. AI Business: The*

Commercial Uses of Artificial Intelligence, MIT Press, Cambridge, MA, 1984.

[90] F. W. Wubs, J. W. Boerstoel, and A. J. van der Wees. Grid-size reduction in flow calculations on infinite domains by higher-order far-field asymptotics in numerical boundary conditions. *Journal of Engineering Mathematics*, 18:157–177, 1984.

Appendix A

MITOSIS User's Manual

This appendix contains information needed to execute program MITOSIS. It begins with a brief discussion of the architecture of the MITOSIS program. The second section contains a description of each of the input/output units which program uses. The appendix finishes with a description of the general input file which MITOSIS uses.

A.1 Program Hierarchy

Program MITOSIS is broken into seven major component, given by:

MITOSIS The main program.

EXPROC The expert system.

EULR2D The two-dimensional Euler integration scheme.

FEAT2D The two-dimensional feature detection routines.

GRID2D The two-dimensional grid and data structure handling routines.

UTILITY Utility sunbroutines, such as a spline fit, etc.

GRAFIC The graphics package.

The program is hierarchical in design, that is the design is such that some of the components are built upon the functions performed in other components. The hierarchy is shown in figure A.1, where a subroutine in any component may call only those subroutines in its own component or one below it in the figure. This design allows easy changes to the integration technique, flow model, etc.

A.2 Input/Output Units

Program MITOSIS uses a number of files during its execution, each of which is described below. Only the files for unit numbers 003 and 008 (as well as the terminal for interactive runs) are needed to execute MITOSIS.

unit=002 An unformatted file which MITOSIS uses as a scratch file to which the neighbor table is written. This unit number is actually chosen by the knowledge base attributes `nbor_in` and `nbor_out`.

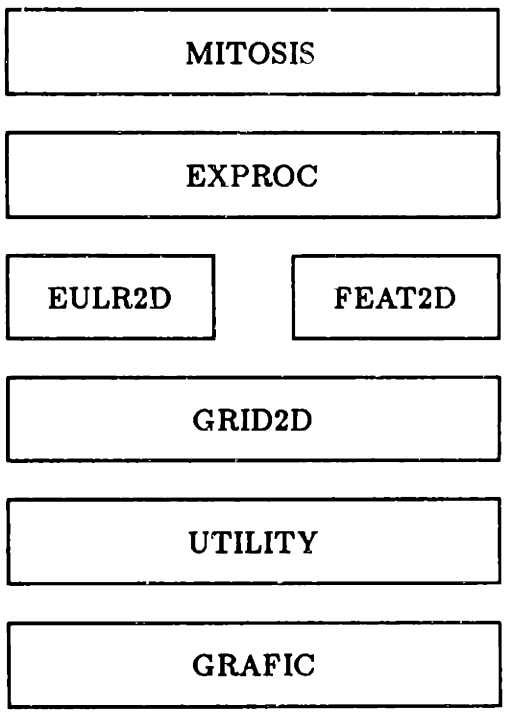


Figure A.1: Hiearachy of program MITOSIS

unit=003 The knowledge base file (which is formatted according to the rules in Appendix D).

unit=004 A formatted file which contains the stop flag. The format of the one record in this file is (I5). If the stop flag is set to 0, then program execution continues as usual. Other settings of the stop flag are defined in the knowledge base.

unit=005 Terminal input.

unit=006 Terminal output.

unit=007 The listing file generated by the program. It contains a copy of the input file, a compiled listing of the knowledge base, the convergence histories, as well as any other status information printed by the program.

unit=008 The general input file, described below.

unit=010 A formatted file which contains comparison information which is plotted along with the surface distrbibutions if required.

unit=012 A formatted file which contains the convergence history information. This file is rewound and read when a convergence history plot is required.

units=013 through 020 Unformatted files which contain a complete dump of the data base and solution each time the program converges.

A.3 General Input File Description

MITOSIS Basic input – card type 1 1 required		
VARIABLE	FORMAT	DESCRIPTION
TITLE	A80	Case title. This title will appear on the top of all printed tables as well as all plots.

MITOSIS Basic input – card type 2 optional – any number may appear		
VARIABLE	FORMAT	DESCRIPTION
*	A1	A star (*) in column 1 signifies a comment line.
COMMNT	A79	A free-form comment which will appear in the print file

MITOSIS Basic input – card type 3 1 required		
VARIABLE	FORMAT	DESCRIPTION
IBATCH	I5	Run type indicator. IBATCH = 0 indicates an interactive run IBATCH = 1 indicates a batch run (no terminal interaction) IBATCH = 2 indicates an interactive run, but the interactive commands are contained at the end of the Basic input file

MITOSIS Basic input – card type 4 1 required		
VARIABLE	FORMAT	DESCRIPTION
U1RFE2	F10	Reference density. Recommended value = 1.00.
U2RFE2	F10	Reference x-velocity. Recommended value = $\cos(\alpha)$ where α is the free–stream angle of attack.
U3RFE2	F10	Reference y-velocity. Recommended value = $\sin(\alpha)$.
U4RFE2	F10	Reference density. Recommended value = $0.50 + \frac{1}{\texttt{GM00E2}(\texttt{GM00E2}-1)M^2}$ where M is the free–stream Mach number.
CLFTE2	F10	Initial guess for the lift coefficient. This value is used in the first iteration to set the vortex strength in the far–field representation. It is not needed if CHRDE2 = 0.
CDRGE2	F10	Initial guess for the drag coefficient. (Currently not used)
CHRDE2	F10	The chord–length used in the definitions of lift– and drag–coefficients. If CHRDE2 = 0, then there is no vortex term in the far–field representation.

MITOSIS Basic input – card type 5 1 required		
VARIABLE	FORMAT	DESCRIPTION
AMUE2(1)	F10	Smoothing equation relaxation parameter. Recommended value = 0.100.
AMUE2(2)	F10	Smoothing equation smoothness parameter. Recommended value = 0.500.
AMUE2(3)	F10	Background smoothing level. Recommended value = 0.0250.
AMUE2(4)	F10	Smoothing weighting factor. Recommended value = 200.0.
GMOOE2	F10	Ratio of specific heats.
CFLE2	F10	CFL number for Euler integration. Recommended value = 0.95.
NWNDE2	I5	Window size for convergence checker. This value should be large enough so that at least one period of any oscillations in the lift– and drag–coefficient curves (as a function of iteration) is contained within the window. For airfoils with a base grid of 33 points around the airfoil, an acceptable value is 30.

MITOSIS Basic input – card type 6 1 required		
VARIABLE	FORMAT	DESCRIPTION
ALFAF2	**F10**	Divide threshold set at the largest point where $\frac{df}{dT}$ > **ALFAF2** where f is the cummulative refinement function and T is the threshold. Recommended value = −0.20.
BETAF2	**F10**	Smallest permissible value of the divide threshold. Recommended value = 1.25.
GAMAF2	**F10**	The divide threshold is adjusted such that the fraction of nodes above the threshold cannot exceed **GAMAF2**. Recommended value = 0.250.
DLTAF2	**F10**	The value for the fusion threshold. Recommended value = 0.50..
NSNGF2	**I5**	The number of refinement singularities. After the refinement parameters at each node and the divide and fusion thresholds are computed, the refinement parameter in the vicinity of the refinement singularities is artificially increased so as to force refinement near these points. Typically these are set up in the vicinity of leading and trailing edges.

MITOSIS Basic input – card type 7 NSNGF2 required		
VARIABLE	FORMAT	DESCRIPTION
XSNGF2(i)	F10	The x-location of the ith refinement singularity.
YSNGF2(i)	F10	The y-location of the ith refinement singularity.
RSNGF2(i)	F10	The radius of the ith refinement singularity.

MITOSIS Basic input – card type 8 1 required		
VARIABLE	FORMAT	DESCRIPTION
IMESH	I5	Mesh type indicator. IMESH = 1 for a simply–connected rectangle. IMESH = 2 for an O–mesh around a one–part airfoil. IMESH = 3 (currently not supported). IMESH = 4 for a H–mesh around a one–part airfoil. IMESH = 5 for a H–mesh around a two–part airfoil. IMESH = 6 for a H–mesh around a bi–plane.

MITOSIS Basic input – card type 9 1 required if IMESH = 1		
VARIABLE	FORMAT	DESCRIPTION
NXNODE	I5	Number of nodes along the south and north sides of the domain. Note that NXNODE − 1 must be evenly divisible by 2^{NCOARS}.
NYNODE	I5	Number of nodes along the east and west sides of the domain. Note that NYNODE − 1 must be evenly divisible by 2^{NCOARS}.
KDEFG2	I5	The grid level at which the sides of the domain are defined. The finest global grid is the level-0 grid, the next finer is the level-1 grid, etc. This option allows the user to define the edges of the domain precisely, even though such precision is not required for the level-0 grid.
NITER	I5	The number of iteration that the Laplace grid smoother is allowed to take. Recommended value = 500.
NCOARS	I5	The number of multipe-grid levels below the fine global (level-0) grid.
IBCSW	I5	Boundary condition type for the southwest corner of the domain. IBCSW = 1 for radiation condition. IBCSW = 2 for Dirichlet condition. IBCSW = 5 for corner with solid wall along south side. IBCSW = 6 for corner with solid wall along west side.

IBCS	I5	Boundary condition type for the south side of the domain. IBCS = 1 for radiation condition. IBCS = 2 for Dirichlet condition. IBCS = 3 for solid wall. IBCS = 4 for free–stream condition.
IBCSE	I5	Boundary condition type for the southeast corner of the domain. See description of IBCSW for more details.
IBCE	I5	Boundary condition type for the east side of the domain. See desription of IBCS for more details.
IBCNE	I5	Boundary condition type for the northeast corner of the domain. See description of IBCSW for more details.
IBCN	I5	Boundary condition type for the north side of the domain. See desription of IBCS for more details.
IBCNW	I5	Boundary condition type for the northwest corner of the domain. See description of IBCSW for more details.
IBCW	I5	Boundary condition type for the west side of the domain. See desription of IBCS for more details.

MITOSIS Basic input – card type 10

$1 + (\texttt{NXNODE} - 1) * 2^{\texttt{KDEFG2}}$ required if IMESH = 1

Description of the south boundary points, going from west to east

VARIABLE	FORMAT	DESCRIPTION
BODYG2(1)	F10	x-coordinate.
BODYG2(2)	F10	y-coordinate.
BODYG2(3)	F10	Initial value for density.
BODYG2(4)	F10	Initial value for x-velocity.
BODYG2(5)	F10	Initial value for y-velocity.
BODYG2(6)	F10	Initial value for energy.
BODYG2(7)	F10	Initial value for smoothing coefficient.

MITOSIS Basic input – card type 11

$1 + (\texttt{NXNODE} - 1) * 2^{\texttt{KDEFG2}}$ required if IMESH = 1

Description of the north boundary points, going from west to east

VARIABLE	FORMAT	DESCRIPTION
BODYG2(1)	F10	x-coordinate.
BODYG2(2)	F10	y-coordinate.
BODYG2(3)	F10	Initial value for density.
BODYG2(4)	F10	Initial value for x-velocity.
BODYG2(5)	F10	Initial value for y-velocity.
BODYG2(6)	F10	Initial value for energy.
BODYG2(7)	F10	Initial value for smoothing coefficient.

MITOSIS Basic input – card type 12

$1 + (\texttt{NYNODE} - 1) * 2^{\texttt{KDEFG2}}$ required if IMESH = 1

Description of the west boundary points, going from south to north

VARIABLE	FORMAT	DESCRIPTION
BODYG2(1)	F10	x-coordinate.
BODYG2(2)	F10	y-coordinate.
BODYG2(3)	F10	Initial value for density.
BODYG2(4)	F10	Initial value for x-velocity.
BODYG2(5)	F10	Initial value for y-velocity.
BODYG2(6)	F10	Initial value for energy.
BODYG2(7)	F10	Initial value for smoothing coefficient.

MITOSIS Basic input – card type 13

$1 + (\texttt{NYNODE} - 1) * 2^{\texttt{KDEFG2}}$ required if IMESH = 1

Description of the east boundary points, going from south to north

VARIABLE	FORMAT	DESCRIPTION
BODYG2(1)	F10	x-coordinate.
BODYG2(2)	F10	y-coordinate.
BODYG2(3)	F10	Initial value for density.
BODYG2(4)	F10	Initial value for x-velocity.
BODYG2(5)	F10	Initial value for y-velocity.
BODYG2(6)	F10	Initial value for energy.
BODYG2(7)	F10	Initial value for smoothing coefficient.

MITOSIS Basic input – card type 14 1 required if IMESH = 2		
VARIABLE	FORMAT	DESCRIPTION
NAIRFL	I5	Number of nodes along airfoil surface on the fine global (level-0) grid. Note that NAIRFL–1 must be evenly divisible by 2^{NCOARS}.
KDEFG2	I5	The grid level at which the sides of the domain are defined. The finest global grid is the level-0 grid, the next finer is the level-1 grid, etc. This option allows the user to define the edges of the domain precisely, even though such precision is not required for the level-0 grid.
NITER	I5	The number of iteration that the Laplace grid smoother is allowed to take. Recommended value = 500.
NCOARS	I5	The number of multipe-grid levels below the fine global (level-0) grid.
NSURF	I5	The number of levels of pre–embedding to be performed adjacent to the airfoil surface.
NGROW1	I5	The number of cells that the pre–embedded regions should grown.
NGROW2	I5	If NGROW2 = 0, the growth decribed above is in terms of level-0 cells, thereby growing all the pre–embedded regions the same amount. If NGROW2 = 1, each level only grows NGROW1 cells.

MITOSIS Basic input - card type 15 1 required if IMESH = 2		
VARIABLE	FORMAT	DESCRIPTION
DPEN1	F10	The initial value for the first dependent variable, density. Recommended value = U1RFE2.
DPEN2	F10	The initial value for the second dependent variable, x-velocity. Recommended value = U2RFE2.
DPEN3	F10	The initial value for the third dependent variable, y-velocity. Recommended value = U3RFE2.
DPEN4	F10	The initial value for the fourth dependent variable, energy. Recommended value = U4RFE2.
DPEN5	F10	The initial value for the fifth dependent variable, the smoothing coefficient. Recommended value = AMUE2(3).

MITOSIS Basic input - card type 16 1 required if IMESH = 2		
VARIABLE	FORMAT	DESCRIPTION
RADIUS	F10	The radius of the far-field boundary. Recommended value = 10.
NYNODE	I5	The number of circumferential lines in the level-0 grid. Note that NYNODE − 1 must be evenly divisible by 2^{NCOARS}. Recommended value = (NAIRFL − 1)/2 + 1.

MITOSIS Basic input – card type 17 $1 + (\texttt{NAIRFL} - 1) * 2^{\texttt{KDEFG2}}$ required if IMESH = 2 Description of the airfoil, clockwise from the trailing edge.		
VARIABLE	FORMAT	DESCRIPTION
BODYG2(1)	F10	x-coordinate.
BODYG2(2)	F10	y-coordinate.

Similar inputs for cases with IMESH greater than 1 can be deduced from the program listing of routines G2INP2 through G2INP7.

MITOSIS Basic input – card type 98 1 required if IBATCH=2 used to indicate the beginning of the interactive commands which are included in the input file		
VARIABLE	FORMAT	DESCRIPTION
INDIC	A12	the character string **BATCH IN**

MITOSIS Basic input – card type 99 required if IBATCH=2 interactive commands. See main program for a complete description		
VARIABLE	FORMAT	DESCRIPTION
command		an interactive command

Appendix B

Listing of MITOSIS Program

This Appendix contains a complete listing of the MITOSIS computer program. The Appendix is broken into seven sections, one for each of the major program units within MITOSIS.

The program is written in FORTRAN-77 as implemented on the Digital Equipment Corporation VAX computers. All calls to VAX-specific routines (for example system services) are contained in routines listed in the UTILITY and GRAFIC sections. In addition, there are some primitive plotting calls in GRAFIC, the names and calling sequences of which which are loosely patterned after the calling sequences in CALCOMP graphics.

B.1 MITOSIS — Main Program

```
      PROGRAM MITOSIS
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS PROGRAM COMPUTES THE TWO-DIMENSIONAL, STEADY-STATE SOLUTION
C       OF THE EULER EQUATIONS, USING AN ADAPTIVE GRID ALGORITHM.  THE
C       BASIC SCHEME IS THE EXPLICIT, LAX-WENDROFF SCHEME AS PROPOSED
C       BY NI, EMPLOYING NI'S MULTIPLE-GRID ACCELERATION.  THE EMBEDDED
C       MESH STRATEGY IS SIMILAR TO THE ONE DEVELOPED BY USAB.  THE
C       EMBEDDED MESH IS GENERATED ADAPTIVELY , WITH CONTROL BY A
C       FORWARD-CHAINING EXPERT SYSTEM.
C
C*************************************************************************
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
      CHARACTER IFLAG*1,LABL*10
      CHARACTER INDIC*12,GOAL*10
C
C*************************************************************************
C
C*****INITIALIZE
C
C     INITIALIZE INPUT/OUTPUT UNITS AND UTILITY LIBRARY
C
      CALL UTINIT(1)
C
C     INITIALIZE THE EXPERT SYSTEM AND PRINT OUT A LISTING
C
      CALL EXREAD(     3)
      CALL EXLIST(JPRINT)
C
C     READ TYE BATCH INDICATOR AND KNOWLEDGE BASE UPDATES IF ANY
C
4     READ(JCARDS,5) IFLAG,LABL,VALUE
5     FORMAT(A1,A10,F10.0)
C
      IF(IFLAG.EQ.'$') THEN
         CALL EXEDIT(LABL,VALUE)
         GOTO 4
      ELSE
         READ(LABL,6) IBATCH
6        FORMAT(I4)
      ENDIF
C
C     READ AERODYNAMIC INPUTS AND EULER CONTROLLING PARAMETERS
C
      CALL E2INIT(0)
C
```

```
C     INITIALIZE THE FEATURE FINDER
C
        CALL F2INIT(0)
C
C     READ THE GEOMETRY AND INITIAL CONDITIONS
C
        CALL G2READ(JCARDS,JPRINT)
C
        CALL UTWTIM('INITIALIZATION COMPLETED')
C
C     IF IBATCH=1, THEN THIS IS A BATCH RUN
C
        IF(IBATCH.EQ.1) GOTO 100
C
C     IF IBATCH=2, THEN INTERACTIVE COMMANDS ARE STACKED AT THE
C        END OF THE INPUT FILE, SO ALTER THE TERMINAL INPUT
C        UNIT AND ADVANCE THE INPUT FILE UNTIL IT GETS TO END
C
        IF(IBATCH.EQ.2) THEN
           IBATCH=0
           JTERMI=JCARDS
C
           OPEN(UNIT=JCARDS,STATUS='OLD')
10         READ(JCARDS,20) INDIC
20         FORMAT(A)
           IF(INDIC.NE.'**BATCH IN**') GOTO 10
        ENDIF
C
C*****MAIN MENU
C
30      WRITE(JTERMO,40)
40      FORMAT(' 1. COMPUTE SOLUTION      '/
     &         ' 2. PLOT SOLUTION         '/
     &         ' 3. DIVIDE CELLS          '/
     &         ' 4. ITERATE ON THIS GRID  '/
     &         ' 5. EXECUTE PRIMITIVES    '/)
C
        CALL UTINPI('MENU SELECTION',IMENU)
        GOTO (999,100,200,300,400,500), (IMENU+1)
        GOTO 30
C
C*****COMPUTE SOLUTION
C
C     SOLVE USING ADAPTATION
C
100     CALL EXFWRD(JPRINT)
C
        IF(IBATCH.EQ.1) THEN
           GOTO 999
        ELSE
           GOTO 30
        ENDIF
C
```

```
C*****PLOT SOLUTION
C
C     INITIALLY SET PLOT OFFSET TO ZERO
C
200      INDGR=0
         JNDGR=0
C
210      WRITE(JTERMO,220)
220      FORMAT(' 1. GRID PLOT           '/
     &          ' 2. VECTOR PLOT         '/
     &          ' 3. SOLUTION PLOTS      '/
     &          ' 4. CONVERGENCE PLOTS   '/
     &          ' 5. FEATURE FINDER PLOTS'/
     &          ' 6. SET PLOT SCALES     '/
     &          ' 7. ALTER PLOT INDICATOR'/)
C
         CALL UTINPI('OPTION',IOPT)
C
C.....RETURN TO MAIN MENU
C
         IF(IOPT.EQ.0) THEN
            GOTO 30
C
C.....GRID PLOT
C
         ELSEIF(IOPT.EQ.1) THEN
            CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
            CALL G2PLOT(1,0,INDGR+JNDGR)
C
C.....VECTOR PLOT
C
         ELSEIF(IOPT.EQ.2) THEN
            CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
            CALL G2PLOT(2,0,INDGR+JNDGR)
C
C.....SOLUTION PLOTS (LINE, CONTOUR, AND SURFACE)
C
         ELSEIF(IOPT.EQ.3) THEN
            WRITE(JTERMO,230)
230         FORMAT('  1. X-LOCATION      31. VELOCITY             '/
     &             '  2. Y-LOCATION      32. FLOW ANGLE           '/
     &             '                     33. MACH NUMBER          '/
     &             ' 11. RHO             34. P/PT                 '/
     &             ' 12. RHO*U           35. P/PREF               '/
     &             ' 13. RHO*V           36. ABS(PT-PTREF)/PTREF  '/
     &             ' 14. RHO*E           37. (S-SREF)/CV          '/
     &             ' 15. SMOOTH          38. -CP                  '/
     &             '                     39. (PT-PTREF)/PTREF     '/)
C
            CALL UTINPI('VARIABLE NUMBER',IVAR)
            IF(IVAR.GT.0) CALL E2DATA(IVAR)
C
            WRITE(JTERMO,240)
```

```
240       FORMAT(' 1. LINE PLOT      '/
     &           ' 2. CONTOUR PLOT   '/
     &           ' 3. DATA-CONTOUR   '/
     &           ' 4. SURFACE        '/
     &           ' 5. SURFACE & DATA'/
     &           ' 6. DIMPLED CONTS  ')
C
          CALL UTINPI('PLOT TYPE',ITYPE)
C
          IF(ITYPE.EQ.1) THEN
             CALL UTINPI('STARTING POINT',IBEG)
             CALL G2PLOT(3,IBEG,JNDGR)
          ELSEIF(ITYPE.EQ.2) THEN
             CALL UTINPI('NUMBER OF CONTOURS',NCONT)
             CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
             CALL G2PLOT(4,NCONT,INDGR+JNDGR)
          ELSEIF(ITYPE.EQ.3) THEN
             CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
             CALL G2PLOT(6,IDUM,INDGR+JNDGR)
          ELSEIF(ITYPE.EQ.4) THEN
             IWALL=3
             CALL G2PLOT(7,IWALL,JNDGR)
          ELSEIF(ITYPE.EQ.5) THEN
             IWALL=-3
             CALL G2PLOT(7,IWALL,JNDGR)
          ELSEIF(ITYPE.EQ.6) THEN
             CALL UTINPI('NUMBER OF CONTOURS',NCONT)
             CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
             CALL G2PLOT(8,NCONT,INDGR+JNDGR)
          ENDIF
C
C.....CONVERGENCE PLOTS
C
       ELSEIF(IOPT.EQ.4) THEN
          WRITE(JTERMO,250)
250       FORMAT(' 1. ITERATION      6. RMS DU2    '/
     &           ' 2. CPU TIME       7. C-LIFT     '/
     &           ' 3. MAX DU2        8. C-DRAG     '/
     &           ' 4. IMAX DU2       9. SMTH (MAX) '/
     &           ' 5. AVG DU2                      '/)
C
          CALL UTINPI('ABSCISSA',IX)
          CALL UTINPI('ORDINATE',IY)
          CALL E2HSPL(IX*10+IY)
C
C.....FEATURE FINDER PLOTS
C
       ELSEIF(IOPT.EQ.5) THEN
          WRITE(JTERMO,260)
260       FORMAT(' 1. THRESHOLDED NODES'/
     &           ' 2. THRESHOLD FINDER '/
     &           ' 3. CHANGE VS INODE  '/
     &           ' 4. HISTOGRAMS       '/)
```

```
          CALL UTINPI('PLOT OPTION',IPLOT)
          IF(IPLOT.EQ.1) CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
          CALL F2PLOT(IPLOT,JNDGR)
C
C.....SET PLOT SCALES
C
        ELSEIF(IOPT.EQ.6) THEN
          CALL UTINPF('XMIN',XMIN)
          CALL UTINPF('XMAX',XMAX)
          CALL UTINPF('YMIN',YMIN)
          CALL UTINPF('YMAX',YMAX)
          IF(XMIN.EQ.XMAX) THEN
            INDGR=0
          ELSE
            INDGR=-1
            CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
          ENDIF
C
C.....ALTER PLOT INDICATOR
C
        ELSEIF(IOPT.EQ.7) THEN
          CALL UTINPI('DELTA-INDGR',JNDGR)
C
        ENDIF
C
        GOTO 210
C
C*****DIVIDE CELLS
C
300     CALL UTINPI('FIRST CELL TO DIVIDE',IBEG)
        CALL UTINPI('LAST  CELL TO DIVIDE',IEND)
        IF (IBEG.LE.0 .OR. IEND.LE.0) GOTO 30
C
                          IINC=+1
        IF(IEND.LT.IBEG) IINC=-1
        DO 310 ICELL=IBEG,IEND,IINC
        CALL G2DIVD(ICELL,ICHANG)
310     CONTINUE
C
C     GARBAGE COLLECT
C
        CALL G2GRBG
C
        GOTO 30
C
C*****ITERATE ON THIS GRID
C
400     CALL UTINPI('INITIALIZE? (0/1)',INIT)
        IF(INIT.EQ.1) THEN
          CALL UTINPI('UNIT NUMBER FOR NEIGHBORS',IUNIT)
          CALL G2NBOR(IUNIT)
C
          CALL UTINPF('DU-TOL',DUTOL)
```

```
            CALL UTINPF('CL-TOL',CLTOL)
            CALL UTINPF('CD-TOL',CDTOL)
            CALL E2CTRL(DUTOL,CLTOL,CDTOL)
         ENDIF
C
         IPASS=0
         CALL UTINPI('MAXIT',MAXIT)
         DO 410 ITER=1,MAXIT
         CALL E2MAIN(DELDU,CLIFT,CDRAG)
         CALL E2WIND(IPASS,ICONV)
         CALL E2HSPR(1)
         IF(ICONV.NE.0) GOTO 420
410      CONTINUE
C
420      GOTO 30
C
C*****EXECUTE PRIMITIVES
C
500      WRITE(JTERMO,510)
510      FORMAT('  1. G2DIVD   21. E2INIT   31. F2INIT   41. EXREAD'/
     &          '  2. G2FUSE   22. E2CTRL   32. F2VALU   42. EXLIST'/
     &          '  3. G2GRBG   23. E2MAIN   33. F2GRAD   43. EXFWRD'/
     &          '  4. G2SUMY   24. E2HSPR   34. F2LAPL   44. EXBACK'/
     &          '  5. G2READ   25. E2DATA   35. F2THRS   45. EXEDIT'/
     &          '  6. G2WRIT   26. E2FORC   36. F2ADPT   46. EXDUMP'/
     &          '  7. G2PRNT   27. E2PRNT   37. F2PRNT             '/
     &          '  8. G2GROW   28. E2STAT   38. F2DATA             '/
     &          '  9. G2SURF   29. E2VRFY                          '/
     &          ' 10. G2VOID   30. E2COMP                          '/
     &          ' 11. G2XPND                                       '/
     &          ' 12. G2ADPT   14. G2SORT   16. G2DATA   18. G2CHEK'/
     &          ' 13. G2EXAM   15. G2NBOR   17. G2INTG             '/)
C
         CALL UTINPI('ROUTINE TO BE EXECUTED',IROUT)
C
         IF(IROUT.EQ.0) THEN
            GOTO 30
C
         ELSEIF(IROUT.EQ. 1) THEN
            CALL UTINPI('CELL TO DIVIDE',ICELL)
            CALL G2DIVD(ICELL,ICHANG)
            WRITE(JTERMO,540) ICHANG
C
         ELSEIF(IROUT.EQ. 2) THEN
            CALL UTINPI('CELL TO FUSE',ICELL)
            CALL G2FUSE(ICELL,ICHANG)
            WRITE(JTERMO,540) ICHANG
C
         ELSEIF(IROUT.EQ. 3) THEN
            CALL G2GRBG
C
         ELSEIF(IROUT.EQ. 4) THEN
            CALL UTINPI('UNIT NUMBER',JUNIT)
```

```
            CALL G2SUMY(JUNIT)
C
         ELSEIF(IROUT.EQ. 5) THEN
            CALL UTINPI('INPUT  UNIT NUMBER',JUNIT)
            CALL G2READ(JUNIT,0)
C
         ELSEIF(IROUT.EQ. 6) THEN
            CALL UTINPI('UNIT NUMBER',JUNIT)
            CALL G2WRIT(JUNIT)
C
         ELSEIF(IROUT.EQ. 7) THEN
            CALL UTINPI('PRINT OPTION',JOPT)
            CALL G2PRNT(JOPT)
C
         ELSEIF(IROUT.EQ. 8) THEN
            CALL UTINPI('NGROW',NGROW)
            CALL UTINPI('ITYPE',ITYPE)
            CALL G2GROW(NGROW,ITYPE,ICHANG)
            WRITE(JTERMO,540) ICHANG
C
         ELSEIF(IROUT.EQ. 9) THEN
            CALL UTINPI('IBCTY1',IBCTY1)
            CALL UTINPI('IBCTY2',IBCTY2)
            CALL G2SURF(IBCTY1,IBCTY2,ICHANG)
            WRITE(JTERMO,540) ICHANG
C
         ELSEIF(IROUT.EQ.10) THEN
           CALL G2VOID(ICHANG)
            WRITE(JTERMO,540) ICHANG
C
         ELSEIF(IROUT.EQ.11) THEN
            CALL UTINPI('LEVEL',LEVEL)
            CALL UTINPI('NADD ',NADD )
            CALL G2XPND(LEVEL,NADD)
C
         ELSEIF(IROUT.EQ.12) THEN
            CALL G2ADPT(ICHANG)
            WRITE(JTERMO,540) ICHANG
C
         ELSEIF(IROUT.EQ.13) THEN
            CALL G2EXAM
C
         ELSEIF(IROUT.EQ.14) THEN
            CALL G2SORT
C
         ELSEIF(IROUT.EQ.15) THEN
            CALL UTINPI('UNIT (<0 FOR INPUT, >0 FOR OUTPUT)',IUNIT)
            CALL G2NBOR(IUNIT)
C
         ELSEIF(IROUT.EQ.16) THEN
            CALL UTINPI('OUTPUT UNIT',IUNIT)
            CALL G2DATA(IUNIT)
C
```

```
      ELSEIF(IROUT.EQ.17) THEN
         CALL UTINPI('STARTING NODE (<0 FOR S-N, >0 FOR W-E)',ISTART)
         CALL G2INTG(ISTART,XINT,YINT)
         WRITE(JTERMO,*) 'XINT=',XINT,'     YINT=',YINT
C
      ELSEIF(IROUT.EQ.18) THEN
         CALL UTINPI('OUTPUT UNIT',IUNIT)
         CALL G2CHEK(IUNIT,IER)
         WRITE(JTERMO,FMT='('' NUMBER OF ERRORS FOUND='',I6)') IER
C
      ELSEIF(IROUT.EQ.21) THEN
         CALL E2INIT(1)
C
      ELSEIF(IROUT.EQ.22) THEN
         CALL UTINPF('DUTOL',DUTOL)
         CALL UTINPF('CLTOL',CLTOL)
         CALL UTINPF('CDTOL',CDTOL)
         CALL E2CTRL(DUTOL,CLTOL,CDTOL)
C
      ELSEIF(IROUT.EQ.23) THEN
         CALL E2MAIN(CHANGE,CLIFT,CDRAG)
         WRITE(JTERMO,520) CHANGE,CLIFT,CDRAG
520      FORMAT(' MAX DU2=',E12.5,'  CLIFT=',F6.4,'  CDRAG=',F6.4)
C
      ELSEIF(IROUT.EQ.24) THEN
         CALL E2HSPR(1)
C
      ELSEIF(IROUT.EQ.25) THEN
         WRITE(JTERMO,530)
530      FORMAT('  1. X-LOCATION        31. VELOCITY           '/
     &          '  2. Y-LOCATION        32. FLOW ANGLE         '/
     &          '                       33. MACH NUMBER        '/
     &          ' 11. RHO               34. P/PT               '/
     &          ' 12. RHO*U             35. P/PREF             '/
     &          ' 13. RHO*V             36. ABS(PT-PTREF)/PTREF'/
     &          ' 14. RHO*E             37. (S-SREF)/CV        '/
     &          ' 15. WGHT              38. -CP                '/
     &          '                       39. (PT-PTREF)/PTREF   '/)
         CALL UTINPI('VARIABLE NUMBER',IVAR)
         CALL E2DATA(IVAR)
C
      ELSEIF(IROUT.EQ.26) THEN
         CALL E2FORC
C
      ELSEIF(IROUT.EQ.27) THEN
         CALL E2PRNT
C
      ELSEIF(IROUT.EQ.28) THEN
         CALL UTINPI('UNIT NUMBER',IUNIT)
         CALL E2STAT(IUNIT)
C
      ELSEIF(IROUT.EQ.29) THEN
         CALL UTINPI('IBEG  ',IBEG  )
```

```
         CALL UTINPI('IEND  ',IEND  )
         CALL UTINPI('ISTEP ',ISTEP )
         CALL UTINPI('IUNIT ',IUNIT )
         CALL E2VRFY(IBEG,IEND,ISTEP,IUNIT)
C
      ELSEIF(IROUT.EQ.30) THEN
         CALL UTINPI('IUNIT',IUNIT)
         CALL E2COMP(IUNIT)
C
      ELSEIF(IROUT.EQ.31) THEN
         CALL F2INIT(1)
C
      ELSEIF(IROUT.EQ.32) THEN
         CALL F2VALU
C
      ELSEIF(IROUT.EQ.33) THEN
         CALL F2GRAD
C
      ELSEIF(IROUT.EQ.34) THEN
         CALL F2LAPL
C
      ELSEIF(IROUT.EQ.35) THEN
         CALL F2THRS(TDIVD,TFUSE)
        WRITE(JTERMO,535) TDIVD,TFUSE
535      FORMAT('     DIVIDE THRESHOLD=',F10.5,
     &          '       FUSE THRESHOLD=',F10.5)
C
      ELSEIF(IROUT.EQ.36) THEN
         CALL UTINPF('TDIVD',TDIVD)
         CALL UTINPF('TFUSE',TFUSE)
         CALL F2ADPT(TDIVD,TFUSE,NCHANG)
         WRITE(JTERMO,540) NCHANG
540      FORMAT(' NUMBER OF CHANGED CELLS=',I5)
C
      ELSEIF(IROUT.EQ.37) THEN
         CALL F2PRNT
C
      ELSEIF(IROUT.EQ.38) THEN
         CALL UTINPI('CHNGF2(??)',IVAR)
         CALL F2DATA(IVAR)
C
      ELSEIF(IROUT.EQ.41) THEN
         CALL UTINPI('INPUT UNIT',IUNIT)
         CALL EXREAD(IUNIT)
C
      ELSEIF(IROUT.EQ.42) THEN
         CALL UTINPI('OUTPUT UNIT',IUNIT)
         CALL EXLIST(IUNIT)
C
      ELSEIF(IROUT.EQ.43) THEN
         CALL UTINPI('OUTPUT UNIT',IUNIT)
         CALL EXFWRD(IUNIT)
C
```

```
        ELSEIF(IROUT.EQ.44) THEN
           CALL UTINPI('OUTPUT UNIT',IUNIT)
           CALL UTINPC('GOAL ATTRIBUTE',GOAL)
           CALL EXBACK(IUNIT,GOAL,GOALV)
           WRITE(JTERMO,660) GOAL,GOALV
660        FORMAT(' EXBACK RESULT:  ',A,'=',G15.7)
C
        ELSEIF(IROUT.EQ.45) THEN
           CALL EXEDIT('          ',DUM)
C
        ELSEIF(IROUT.EQ.46) THEN
           CALL UTINPI('OUTPUT UNIT',IUNIT)
           CALL EXDUMP(IUNIT)
C
        ENDIF
        GOTO 500
C
C*****TERMINATE PROCESSING
C
999     CALL UTWTIM('PROCESSING COMPLETE')
C
        STOP
        END
```

B.2 EXPROC — EXpert/PROCedural System

EXCOMN

```
C-----SCALARS
C     UKNOEX       VALUE ASSIGNED TO UNKNOWN PARAMETERS
C     NEXMEX       NUMBER OF RULES EXAMINED
C     NTRGEX       NUMBER OF RULES TRIGGERED
C     NFIREX       NUMBER OF RULES FIRED (ALSO TIME STAMP)
C     NMXQEX       MAXIMUM NUMBER OF RULES IN RULE STACK
C     IUNTEX       UNIT NUMBER FOR OUTPUT
C     ITRCEX       TRACE LEVEL
C     ICNTEX       NUMBER OF CURRENT CONTEXT
C
C-----RULE ORDERING CONTROLS
C     DYMCEX       DYNAMIC RULE ORDERING ON OR OFF
C     PRODEX       PRODUCTION SYSTEM ON OR OFF
C     ORD1EX       NAME OF 1ST MOST MAJOR RULE ORDERING
C     ORD2EX       NAME OF 2ND MOST MAJOR RULE ORDERING
C     ORD3EX       NAME OF 3RD MOST MAJOR RULE ORDERING
C         VALID ORDERINGS:'UNKNOWNS  '=MOST UNKNOWS FIRST
C                         'M_RECENT  '=MOST RECENTLY USED RULE FIRST
C                         'L_RECENT  '=LEAST RECENTLY USED RULE FIRST
```

```
C                        'PRIORITY  '=HIGHEST PRIORITY FIRST
C                        'PREMISES  '=MOST TRUE PREMISES FIRST
C                        'RANDOM    '=RANDOM ORDERING
C
C-----PARAMETER ARRAYS
C     LABLEX(IPARAM)   IPARAM=1,NVAREX
C                      NAME OF PARAMETER
C     VALUEX(IPARAM)   IPARAM=1,NVAREX
C                      VALUE OF PARAMETER
C
C-----RULE ARRAYS
C     RULEEX(IRULE)    IRULE=1,NRULEX
C                      NAME OF RULE
C     PRTYEX(IRULE)    IRULE=1,NRULEX
C                      PRIORITY OF RULE
C     IRULEX(I,IRULE)  IRULE=1,NRULEX
C            I=1   PREMISE NUMBER OF FIRST PREMISE
C              =2   ACTION NUMBER  OF FIRST ACTION
C              =3   NOT USED
C              =4   TIME STAMP
C              =5   CONTEXT NUMBER
C
C-----PREMISE ARRAYS
C     COMPEX(IPREM)    IPREM=1,NPRMEX
C                      COMPARATOR
C     IPRMEX(I,IPREM)  IPREM=1,NPRMEX
C            I=1   ATTRIBUTE NUMBER OF LEFT ARGUMENT
C            I=2   ATTRIBUTE NUMBER OF RIGHT ARGUMENT
C
C-----ACTION ARRAYS
C     ACTNEX(IACTN)    IACTN=1,NACTEX
C                      ACTION
C     IACTEX(I,IACTN)  IACTN=1,NACTEX
C            I=1   ATTRIBUTE OF 1ST ARGUMENT
C            I=2   ATTRIBUTE OF 2ND ARGUMENT
C            I=3   ATTRIBUTE OF 3RD ARGUMENT
C
C-----CONTEXT ARRAY
C     CONTEX(ICONT)    ICONT=0,NCNTEX
C                      CONTEXT NAMES (CONTEX(0)='initial   ')
C
C-----RULE STACK ARRAY
C     ISTKEX(I,ISTAK)  ISTAK=1,NSTKEX
C            I=1   RULE NUMBER
C              =2   NUMBER OF TRUE PREMISES
C              =3   NUMBER OF UNKNOWN INPUTS
C
C-----GOAL STACK ARRAY
C     IGOLEX(IGOAL)    IGOAL=1,NGOLEX
C                      >0  ATTRIBUTE NUMBER IS AN ACTIVE GOAL
C                      <0  -ATTRIUBUTE NUMBER IS NOT AN ACTIVE GOAL
C
        PARAMETER(NVAREX=100,
```

```
     &            MRULEX=100,
     &            MPRMEX=600,
     &            MACTEX=600,
     &            MCNTEX=20 )
C
      CHARACTER*10 DYMCEX,PRODEX,ORD1EX,ORD2EX,ORD3EX,
     &             LABLEX,COMPEX,ACTNEX,CONTEX
      CHARACTER*40 RULEEX
C
      COMMON /EXCOMN/ DYMCEX,PRODEX,ORD1EX,ORD2EX,ORD3EX,UKNOEX,
     &                NEXMEX,NTRGEX,NFIREX,NMXQEX,IUNTEX,ITRCEX,
     &                ICNTEX,
     &                LABLEX(MVAREX),VALUEX( MVAREX),NVAREX,
     &                RULEEX(MRULEX),PRTYEX( MRULEX),
     &                               IRULEX(5,MRULEX),NRULEX,
     &                COMPEX(MPRMEX),IPRMEX(2,MPRMEX),NPRMEX,
     &                ACTNEX(MACTEX),IACTEX(3,MACTEX),NACTEX,
     &                CONTEX(0:MCNTEX),               NCNTEX,
     &                ISTKEX(3,MRULEX),               NSTKEX,
     &                IGOLEX( MRULEX),                NGOLEX
```

EXBACK

```
      SUBROUTINE EXBACK(IUNIT,GOAL,
     &
     &                  ANSWER    )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE EXECUTES THE BACKWARD CHAINER ON THE EXPERT
C       SYSTEM.
C
      CHARACTER*(10) GOAL
C
C*****************************************************************************
C
      INCLUDE '[.EXPROC]EXCOMN.INC'
C
      DIMENSION IUNIN(0:MVAREX),IUNOUT(0:MVAREX)
C
      CHARACTER*40 BLANK /'                                        '/
C
C*****************************************************************************
C
C*****WRITE OUT HEADER AND STRATEGY INFORMATION
C
      IUNTEX=IUNIT
C
      CALL UTHEAD('BACKWARD CHAINING EXPERT SYSTEM INVOKED')
C
```

```
        WRITE(IUNTEX,10) ORD1EX,ORD2EX,ORD3EX
10      FORMAT(' RULES ORDERED BY (MAJOR TO MINOR):',3A10,'NUMERIC'/)
C
C     INITIALIZE PERFORMANCE STATISTICS
C
        NEXMEX=0
        NTRGEX=0
        NFIREX=0
        NMXQEX=0
C
C     INITIALIZE CONTEXT
C
        ICNTEX=1
C
C     INITIALIZE TRIGGERED RULE STACK
C
        NSTKEX=0
C
C     PUT THE INPUT GOAL ON THE TOP OF THE GOAL STACK (INACTIVE)
C
        CALL EXVLST(GOAL,IGOAL)
        IGOLEX(1)=-IGOAL
        NGOLEX=1
C
C*****START AN EXPERT SYSTEM CYCLE
C
C     IF THERE AREN'T ANY GOALS ON THE STACK, THEN WE ARE FINISHED
C
20      IF(NGOLEX.EQ.0) GOTO 140
C
        IF(ITRCEX.GE.2) WRITE(IUNTEX,25) (IGOLEX(II),II=1,NGOLEX)
25      FORMAT(10X,'EX-TRACE>> CURRENT GOAL STACK:',10I6)
C
C     IF THE CURRENT GOAL IS KNOWN, POP IT OFF THE STACK AND START A
C        NEW CYCLE
C
        IGOAL=ABS(IGOLEX(NGOLEX))
C
        IF(VALUEX(IGOAL).NE.UKNOEX) THEN
           NGOLEX=NGOLEX-1
           GOTO 20
        ENDIF
C
C     IF CURRENT GOAL IS ACTIVE, USE THE RULE THAT IS ALREADY ON THE TOP
C        OF THE STACK
C
        IF(IGOLEX(NGOLEX).GT.0) GOTO 70
C
C     ACTIVATE THE CURRENT GOAL
C
        IGOLEX(NGOLEX)=IGOAL
C
C     REMEMBER THE STACK SIZE BEFORE THE FOLLOWING RULES ARE ADDED
```

```
C
        NSTACK=NSTKEX
C
C     LOOP THROUGH ALL RULES, LOOKING FOR THOSE WHICH:
C        - ASSERT A VALUE FOR THE CURRENT GOAL, AND
C        - HAS NO FALSE PREMISES (NTRUE.EQ.-1), AND
C        - IS IN A VALID CONTEXT (NTRUE.EQ.-2), AND
C        - HAS NO UNKNOWNS INPUT WHICH ARE CURRENTLY ACTIVE GOALS
C
        DO 60 IRULE=NRULEX,1,-1
C
        CALL EXEXAM(IRULE,NTRUE,IUNIN,IUNOUT)
C
C     ELIMINATE THOSE WITH A FLASE PREMISE OF WRONG CONTEXT AS WELL AS
C        THOSE WITHOUT ANY UNKNOWN ASSERTIONS
C
        IF(NTRUE.LT.0 .OR. IUNOUT(0).EQ.0) GOTO 60
C
C     SEARCH THROUGH THE LIST OF UNKNOWN ASSERTIONS, LOOKING FOR
C        THE CURRENT GOAL
C
        DO 30 IOUT=1,IUNOUT(0)
        IF(IUNOUT(IOUT).EQ.IGOAL) GOTO 40
30      CONTINUE
C
C     THIS RULE DOES NOT ASSERT THE CURRENT GOAL, SO GO TO NEXT RULE
C
        GOTO 60
C
C     SEARCH THROUGH THE LIST OF UNKNOWN INPUTS, LOOKING FOR A MATCH
C        BETWEEN THAT LIST AND THE LIST OF ACTIVE GOALS
C
40      DO 50 IIN=1,IUNIN(0)
        DO 50 IGOL=1,NGOLEX
        IF(IUNIN(IIN).EQ.IGOLEX(IGOL)) GOTO 60
50      CONTINUE
C
C     ALL THE ABOVE CRITERIA ARE MET, SO ADD THIS RULE TO THE STACK
C
        NSTKEX=NSTKEX+1
        ISTKEX(1,NSTKEX)=IRULE
        ISTKEX(2,NSTKEX)=NTRUE
        ISTKEX(3,NSTKEX)=IUNIN(0)
C
60      CONTINUE
C
C     IF NO RULES WERE ADD TO THE STACK, WE HAVE HIT A DEAD-END, SO
C        JUMP TO BACK-TRACKING LOGIC
C
        IF(NSTKEX.EQ.NSTACK) GOTO 110
C
C     IF MORE THAN ON RULE WAS ADDED TO THE STACK, USE CONFLICT
C        RESOLUTION ONLY ON THE NEWLY ADDED RULES
```

```
C
      IF(NSTKEX.GT.NSTACK+1) THEN
         CALL EXORDR(ORD3EX,NSTACK+1,NSTKEX)
         CALL EXORDR(ORD2EX,NSTACK+1,NSTKEX)
         CALL EXORDR(ORD1EX,NSTACK+1,NSTKEX)
      ENDIF
C
      IF(ITRCEX.GE.2) WRITE(IUNTEX,65) (ISTKEX(1,II),II=1,NSTKEX)
65    FORMAT(10X,'EX-TRACE>> CURRENT RULE STACK:',10I5)
C
C     CHECK THAT THE RULE ON TOP OF STACK IS STILL VALID TO FIRE
C       (HAS NO FALSE PREMISES OR CONTEXT).  IF NOT, JUMP TO
C       RULE DELETION
C
70    IRULE=ISTKEX(1,NSTKEX)
      CALL EXEXAM(IRULE,NTRUE,IUNIN,IUNOUT)
C
      IF(NTRUE.LT.0) GOTO 120
C
C     IF THIS RULE HAS ANY UNKNOW INPUTS, PUT THEM ON THE GOAL STACK
C       (INACTIVE) AND START A NEW CYCLE
C
      IF(IUNIN(0).GT.0) THEN
         IF(ITRCEX.GE.2) WRITE(IUNTEX,75) IUNIN(0)
75       FORMAT(10X,'EX-TRACE>> ADDING',I5,' GOALS TO STACK')
         DO 80 IIN=1,IUNIN(0)
         NGOLEX=NGOLEX+1
         IGOLEX(NGOLEX)=-IUNIN(IIN)
80       CONTINUE
C
         GOTO 20
      ENDIF
C
C     THERE ARE NO UNKNOWN INPUTS, SO NOW WE CAN FIRE THE RULE
C       ON THE TOP OF THE STACK
C
      CALL EXFIRE(IRULE)
C
C     DELETE ALL RULES WHICH ASSERT THE CURRENT GOAL
C
90    IRULE=ISTKEX(1,NSTKEX)
C
      DO 100 IACTN=IRULEX(2,IRULE),(IRULEX(2,IRULE+1)-1)
      DO 100 IARG=1,3
C
      IF(IACTEX(IARG,IACTN).EQ.(-IGOAL)) THEN
         IF(ITRCEX.GE.2) WRITE(IUNTEX,95) ISTKEX(1,NSTKEX)
95       FORMAT(10X,'EX-TRACE>> DELETING RULE',I5,' FROM STACK')
         NSTKEX=NSTKEX-1
         GOTO 90
      ENDIF
100   CONTINUE
C
```

```
C     GO BACK FOR NEXT CYCLE
C
        GOTO 20
C
C     BACK-TRACKING LOGIC ... DELETE ALL GOAL BACK TO THE PREVIOUS
C        ACTIVE ONE
C
110     IF(ITRCEX.GE.2) WRITE(IUNTEX,115) IGOLEX(NGOLEX)
115     FORMAT(10X,'EX-TRACE>> BACK-TRACKING...DELETING',I5,
     &                                          ' FROM GOAL STACK')
C
        NGOLEX=NGOLEX-1
        IGOAL=ABS(IGOLEX(NGOLEX))
C
        IF(IGOLEX(NGOLEX).LT.0) GOTO 110
C
C     DELETE THE CURRENT RULE FROM THE STACK
C
120     IF(ITRCEX.GE.2) WRITE(IUNTEX,125) ISTKEX(1,NSTKEX)
125     FORMAT(10X,'EX-TRACE>> DELETING',I5,' FROM RULE STACK')
C
        NSTKEX=NSTKEX-1
C
C     IF THERE AREN'T RULES ON THE STACK, WE CANNOT DETERMINE THE
C        CURRENT GOAL VALUE, SO RETURN
C
        IF(NSTKEX.LE.0) GOTO 140
C
C     IF RULE ON TOP OF STACK ASSERTS THE CURRENT GOAL, TRY TO
C        FIRE IT
C
        IRULE=ISTKEX(1,NSTKEX)
        CALL EXEXAM(IRULE,NTRUE,IUNIN,IUNOUT)
C
        DO 130 IOUT=1,IUNOUT(0)
        IF(IUNOUT(IOUT).EQ.IGOAL) GOTO 70
130     CONTINUE
C
C     THE RULE ON THE TOP OF THE STACK DOES NOT ASSERT THE CURRENT GOAL,
C        SO CONTINUE BACK-TRACKING
C
        IF(ITRCEX.GE.2) WRITE(IUNTEX,135) ISTKEX(1,NSTKEX),IGOAL
135     FORMAT(10X,'EX-TRACE>> CURRENT RULE (',I5,') DOES NOT ASSERT'
     &                        ' CURRENT GOAL (',I5,')'                )
        GOTO 110
C
C*****WE HAVE EITHER FOUND THE DESIRED GOAL OR COULD NOT.  IN EITHER
C        CASE PRINT FINAL MESSAGES
C
140     CALL EXVLST(GOAL,IGOAL)
        ANSWER=VALUEX(IGOAL)
C
        WRITE(IUNTEX,150) GOAL,ANSWER,
```

```
     &                    CONTEX(ICNTEX),
     &                    NEXMEX,
     &                    NTRGEX,
     &                    NFIREX,
     &                    NMXQEX
150     FORMAT(' EXPERT SYSTEM TERMINATING'//
     &         ' BACKWARD CHAINING DEDUCED THAT ',A,'=',G15.7//
     &         ' FINAL CONTEXT: ',A10/
     &         ' STATISTICS   : NUMBER OF RULES EXAMINED  =',I5/
     &         '                NUMBER OF RULES TRIGGERED =',I5/
     &         '                NUMBER OF RULES FIRED     =',I5/
     &         '                LONGEST STACK LENGTH      =',I5)
C
        RETURN
        END
```

EXCLST

```
        SUBROUTINE EXCLST(CONT,
     &
     &                    ICNT)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE LOOKS FOR THE CONTEXT 'CONT' IN THE CONTEXT
C        LIST.  IF IT FINDS IT, IT RETURNS THE CONTEXT NUMBER.
C        IF NOT FOUND IN THE LIST, THEN IT IS ADDED TO END OF THE LIST
C        AND ITS NUMBER IS RETURNED
C
        CHARACTER*10 CONT
C
C*************************************************************************
C
        INCLUDE '[.EXPROC]EXCOMN.INC'
C
C*************************************************************************
C
C     LOOK FOR THIS CONTEXT IN THE LIST
C
        DO 10 I=1,NCNTEX
        IF(CONT.NE.CONTEX(I)) GOTO 10
C
        ICNT=I
        RETURN
C
10      CONTINUE
C
C     CONTEXT IS NOT IN LIST, SO ADD TO END OF LIST
C
        IF(NCNTEX.GE.MCNTEX) THEN
```

```
         CALL UTEROR(1,REAL(NCNTEX),REAL(NCNTEX))
       ENDIF
C
       NCNTEX=NCNTEX+1
       CONTEX(NCNTEX)=CONT
C
       ICNT=NCNTEX
       RETURN
C
       END
```

EXDUMP

```
       SUBROUTINE EXDUMP(IUNIT
     &
     &                         )
C
       INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE DUMPS THE CURRENT EXCOMN CONTENTS ON UNIT IUNIT
C
C*****************************************************************************
C
       INCLUDE '[.EXPROC]EXCOMN.INC'
C
C*****************************************************************************
C
       JUNIT=ABS(IUNIT)
C
       WRITE(JUNIT,10) DYMCEX,PRODEX,ORD1EX,ORD2EX,ORD3EX
10     FORMAT(' DUMP OF EXCOMN:'                          /
     &         ' DYMCEX=',A,' PRODEX=',A,' ORD1EX=',A,
     &         ' ORD2EX=',A,' ORD3EX=',A                 )
C
       WRITE(JUNIT,20) (I,LABLEX(I),VALUEX(I),I=1,NVAREX)
20     FORMAT(I5,A,'=',G13.6)
C
       WRITE(JUNIT,30) (I,RULEEX(I),PRTYEX(I),(IRULEX(J,I),J=1,5),
     &                                                  I=1,NRULEX)
30     FORMAT(I5,A,' PRTY=',F6.3,' IRULEX=',5I5)
C
       WRITE(JUNIT,40) (I,IPRMEX(1,I),COMPEX(I),IPRMEX(2,I),I=1,NPRMEX)
40     FORMAT(I5,' PREMISE=',I5,A,I5)
C
       WRITE(JUNIT,50) (I,ACTNEX(I),(IACTEX(J,I),J=1,3),
     &                                          I=1,NACTEX)
50     FORMAT(I5,' ACTION=',A,3I5)
C
       WRITE(JUNIT,60) (I,CONTEX(I),I=0,NCNTEX)
60     FORMAT(I5,' CONTEXT=',A)
```

```
C
      WRITE(JUNIT,70) (I,(ISTKEX(J,I),J=1,3),I=1,NSTKEX)
70    FORMAT(I5,' ISTKEX=',3I5)
C
      WRITE(JUNIT,80) (I,IGOLEX(I),I=1,NGOLEX)
80    FORMAT(I5,' IGOLEX=',I5)
C
      RETURN
      END
```

EXEDIT

```
      SUBROUTINE EXEDIT(LABL,VALUE
     &
     &                                )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE ALLOWS A USER TO EXAMINE AND/OR CHANGE EXPROC
C       ATTRIBUTES
C
      CHARACTER*10 LABL
C
C*****************************************************************************
C
      INCLUDE '[.EXPROC]EXCOMN.INC'
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*****************************************************************************
C
      IF(LABL.NE.'          ') THEN
         CALL EXVLST(LABL,ILABL)
         VALUEX(ILABL)=VALUE
         RETURN
      ENDIF
C
10    WRITE(JTERMO,20) (I,LABLEX(I),VALUEX(I),I=7,NVAREX)
20    FORMAT(I5,'. ',A,'=',G15.7)
C
      CALL UTINPI('ATTRIBUTE TO CHANGE',IOPT)
C
      IF(IOPT.EQ.0) THEN
         RETURN
      ELSEIF(IOPT.GE.8 .AND. IOPT.LE.NVAREX) THEN
         CALL UTINPF(LABLEX(IOPT),VALUEX(IOPT))
      ENDIF
C
      GOTO 10
C
```

```
      END
```

EXEXAM

```
      SUBROUTINE EXEXAM(IRULE,
     &
     &                  NTRUE,IUNIN,IUNOUT)
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE EVALUATES THE PREMISES OF THE IRULE'TH
C       RULE.  IF ANY PREMISE IS FALSE, NTRUE IS SET TO -1.  IF THE
C       RULE'S CONTEXT IS NOT THE CURRENT ONE OR BLANK, NTRUE
C       IS SET TO -1, ELSE NTRUE CONTAINS THE NUMBER OF TRUE
C       PREMISES.  IUNIN AND IUNOUT ARE ARRAY WHOSE FIRST ELEMENTS
C       CONTAIN THE NUMBER OF UNKNOWN INPUT AND OUTPUT ATTRIBUTES,
C       RESPECTIVELY.  THE REMAINDER OF THESE ARRAYS CONTAIN LISTS
C       OF THE INDICES OF THE UNKNOWNS
C
      DIMENSION IUNIN(0:*),IUNOUT(0:*)
C
C*****************************************************************
C
      INCLUDE '[.EXPROC]EXCOMN.INC'
C
      CHARACTER*10 COMP
C
C*****************************************************************
C
C     INITIALIZE THE NUMBER OF TRUE PREMISES AND THE NUMBERS OF
C       UNKNOWNS
C
      NTRUE=0
      IUNIN (0)=0
      IUNOUT(0)=0
C
C     CHECK IF THE CONTEXT OF THIS RULE IS NOT 0 AND NOT THE CURRENT
C       CONTEXT
C
      IF(IRULEX(5,IRULE).NE.0       .AND.
     &   IRULEX(5,IRULE).NE.ICNTEX      ) GOTO 110
C
C     KEEP TRACK OF HOW MANY RULES HAVE BEEN EXAMINED
C
      NEXMEX=NEXMEX+1
C
C     LOOP THROUGH ALL THE PREMISES
C
      DO 50 IPREM=IRULEX(1,IRULE),(IRULEX(1,IRULE+1)-1)
      ICONT=1
```

```
C
C     GET THE FIRST PARAMETER.  IF IT IS NOT KNOWN, PUT IT IN THE
C        UNKNOWN INPUT PARAMETER LIST
C
        ARGA=VALUEX(IPRMEX(1,IPREM))
        IF(ARGA.EQ.UKNOEX) THEN
           IARGA=IPRMEX(1,IPREM)
           DO 10 II=1,IUNIN(0)
           IF(IARGA.EQ.IUNIN(II)) GOTO 20
10         CONTINUE
C
           IUNIN(0)=IUNIN(0)+1
           IUNIN(IUNIN(0))=IARGA
20         ICONT=0
        ENDIF
C
C     GET THE SECOND PARAMETER.  IF IT IS NOT KNOWN, PUT IT IN THE
C        UNKNOWN INPUT PARAMETER LIST
C
        ARGB=VALUEX(IPRMEX(2,IPREM))
        IF(ARGB.EQ.UKNOEX) THEN
           IARGB=IPRMEX(2,IPREM)
           DO  30 II=1,IUNIN(0)
           IF(IARGB.EQ.IUNIN(II)) GOTO 40
30         CONTINUE
C
           IUNIN(0)=IUNIN(0)+1
           IUNIN(IUNIN(0))=IARGB
40         ICONT=0
        ENDIF
C
C     IF EITHER OF THE ARGUMENTS ABOVE WERE UNKNOWN, GO TO NEXT PREMISE
C
        IF(ICONT.EQ.0) GOTO 50
C
C     GET THE COMPARATOR
C
        COMP=COMPEX(IPREM)
C
C     PERFORM THE INVERSE OF THE SPECIFIED TEST..IF IT IS TRUE, THEN
C        THIS PREMISE IS FALSE, SO JUMP TO END
C
        IF(    COMP.EQ.'.LT.      ') THEN
           IF(ARGA.GE.ARGB) GOTO 100
           NTRUE=NTRUE+1
C
        ELSEIF(COMP.EQ.'.LE.      ') THEN
           IF(ARGA.GT.ARGB) GOTO 100
           NTRUE=NTRUE+1
C
        ELSEIF(COMP.EQ.'.EQ.      ') THEN
           IF(ARGA.NE.ARGB) GOTO 100
           NTRUE=NTRUE+1
```

```
C
        ELSEIF(COMP.EQ.'.GE.      ') THEN
           IF(ARGA.LT.ARGB) GOTO 100
           NTRUE=NTRUE+1
C
        ELSEIF(COMP.EQ.'.GT.      ') THEN
           IF(ARGA.LE.ARGB) GOTO 100
           NTRUE=NTRUE+1
C
        ELSEIF(COMP.EQ.'.NE.      ') THEN
           IF(ARGA.EQ.ARGB) GOTO 100
           NTRUE=NTRUE+1
C
        ELSEIF(COMP.EQ.'NOP       ') THEN
           CONTINUE
C
        ELSE
           CALL UTEROR(1,REAL(IRULE),REAL(IPREM))
C
        ENDIF
C
C     THIS PREMISE IS TRUE, SO GO BACK FOR NEXT
C
50      CONTINUE
C
C     ALL PREMISES ARE TRUE, SO LOOK TO SEE IF ALL VARIABLES NEEDED IN
C        THE ACTION PART ARE KNOWN.  FURTHERMORE, DETERMINE HOW MANY
C        OF THE ASSERTIONS ARE UNKNOWN
C
        DO 90 IACTN=IRULEX(2,IRULE),(IRULEX(2,IRULE+1)-1)
C
C     IF ACTION IS 'CONSIDER  ', THEN IT DOES NOT POINT TO ATTRIBUTES,
C        SO GO TO NEXT ACTION CLAUSE
C
        IF(ACTNEX(IACTN).EQ.'CONSIDER  ') GOTO 90
C
        DO 80 IARG =1,3
C
C     IF VALUE IS KNOWN, JUMP TO END OF LOOP(S)
C
        JARG=IACTEX(IARG,IACTN)
        IF(VALUEX(ABS(JARG)).NE.UKNOEX) GOTO 80
C
C     VALUE IS UNKNOWN, SO DETERMINE IF ITS USE HERE IS AS AN
C        INPUT OR OUT ATTRIBUTE
C
        IF(JARG.GT.0) THEN
C
C     ...ATTTRIBUTE IS INPUT.  ADD IT TO THE UNKNOWN INPUT LIST IF
C        CURRENTLY ON NEITHER THE UNKNOWN INPUT OR UNKNOWN OUTPUT
C        LISTS
C
          DO 60 II=1,IUNIN(0)
```

```
            IF(JARG.EQ.IUNIN(II)) GOTO 80
60          CONTINUE
            DO 65 II=1,IUNOUT(0)
            IF(JARG.EQ.IUNOUT(II)) GOTO 80
65          CONTINUE
C
            IUNIN(0)=IUNIN(0)+1
            IUNIN(IUNIN(0))=JARG
C
          ELSE
C
C       ...ATTRIBUTE IS OUTPUT. ADD IT TO THE UNKNOWN OUTPUT LIST IF
C           CURRENTLY NOT ON THE LIST
C
            DO 70 II=1,IUNOUT(0)
            IF(JARG.EQ.IUNOUT(II)) GOTO 80
70          CONTINUE
C
            IUNOUT(0)=IUNOUT(0)+1
            IUNOUT(IUNOUT(0))=-JARG
C
          ENDIF
C
C       GET NEXT ARGUMENT AND ACTION CLAUSE
C
80        CONTINUE
90        CONTINUE
C
C       RETURN THE NUMBER OF PREMISES AND INCREMENT NUMBER OF TRIGGERED
C           RULES
C
          NTRGEX=NTRGEX+1
C
          GOTO 120
C
C       AT LEAST ONE OF THE PREMISES IS FALSE
C
100       NTRUE=-1
          IUNIN (0)=0
          IUNOUT(0)=0
C
          GOTO 120
C
C       CONTEXT IS NOT CORRECT FOR CURRENT SITUATION
C
110       NTRUE=-2
          IUNIN (0)=0
          IUNOUT(0)=0
C
C       PRINT OUT TRACE INFORMATION
C
120       IF(ITRCEX.LT.4) RETURN
C
```

```
      WRITE(IUNTEX,130) IRULE,PRTYEX(IRULE),CONTEX(IRULEX(5,IRULE)),
     &                                              IRULEX(4,IRULE)
130   FORMAT(10X,'EX-TRACE>> EXAMINING RULE',I5,'  priority =',F10.4,
     &                                         '  context  =',A    ,
     &                                         '  timestamp=',I5   )
C
      IF(IUNIN(0).GT.0) THEN
        WRITE(IUNTEX,140) (IUNIN(I),I=1,IUNIN(0))
140     FORMAT(10X,'EX-TRACE>>',22X,'unknown inputs :    ',10I5)
      ENDIF
C
      IF(IUNOUT(0).GT.0) THEN
        WRITE(IUNTEX,150) (IUNOUT(I),I=1,IUNOUT(0))
150     FORMAT(10X,'EX-TRACE>>',22X,'unknown outputs:    ',10I5)
      ENDIF
C
      RETURN
C
      END
```

EXFIRE

```
      SUBROUTINE EXFIRE(IRULE
     &
     &                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PERFORMS THE ACTION STATEMENTS FOR THE
C       IRULE'TH RULE
C
C*********************************************************************
C
      INCLUDE '[.EXPROC]EXCOMN.INC'
C
      CHARACTER*10 ACTION
C
      CHARACTER*40 BLANK /'                                        '/
C
C*********************************************************************
C
      IF(ITRCEX.GE.1) WRITE(IUNTEX,10) IRULE
10    FORMAT(10X,'EX-TRACE>> FIRING RULE',I5)
C
C     LOOP THROUGH ALL THE ACTION CLAUSES
C
      DO 40 IACTN=IRULEX(2,IRULE),(IRULEX(2,IRULE+1)-1)
C
C     GET THE ACTION TO BE PERFORMED
C
```

```
        ACTION=ACTNEX(IACTN)
C
C     GET THE THREE ARGUMENTS
C
        IA=ABS(IACTEX(1,IACTN))
        IB=ABS(IACTEX(2,IACTN))
        IC=ABS(IACTEX(3,IACTN))
C
        IF(ITRCEX.GE.3) THEN
           OLDA=VALUEX(IA)
           OLDB=VALUEX(IB)
           OLDC=VALUEX(IC)
C
           WRITE(IUNTEX,15) ACTION,LABLEX(IA),LABLEX(IB),LABLEX(IC)
15         FORMAT(10X,'EX-TRACE>> ACTION:',4(A,2X))
        ENDIF
C
C     PERFORM THE APPROPRIATE ACTION
C
C-----NOP
C
        IF(    ACTION.EQ.'NOP       ') THEN
           CONTINUE
C
C-----SET
C
        ELSEIF(ACTION.EQ.'SET       ') THEN
           VALUEX(IA)=VALUEX(IB)
C
C-----GET
C
        ELSEIF(ACTION.EQ.'GET       ') THEN
           IOFF=NINT(VALUEX(IC))
           VALUEX(IA)=VALUEX(IB+IOFF)
C
C-----PUT
C
        ELSEIF(ACTION.EQ.'PUT       ') THEN
           IOFF=NINT(VALUEX(IB))
           VALUEX(IA+IOFF)=VALUEX(IC)
C
C-----RESET
C
        ELSEIF(ACTION.EQ.'RESET     ') THEN
           VALUEX(IA)=UKNOEX
C
C-----ADD
C
        ELSEIF(ACTION.EQ.'ADD       ') THEN
           VALUEX(IA)=VALUEX(IB)+VALUEX(IC)
C
C-----SUBTRACT
C
```

```
        ELSEIF(ACTION.EQ.'SUBTRACT  ') THEN
           VALUEX(IA)=VALUEX(IB)-VALUEX(IC)
C
C-----MULTIPLY
C
        ELSEIF(ACTION.EQ.'MULTIPLY  ') THEN
           VALUEX(IA)=VALUEX(IB)*VALUEX(IC)
C
C-----DIVIDE
C
        ELSEIF(ACTION.EQ.'DIVIDE    ') THEN
           VALUEX(IA)=VALUEX(IB)/VALUEX(IC)
C
C-----ABS
C
        ELSEIF(ACTION.EQ.'ABS       ') THEN
           VALUEX(IA)=ABS(VALUEX(IB))
C
C-----SQRT
C
        ELSEIF(ACTION.EQ.'SQRT      ') THEN
           VALUEX(IA)=SQRT(VALUEX(IB))
C
C-----EXPON
C
        ELSEIF(ACTION.EQ.'EXPON     ') THEN
           VALUEX(IA)=VALUEX(IB)**VALUEX(IC)
C
C-----MIN
C
        ELSEIF(ACTION.EQ.'MIN       ') THEN
           VALUEX(IA)=MIN(VALUEX(IB),VALUEX(IC))
C
C-----MAX
C
        ELSEIF(ACTION.EQ.'MAX       ') THEN
           VALUEX(IA)=MAX(VALUEX(IB),VALUEX(IC))
C
C-----SHOW
C
        ELSEIF(ACTION.EQ.'SHOW      ') THEN
           IF(LABLEX(IA).NE.'          ')
     &                    WRITE(IUNTEX,20) LABLEX(IA),VALUEX(IA)
           IF(LABLEX(IB).NE.'          ')
     &                    WRITE(IUNTEX,20) LABLEX(IB),VALUEX(IB)
           IF(LABLEX(IC).NE.'          ')
     &                    WRITE(IUNTEX,20) LABLEX(IC),VALUEX(IC)
20         FORMAT(' ===> ',A10,'=',G15.7)
C
C-----STOPIT
C
        ELSEIF(ACTION.EQ.'STOPIT    ') THEN
           VALUEX(IA)=UTSTOP(IDUM)
```

```
C
C-----ERROR
C
        ELSEIF(ACTION.EQ.'ERROR     ') THEN
           IERR=NINT(VALUEX(IA))
           ERR1=VALUEX(IB)
           ERR2=VALUEX(IC)
           CALL UTEROR(IERR,ERR1,ERR2)
C
C-----CONSIDER
C
        ELSEIF(ACTION.EQ.'CONSIDER  ') THEN
           ICNTEX=IA
C
C-----E2COMP
C
        ELSEIF(ACTION.EQ.'E2COMP    ') THEN
           IUNIT=VALUEX(IA)
           CALL E2COMP(IUNIT)
C
C-----E2CTRL
C
        ELSEIF(ACTION.EQ.'E2CTRL    ') THEN
           DUTOL=VALUEX(IA)
           CLTOL=VALUEX(IB)
           CDTOL=VALUEX(IC)
           CALL E2CTRL(DUTOL,CLTOL,CDTOL)
C
C-----E2DATA
C
        ELSEIF(ACTION.EQ.'E2DATA    ') THEN
           IVAR=NINT(VALUEX(IA))
           CALL E2DATA(IVAR)
C
C-----E2FORC
C
        ELSEIF(ACTION.EQ.'E2FORC    ') THEN
           CALL E2FORC
C
C-----E2HSPR
C
        ELSEIF(ACTION.EQ.'E2HSPR    ') THEN
           CALL E2HSPR(1)
C
C-----E2INIT
C
        ELSEIF(ACTION.EQ.'E2INIT    ') THEN
           CALL E2INIT(1)
C
C-----E2MAIN
C
        ELSEIF(ACTION.EQ.'E2MAIN    ') THEN
           CALL E2MAIN(CHANGE,CLIFT,CDRAG)
```

```
            VALUEX(IA)=CHANGE
            VALUEX(IB)=CLIFT
            VALUEX(IC)=CDRAG
C
C-----E2PRNT
C
         ELSEIF(ACTION.EQ.'E2PRNT    ') THEN
            CALL E2PRNT
C
C-----E2STAT
C
         ELSEIF(ACTION.EQ.'E2STAT    ') THEN
            JUNIT=NINT(VALUEX(IA))
            CALL E2STAT(JUNIT)
C
C-----E2TIME
C
         ELSEIF(ACTION.EQ.'E2TIME    ') THEN
            JUNIT=NINT(VALUEX(IA))
            CALL E2TIME(JUNIT)
C
C-----E2VRFY
C
         ELSEIF(ACTION.EQ.'E2VRFY    ') THEN
            IBEG =NINT(VALUEX(IA))
            IEND =NINT(VALUEX(IB))
            ISTEP=NINT(VALUEX(IC))
            JUNIT=IUNTEX
            CALL E2VRFY(IBEG,IEND,ISTEP,JUNIT)
C
C-----E2WIND
C
         ELSEIF(ACTION.EQ.'E2WIND    ') THEN
            IPASS=NINT(VALUEX(IA))
            CALL E2WIND(IPASS,ICONV)
            VALUEX(IA)=IPASS
            VALUEX(IB)=ICONV
C
C-----F2ADPT
C
         ELSEIF(ACTION.EQ.'F2ADPT    ') THEN
            TDIVD=VALUEX(IA)
            TFUSE=VALUEX(IB)
            CALL F2ADPT(TDIVD,TFUSE,NCHANG)
            VALUEX(IC)=NCHANG
C
C-----F2CHNG
C
         ELSEIF(ACTION.EQ.'F2CHNG    ') THEN
            ITYPE=NINT(VALUEX(IA))
            CALL F2CHNG(ITYPE)
C
C-----F2DATA
```

```
C
        ELSEIF(ACTION.EQ.'F2DATA    ') THEN
           IVAR=NINT(VALUEX(IA))
           CALL F2DATA(IVAR)
C
C-----F2GRAD
C
        ELSEIF(ACTION.EQ.'F2GRAD    ') THEN
           CALL F2GRAD
C
C-----F2INIT
C
        ELSEIF(ACTION.EQ.'F2INIT    ') THEN
           IOPT=NINT(VALUEX(IA))
           CALL F2INIT(IOPT)
C
C-----F2LAPL
C
        ELSEIF(ACTION.EQ.'F2LAPL    ') THEN
           CALL F2LAPL
C
C-----F2PRNT
C
        ELSEIF(ACTION.EQ.'F2PRNT    ') THEN
           CALL F2PRNT
C
C-----F2THRS
C
        ELSEIF(ACTION.EQ.'F2THRS    ') THEN
           CALL F2THRS(TDIVD,TFUSE)
           VALUEX(IA)=TDIVD
           VALUEX(IB)=TFUSE
C
C-----F2VALU
C
        ELSEIF(ACTION.EQ.'F2VALU    ') THEN
           CALL F2VALU
C
C-----G2ADPT
C
        ELSEIF(ACTION.EQ.'G2ADPT    ') THEN
           CALL G2ADPT(NCHANG)
           VALUEX(IA)=NCHANG
C
C-----G2DATA
C
        ELSEIF(ACTION.EQ.'G2DATA    ') THEN
           IUNIT=NINT(VALUEX(IA))
           CALL G2DATA(IUNIT)
C
C-----G2DIVD
C
        ELSEIF(ACTION.EQ.'G2DIVD    ') THEN
```

```
            ICELL=NINT(VALUEX(IA))
            CALL G2DIVD(ICELL,NCHANG)
            VALUEX(IB)=NCHANG
C
C-----G2EXAM
C
         ELSEIF(ACTION.EQ.'G2EXAM    ') THEN
            CALL G2EXAM
C
C-----G2FUSE
C
         ELSEIF(ACTION.EQ.'G2FUSE    ') THEN
            ICELL=NINT(VALUEX(IA))
            CALL G2FUSE(ICELL,NCHANG)
            VALUEX(IB)=NCHANG
C
C-----G2GRBG
C
         ELSEIF(ACTION.EQ.'G2GRBG    ') THEN
            CALL G2GRBG
C
C-----G2GROW
C
         ELSEIF(ACTION.EQ.'G2GROW    ') THEN
            NGROW=NINT(VALUEX(IA))
            ITYPE=NINT(VALUEX(IB))
            CALL G2GROW(NGROW,ITYPE,NCHANG)
            VALUEX(IC)=NCHANG
C
C-----G2INTG
C
         ELSEIF(ACTION.EQ.'G2INTG    ') THEN
            ISTART=NINT(VALUEX(IA))
            CALL G2INTG(ISTART,XINT,YINT)
            VALUEX(IB)=XINT
            VALUEX(IC)=YINT
C
C-----G2NBOR
C
         ELSEIF(ACTION.EQ.'G2NBOR    ') THEN
            IUNIT=NINT(VALUEX(IA))
            CALL G2NBOR(IUNIT)
C
C-----G2PRNT
C
         ELSEIF(ACTION.EQ.'G2PRNT    ') THEN
            JOPT=NINT(VALUEX(IA))
            CALL G2PRNT(JOPT)
C
C-----G2READ
C
         ELSEIF(ACTION.EQ.'G2READ    ') THEN
            JUNITI=NINT(VALUEX(IA))
```

```
            JUNITO=NINT(VALUEX(IB))
            CALL G2READ(JUNITI,JUNITO)
C
C-----G2SORT
C
         ELSEIF(ACTION.EQ.'G2SORT    ') THEN
            CALL G2SORT
C
C-----G2SUMY
C
         ELSEIF(ACTION.EQ.'G2SUMY    ') THEN
            JUNIT=NINT(VALUEX(IA))
            CALL G2SUMY(JUNIT)
C
C-----G2SURF
C
         ELSEIF(ACTION.EQ.'G2SURF    ') THEN
            IBC1=NINT(VALUEX(IA))
            IBC2=NINT(VALUEX(IB))
            CALL G2SURF(IBC1,IBC2,NCHANG)
            VALUEX(IC)=NCHANG
C
C-----G2VOID
C
         ELSEIF(ACTION.EQ.'G2VOID    ') THEN
            CALL G2VOID(NCHANG)
            VALUEX(IA)=NCHANG
C
C-----G2WRIT
C
         ELSEIF(ACTION.EQ.'G2WRIT    ') THEN
            JUNIT=NINT(VALUEX(IA))
            CALL G2WRIT(JUNIT)
C
C-----G2XPND
C
         ELSEIF(ACTION.EQ.'G2XPND    ') THEN
            LEVEL=NINT(VALUEX(IA))
            NADD =NINT(VALUEX(IB))
            CALL G2XPND(LEVEL,NADD)
C
         ELSE
            CALL UTEROR(1,REAL(IRULE),REAL(IACTN))
C
         ENDIF
C
         IF(ITRCEX.GE.3) THEN
            IF(ACTION.EQ.'CONSIDER  ') THEN
               WRITE(IUNTEX,25) CONTEX(IA)
            ELSE
               IF(IA.GT.2 .AND. OLDA.NE.VALUEX(IA))
     &                        WRITE(IUNTEX,30) LABLEX(IA),OLDA,VALUEX(IA)
               IF(IA.GT.2 .AND. OLDA.EQ.VALUEX(IA))
```

```
     &                    WRITE(IUNTEX,35) LABLEX(IA),OLDA
               IF(IB.GT.2 .AND. OLDB.NE.VALUEX(IB) .AND. IB.NE.IA)
     &                    WRITE(IUNTEX,30) LABLEX(IB),OLDB,VALUEX(IB)
               IF(IB.GT.2 .AND. OLDB.EQ.VALUEX(IB) .AND. IB.NE.IA)
     &                    WRITE(IUNTEX,35) LABLEX(IB),OLDB
               IF(IC.GT.2 .AND. OLDC.NE.VALUEX(IC) .AND. IC.NE.IA
     &                                             .AND. IC.NE.IB)
     &                    WRITE(IUNTEX,30) LABLEX(IC),OLDC,VALUEX(IC)
               IF(IC.GT.2 .AND. OLDC.EQ.VALUEX(IC) .AND. IC.NE.IA
     &                                             .AND. IC.NE.IB)
     &                    WRITE(IUNTEX,35) LABLEX(IC),OLDC
            ENDIF
          ENDIF
25        FORMAT(10X,'EX-TRACE>>     CONTEXT CHANGED TO:',A10)
30        FORMAT(10X,'EX-TRACE>>     ATTRIBUTE: ',A10,' OLD VALUE=',G15.7,
     &                                          ' NEW VALUE=',G15.7)
35        FORMAT(10X,'EX-TRACE>>     ATTRIBUTE: ',A10,'     VALUE=',G15.7,
     &                                        ' ---unchanged--- ')
C
C
C     GO BACK FOR THE NEXT ACTION CLAUSE
C
40        CONTINUE
C
C     WRITE OUT RULE NAME AND TIME IF REQUIRED
C
          IF(RULEEX(IRULE).NE.BLANK) CALL UTWTIM(RULEEX(IRULE))
C
C     KEEP TRACK OF HOW MANY RULES HAVE BEEN FIRED AND TIME STAMP THIS
C        RULE
C
          NFIREX=NFIREX+1
          IRULEX(4,IRULE)=NFIREX
C
          RETURN
          END
```

EXFWRD

```
          SUBROUTINE EXFWRD(IUNIT
     &
     &                           )
C
          INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE EXECUTES THE FORWARD CHAINER ON THE EX... SYSTEM.
C
C**********************************************************************
C
          INCLUDE '[.EXPROC]EXCOMN.INC'
```

```
C
      DIMENSION IUNIN(0:MVAREX),IUNOUT(0:MVAREX)
C
C*************************************************************************
C
C*****WRITE OUT HEADER AND STRATEGY INFORMATION
C
      IUNTEX=IUNIT
C
      CALL UTHEAD('FORWARD CHAINING EXPERT SYSTEM INVOKED')
C
      WRITE(IUNTEX,10) ORD1EX,ORD2EX,ORD3EX,
     &                 DYMCEX,
     &                 PRODEX
10    FORMAT(' RULES ORDERED BY (MAJOR TO MINOR):',3A10,'NUMERIC'/
     &       ' DYNAMIC RULE ORDERING            :', A10          /
     &       ' PRODUCTION-TYPE RUN              :', A10          /)
C
C     INITIALIZE PERFORMANCE STATISTICS
C
      NEXMEX=0
      NTRGEX=0
      NFIREX=0
      NMXQEX=0
C
C     INITIALIZE CONTEXT
C
      ICNTEX=1
C
C     INITIALIZE THE TRIGGERED RULE STACK FOR FIRST TIME THROUGH
C
      NSTKEX=0
C
C*****START AN EXPERT SYSTEM CYCLE
C
20    IF(DYMCEX.EQ.'ON        ' .OR. NSTKEX.EQ.0) THEN
C
C     INITIALIZE THE STACK
C
        NSTKEX=0
C
C     PREVIEW ALL RULES, PUTTING THOSE WITH TRUE PREMISES ON THE STACK
C       IN REVERSE ORDER (THEREBY AUTOMATICALLY USING RULE ORDERING
C       AS MOST MINOR CONFLICT RESOLUTION STRATEGY)
C
        DO 30 IRULE=NRULEX,1,-1
C
C     EXAMINE THE RULE
C
        CALL EXEXAM(IRULE,NTRUE,IUNIN,IUNOUT)
C
C     REJECT RULES FROM CONSIDERATION IF:
C       - THERE ARE ANY FALSE PREMISES (NTRUE.EQ.-1),  OR
```

```
C        - IT IS NOT IN AN ALLOWABLE CONTEXT (NTRUE.EQ.-2), OR
C        - THE ARE UNKNOWN INPUT ATTRIBUTES, OR
C        - THERE IS NOT AN UNKNOWN ASSERTION (UNLESS PRODEX='ON')
C
           IF(NTRUE    .LT.0  .OR. IUNIN(0) .GT.0          ) GOTO 30
           IF(IUNOUT(0).EQ.0 .AND. PRODEX   .EQ.'OFF       ') GOTO 30
C
C     ADD THIS RULE TO THE STACK
C
           NSTKEX=NSTKEX+1
           ISTKEX(1,NSTKEX)=IRULE
           ISTKEX(2,NSTKEX)=NTRUE
           ISTKEX(3,NSTKEX)=0
C
30         CONTINUE
C
C     IF NO RULES ARE NOW IN THE STACK, WE ARE FINISHED, SO SKIP TO END
C
           IF(NSTKEX.EQ.0) GOTO 60
C
C     IF MORE THAN ONE RULE IS IN STACK, USE THE APPROPRIATE CONFICT
C        RESOLUTION ORDERINGS (MINOR TO MAJOR)
C
           IF(NSTKEX.GT.1) THEN
              CALL EXORDR(ORD3EX,1,NSTKEX)
              CALL EXORDR(ORD2EX,1,NSTKEX)
              CALL EXORDR(ORD1EX,1,NSTKEX)
           ENDIF
C
C     UPDATE THE STATISTIC CONCERNING THE LARGEST STACK SIZE
C
           NMXQEX=MAX(NMXQEX,NSTKEX)
C
C     SET THE RULE AT THE TOP OF THE STACK TO FIRE.  ALSO NOTE IF THE
C        IS THE LARGEST SO FAR
C
           IFIRE=ISTKEX(1,NSTKEX)
C
           IF(ITRCEX.GE.2) WRITE(IUNTEX,40) CONTEX(ICNTEX),
     &                                    (ISTKEX(1,I),I=1,NSTKEX)
40         FORMAT(10X,'EX-TRACE>> RULE STACK AFTER PREVIEWING AND',
     &                ' ORDERING  (CONTEXT=',A10,')'              /
     &           10X,'EX-TRACE>>',5X,20I5                         )
C
         ELSE
C
C     DYNAMIC RULE ORDERING IS IN EFFECT AND THIS IS NOT THE FIRST PASS,
C        SO RE-EVALUATE PREMISE OF LAST RULE IN STACK JUST TO BE
C        SURE THAT IT CAN STILL BE FIRED.  (USE SAME CRITERIA AS ABOVE)
C
           IFIRE=ISTKEX(1,NSTKEX)
           CALL EXEXAM(IFIRE,NTRUE,IUNIN,IUNOUT)
C
```

```
           IF(NTRUE    .LT.0  .OR.
     &        IUNIN(0) .GT.0  .OR.
     &        IUNOUT(0).EQ.0      ) IFIRE=0
C
        ENDIF
C
C     FIRE THE LAST RULE ON THE STACK AND WRITE OUT A TIMER MESSAGE
C        IF RULE HAS A NAME
C
        IF(IFIRE.GT.0) CALL EXFIRE(IFIRE)
C
C     DELETE THIS RULE FROM THE STACK, AND GO BACK FOR NEXT PASS
C
        NSTKEX=NSTKEX-1
        GOTO 20
C
C*****NO MORE RULE, SO PRINT FINAL MESSAGES
C
50      WRITE(IUNTEX,60) CONTEX(ICNTEX),
     &                   NEXMEX,
     &                   NTRGEX,
     &                   NFIREX,
     &                   NMXQEX
60      FORMAT(' EXPERT SYSTEM TERMINATING'//
     &         ' FINAL CONTEXT: ',A10/
     &         ' STATISTICS   : NUMBER OF RULES EXAMINED  =',I5/
     &         '                NUMBER OF RULES TRIGGERED =',I5/
     &         '                NUMBER OF RULES FIRED     =',I5/
     &         '                LONGEST STACK LENGTH      =',I5)
C
        RETURN
        END
```

EXLIST

```
        SUBROUTINE EXLIST(IUNIT
     &
     &                         )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CREATES A LISTING OF THE EXPERTS SYSTEM PROGRAM
C        ON UNIT IUNIT.  IF IUNIT.LT.0, JUST PRINT THE ATTRIBUTES
C
C*********************************************************************
C
        INCLUDE '[.EXPROC]EXCOMN.INC'
C
        CHARACTER*11 ARG1,ARG2,ARG3
        CHARACTER*10 ENTRY(100)
```

```
C
C*************************************************************************
C
      JUNIT=ABS(IUNIT)
C
C*****ATTRIBUTE TABLE
C
      WRITE(JUNIT,10)
10    FORMAT(/' EXPERT SYSTEM ATTRIBUTES:'/)
C
      WRITE(JUNIT,20) (LABLEX(I),VALUEX(I),I=1,NVAREX)
20    FORMAT(4X,'|',A10,'|=',G14.7:3X,'|',A10,'|=',G14.7:
     &       3X,'|',A10,'|=',G14.7:3X,'|',A10,'|=',G14.7)
C
      IF(IUNIT.LT.0) RETURN
C
C*****CONTEXT TABLE
C
      WRITE(JUNIT,30)
30    FORMAT(/' EXPERT SYSTEM CONTEXTS:'/)
C
      WRITE(JUNIT,40) (CONTEX(I),I=1,NCNTEX)
40    FORMAT(4X,'|',A10,'|')
C
C*****RULE TABLE
C
      WRITE(JUNIT,50)
50    FORMAT(/' EXPERT SYSTEM RULES:')
C
      DO 120 IRULE=1,NRULEX
C
      IPREM=IRULEX(1,IRULE)
      WRITE(JUNIT,60) IRULE,RULEEX(IRULE),
     &                      PRTYEX(IRULE),
     &                      CONTEX(IRULEX(5,IRULE)),
     &                      LABLEX(IPRMEX(1,IPREM)),COMPEX(IPREM),
     &                      LABLEX(IPRMEX(2,IPREM))
60    FORMAT(/' Rule',I4,'      :',A40  /
     &        '     priority  :',F10.4/
     &        '     context   :', A10 /
     &        '           if  :',A10,3X,A10,3X,A10)
C
      DO 80 IPREM=(IRULEX(1,IRULE)+1),(IRULEX(1,IRULE+1)-1)
      WRITE(JUNIT,70)       LABLEX(IPRMEX(1,IPREM)),COMPEX(IPREM),
     &                      LABLEX(IPRMEX(2,IPREM))
70    FORMAT('              :',A10,3X,A10,3X,A10)
80    CONTINUE
C
      IACTN=IRULEX(2,IRULE)
      ARG1=' '//LABLEX(ABS(IACTEX(1,IACTN)))
      ARG2=' '//LABLEX(ABS(IACTEX(2,IACTN)))
      ARG3=' '//LABLEX(ABS(IACTEX(3,IACTN)))
      IF(IACTEX(1,IACTN).LT.0) ARG1(1:1)='%'
```

```
      IF(IACTEX(2,IACTN).LT.0) ARG2(1:1)='%'
      IF(IACTEX(3,IACTN).LT.0) ARG3(1:1)='%'
C
      IF(ACTNEX(IACTN).EQ.'CONSIDER  ') THEN
         WRITE(JUNIT,90) ACTNEX(IACTN),' '//CONTEX(IACTEX(1,IACTN)),
     &                                   ARG2,
     &                                   ARG3
      ELSE
         WRITE(JUNIT,90) ACTNEX(IACTN),ARG1,
     &                                   ARG2,
     &                                   ARG3
      ENDIF
90    FORMAT( '           then:',A10,3(2X,A11))
C
      DO 110 IACTN=(IRULEX(2,IRULE)+1),(IRULEX(2,IRULE+1)-1)
      ARG1=' '//LABLEX(ABS(IACTEX(1,IACTN)))
      ARG2=' '//LABLEX(ABS(IACTEX(2,IACTN)))
      ARG3=' '//LABLEX(ABS(IACTEX(3,IACTN)))
      IF(IACTEX(1,IACTN).LT.0) ARG1(1:1)='%'
      IF(IACTEX(2,IACTN).LT.0) ARG2(1:1)='%'
      IF(IACTEX(3,IACTN).LT.0) ARG3(1:1)='%'
C
      IF(ACTNEX(IACTN).EQ.'CONSIDER  ') THEN
         WRITE(JUNIT,100) ACTNEX(IACTN),' '//CONTEX(IACTEX(1,IACTN)),
     &                                   ARG2,
     &                                   ARG3
      ELSE
         WRITE(JUNIT,100) ACTNEX(IACTN),ARG1,
     &                                   ARG2,
     &                                   ARG3
      ENDIF
100   FORMAT('               :',A10,3(2X,A11))
110   CONTINUE
C
120   CONTINUE
C
C*****CROSS REFERENCE TABLE
C
      WRITE(JUNIT,130)
130   FORMAT(/' CROSS REFERENCE TABLE (ATTRIBUTES):')
C
      DO 200 IARG=8,NVAREX
      IENT=0
C
      DO 180 IRULE=1,NRULEX
C
      DO 160 IPREM=IRULEX(1,IRULE),(IRULEX(1,IRULE+1)-1)
      IF(ABS(IPRMEX(1,IPREM)).EQ.IARG .OR.
     &   ABS(IPRMEX(2,IPREM)).EQ.IARG     ) THEN
         IENT=IENT+1
         WRITE(ENTRY(IENT),140) IRULE,(IPREM+1-IRULEX(1,IRULE))
140      FORMAT(I5,'(P',I2,')')
      ENDIF
```

```
150     CONTINUE
C
        DO 170 IACTN=IRULEX(2,IRULE),(IRULEX(2,IRULE+1)-1)
        IF(ABS(IACTEX(1,IACTN)).EQ.IARG .OR.
     &     ABS(IACTEX(2,IACTN)).EQ.IARG .OR.
     &     ABS(IACTEX(3,IACTN)).EQ.IARG     ) THEN
           IENT=IENT+1
           WRITE(ENTRY(IENT),160) IRULE,(IACTN+1-IRULEX(2,IRULE))
160        FORMAT(I5,'(A',I2,')')
        ENDIF
170     CONTINUE
C
180     CONTINUE
C
        WRITE(JUNIT,190) LABLEX(IARG),(ENTRY(I),I=1,IENT)
190     FORMAT(' Attribute:',A,(T25,8A))
200     CONTINUE
C
        WRITE(JUNIT,210)
210     FORMAT(/' CROSS REFERENCE TABLE (CONTEXTS):')
C
        DO 250 ICONT=1,NCNTEX
        IENT=0
C
        DO 230 IRULE=1,NRULEX
        IF(IRULEX(5,IRULE).EQ.ICONT) THEN
           IENT=IENT+1
           WRITE(ENTRY(IENT),220) IRULE
220        FORMAT(I10)
        ENDIF
230     CONTINUE
C
        WRITE(JUNIT,240) CONTEX(ICONT),(ENTRY(I),I=1,IENT)
240     FORMAT(' Context  :',A,(T25,8A))
C
250     CONTINUE
C
        RETURN
        END
```

EXORDR

```
        SUBROUTINE EXORDR(ORDKEY,IBEG,IEND
     &
     &                                  )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE ORDERS THE ENTRIES IN THE TRIGGERED STACK
C        ACCORDING TO THE KEY SPECIFIED.  A BUBBLE SORT IS USED
```

```
C        FOR THE SORTING.  BY DEFINITION, THE TOP OF THE STACK
C        IS AT LOCATION NSTKEX
C
        CHARACTER*10 ORDKEY
C
C*************************************************************************
C
        INCLUDE '[.EXPROC]EXCOMN.INC'
C
        DIMENSION RANNUM(MRULEX)
        DATA ISEED /12345/
C
C*************************************************************************
C
        IF(ORDKEY.EQ.'PRIORITY  ') THEN
C
C.....ORDER STACK WITH HIGHEST PRIORITY AT TOP OF STACK
C
C        LOOP FOR ELEMENT TO BUBBLED
C
           DO 20 I=IBEG+1,IEND
C
C        BUBBLE THIS ENTRY UP INTO ITS PROPER PLACE
C
           DO 10 J=I,IBEG+1,-1
           JJ=ISTKEX(1,J  )
           KK=ISTKEX(1,J-1)
C
           IF(PRTYEX(JJ).GE.PRTYEX(KK)) GOTO 20
C
C        INTERCHANGE
C
           ITEMP        =ISTKEX(1,J  )
           ISTKEX(1,J  )=ISTKEX(1,J-1)
           ISTKEX(1,J-1)=ITEMP
C
10         CONTINUE
20         CONTINUE
C
       ELSEIF(ORDKEY.EQ.'PREMISES  ') THEN
C
C.....ORDER STACK WITH HIGHEST NUMBER OF PREMISES AT TOP OF STACK
C
C        LOOP FOR ELEMENT TO BUBBLED
C
           DO 40 I=IBEG+1,IEND
C
C        BUBBLE THIS ENTRY UP INTO ITS PROPER PLACE
C
           DO 30 J=I,IBEG+1,-1
           JJ=ISTKEX(1,J  )
           KK=ISTKEX(1,J-1)
C
```

```
            IF(ISTKEX(2,J).GE.ISTKEX(2,J-1)) GOTO 40
C
C         INTERCHANGE
C
            ITEMP         =ISTKEX(1,J  )
            ISTKEX(1,J  )=ISTKEX(1,J-1)
            ISTKEX(1,J-1)=ITEMP
C
30          CONTINUE
40          CONTINUE
C
          ELSEIF(ORDKEY.EQ.'M_RECENT  ') THEN
C
C.....ORDER STACK WITH MOST RECENTLY USED AT TOP OF STACK
C
C         LOOP FOR ELEMENT TO BUBBLED
C
            DO 60 I=IBEG+1,IEND
C
C         BUBBLE THIS ENTRY UP INTO ITS PROPER PLACE
C
            DO 50 J=I,IBEG+1,-1
            JJ=ISTKEX(1,J  )
            KK=ISTKEX(1,J-1)
C
            IF(IRULEX(4,JJ).GE.IRULEX(4,KK)) GOTO 60
C
C         INTERCHANGE
C
            ITEMP         =ISTKEX(1,J  )
            ISTKEX(1,J  )=ISTKEX(1,J-1)
            ISTKEX(1,J-1)=ITEMP
C
50          CONTINUE
60          CONTINUE
C
          ELSEIF(ORDKEY.EQ.'L_RECENT  ') THEN
C
C.....ORDER STACK WITH LEAST RECENTLY USED AT TOP OF STACK
C
C         LOOP FOR ELEMENT TO BUBBLED
C
            DO 80 I=IBEG+1,IEND
C
C         BUBBLE THIS ENTRY UP INTO ITS PROPER PLACE
C
            DO 70 J=I,IBEG+1,-1
            JJ=ISTKEX(1,J  )
            KK=ISTKEX(1,J-1)
C
            IF(IRULEX(4,JJ).LE.IRULEX(4,KK)) GOTO 80
C
C         INTERCHANGE
```

```
C
           ITEMP        =ISTKEX(1,J  )
           ISTKEX(1,J  )=ISTKEX(1,J-1)
           ISTKEX(1,J-1)=ITEMP
C
70         CONTINUE
80         CONTINUE
C
        ELSEIF(ORDKEY.EQ.'UNKNOWNS  ') THEN
C
C.....ORDER STACK WITH LEAST NUMBER OF UNKNOWNS AT TOP OF STACK
C
C        LOOP FOR ELEMENT TO BUBBLED
C
           DO 100 I=IBEG+1,IEND
C
C        BUBBLE THIS ENTRY UP INTO ITS PROPER PLACE
C
           DO 90 J=I,IBEG+1,-1
           JJ=ISTKEX(1,J  )
           KK=ISTKEX(1,J-1)
C
           IF(ISTKEX(3,J).LE.ISTKEX(3,J-1)) GOTO 100
C
C        INTERCHANGE
C
           ITEMP        =ISTKEX(1,J  )
           ISTKEX(1,J  )=ISTKEX(1,J-1)
           ISTKEX(1,J-1)=ITEMP
C
90         CONTINUE
100        CONTINUE
C
        ELSEIF(ORDKEY.EQ.'RANDOM    ') THEN
C
C.....ORDER STACK RANDOMLY
C
           DO 110 I=IBEG,IEND
           RANNUM(I)=RAN(ISEED)
110        CONTINUE
C
C        LOOP FOR ELEMENT TO BUBBLED
C
           DO 130 I=IBEG+1,IEND
C
C        BUBBLE THIS ENTRY UP INTO ITS PROPER PLACE
C
           DO 120 J=I,IBEG+1,-1
           JJ=ISTKEX(1,J  )
           KK=ISTKEX(1,J-1)
C
           IF(RANNUM(J).LE.RANNUM(J-1)) GOTO 130
C
```

```
C        INTERCHANGE
C
           ITEMP        =ISTKEX(1,J  )
           ISTKEX(1,J  )=ISTKEX(1,J-1)
           ISTKEX(1,J-1)=ITEMP
C
120        CONTINUE
130        CONTINUE
C
        ENDIF
C
        RETURN
        END
```

EXPARS

```
        SUBROUTINE EXPARS(INPUT,
     &
     &                    TOKEN,IFLAG)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PARSES THE STRING "INPUT" INTO ITS BLANK-
C        DELIMITED TOKENS, "TOKEN(I)".  IF THE FIRST CHARACTER OF A
C        TOKEN IS "%", THEN SET FLAG(I) TO -1, OTHERWISE 1.
C        FIVE TOKEN ARE CONSIDERED
C
        CHARACTER*(*) INPUT,TOKEN
C
        DIMENSION TOKEN(5),IFLAG(5)
C
C**************************************************************************
C
C     INITIALIZE THE TOKEN STRINGS AND THE NUMBER OF TOKENS FOUND
C
        TOKEN(1)='          '
        TOKEN(2)='          '
        TOKEN(3)='          '
        TOKEN(4)='          '
        TOKEN(5)='          '
        IFLAG(1)=+1
        IFLAG(2)=+1
        IFLAG(3)=+1
        IFLAG(4)=+1
        IFLAG(5)=+1
C
        NTOKEN=0
C
C     INITIALIZE THE CURRENT RANGE TO BE THE WHOLE INPUT STRING UP TO
C        BUT NOT INCLUDING THE FIRST ";"
```

```
C
        IBEG=1
        IEND=INDEX((INPUT//';'),';')-1
C
C     SEARCH FOR THE FIRST NON-BLANK CHARACTER IN THE CURRENT RANGE
C
10      DO 20 I=IBEG,IEND
        IF(INPUT(I:I).NE.' ') THEN
           JBEG=I
           GOTO 30
        ENDIF
20      CONTINUE
C
C     IF NO MORE NON-BLANK CHARACTERS WERE FOUND, THEN EXIT
C
        RETURN
C
C     SEARCH FOR THE END OF THIS TOKEN
C
30      DO 40 I=JBEG+1,IEND
        IF(INPUT(I:I).EQ.' ') THEN
           JEND=I-1
           GOTO 50
        ENDIF
40      CONTINUE
C
C     END OF THIS TOKEN WAS NOT FOUND, SO USE THE END OF THE INPUT
C       STRING
C
        JEND=IEND
C
C     IF THIS TOKEN BEGINS WITH A '%', THEN SET THE ASSOCIATED FLAG
C       AND STRIP OFF THE '%'
C
50      NTOKEN=NTOKEN+1
        IF(INPUT(JBEG:JBEG).EQ.'%') THEN
           IFLAG(NTOKEN)=-1
           JBEG=JBEG+1
        ELSE
           IFLAG(NTOKEN)=+1
        ENDIF
C
C     PUT THE FOUND STRING INTO THE TOKEN VARIABLE
C
        TOKEN(NTOKEN)=INPUT(JBEG:JEND)
C
C     RESTRICT THE SPAN OF THE SEARCH FOR REMAINING TOKENS AND GO BACK
C       TO SEARCH FOR REST OF TOKENS IF NOT AT THE END OF THE INPUT
C       STRING
C
        IBEG=JEND+1
        IF(IBEG.LT.IEND .AND. NTOKEN.LT.6) GOTO 10
C
```

```
      RETURN
      END
```

EXREAD

```
      SUBROUTINE EXREAD(IUNIT
     &
     &                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS EXPERT SYSTEM INPUT FROM IUNIT AND
C       COMPILES IT.  ERROR CHECKING IS ONLY FOR VALID STATEMENT
C       ORDERING
C
C*********************************************************************
C
      INCLUDE '[.EXPROC]EXCOMN.INC'
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
      CHARACTER*10 PTYPE,STYPE,ONOFF,ARG1,ARG2,ARG3,COMP,PROC,TOKEN(5)
C
      CHARACTER*80 INSTRG
C
      DIMENSION IFLAG(5)
C
C*********************************************************************
C
C     INITIALIZE ALL THE TABLES
C
      NVAREX=0
      NRULEX=0
      NPRMEX=0
      NACTEX=0
      NCNTEX=0
C
      IRULEX(1,1)=1
      IRULEX(2,1)=1
C
C     SET UP THE VALUE OF UNKNOWN VARAIBLES
C
      UKNOEX=-1.2345678E+38
C
C     SET UP THE STANDARD VARIABLES AND INITIALIZE THEIR VALUES TO 0
C
      CALL EXVLST('          ',IDUM)
      CALL EXVLST('*         ',IDUM)
      CALL EXVLST('#1        ',IDUM)
      CALL EXVLST('#2        ',IDUM)
```

```
        CALL EXVLST('#3         ',IDUM)
        CALL EXVLST('#4         ',IDUM)
        CALL EXVLST('UNKNOWN    ',IDUM)
C
        VALUEX(1)=0.00
        VALUEX(2)=0.00
        VALUEX(3)=0.00
        VALUEX(4)=0.00
        VALUEX(5)=0.00
        VALUEX(6)=0.00
C
C     SET DUMMY NAME FOR NO CONTEXT AND PUT 'INITIAL' INTO TABLE
C
        CONTEX(0)='-- NONE --'
        CALL EXCLST('initial    ',IDUM)
C
C     INITIALIZE THE STRATEGY VARIABLES
C
        DYMCEX='OFF       '
        PRODEX='OFF       '
        ORD1EX='          '
        ORD2EX='          '
        ORD3EX='          '
        ITRCEX=0
C
C     PTYPE CONTAINS THE PREVIOUS STATEMENT TYPE
C
        PTYPE='PROLOG     '
        ILINE=0
C
        OPEN(UNIT=IUNIT,STATUS='OLD')
C
C     READ AND PARSE THE INPUT STRING
C
10      READ(IUNIT,20) INSTRG
20      FORMAT(A80)
C
        ILINE=ILINE+1
        IF(ILINE.EQ.1) WRITE(JPRINT,25) INSTRG
25      FORMAT(' EXPROC INPUT HAS BEEN READ:'/
     &         11X,A                          )
C
        CALL EXPARS(INSTRG,TOKEN,IFLAG)
C
        STYPE=TOKEN(1)
C
C     BRANCH OUT DEPENDING ON CURRENT STATEMENT TYPE:
C
C-------COMMENT STATEMENT
C
        IF(STYPE.EQ.'           ' .OR. STYPE.EQ.'COMMENT    ') THEN
           CONTINUE
C
```

```
C-------DYNAMIC STATEMENT
C
      ELSEIF(STYPE.EQ.'DYNAMIC   ') THEN
C
         IF(PTYPE.NE.'PROLOG    ') THEN
            CALL UTERROR(1,REAL(ILINE),0.0)
         ENDIF
C
C     MAKE SURE THAT THE ARGUMENT IS VALID
C
         ONOFF=TOKEN(2)
C
         IF(ONOFF.EQ.'ON         ' .OR. ONOFF.EQ.'OFF        ') THEN
            DYNCEX=ONOFF
         ELSE
            CALL UTERROR(2,REAL(ILINE),0.0)
         ENDIF
C
C-------PRODCTN STATEMENT
C
      ELSEIF(STYPE.EQ.'PRODCTN   ') THEN
C
         IF(PTYPE.NE.'PROLOG    ') THEN
            CALL UTERROR(3,REAL(ILINE),0.0)
         ENDIF
C
C     MAKE SURE THAT THE ARGUMENT IS VALID
C
         ONOFF=TOKEN(2)
C
         IF(ONOFF.EQ.'ON         ' .OR. ONOFF.EQ.'OFF        ') THEN
            PRODEX=ONOFF
         ELSE
            CALL UTERROR(4,REAL(ILINE),0.0)
         ENDIF
C
C-------ORDER STATEMENT
C
      ELSEIF(STYPE.EQ.'ORDER     ') THEN
C
         IF(PTYPE.NE.'PROLOG    ') THEN
            CALL UTERROR(5,REAL(ILINE),0.0)
         ENDIF
C
C     MAKE SURE THAT THE ARGUMENT IS VALID
C
         ARG1=TOKEN(2)
         ARG2=TOKEN(3)
         ARG3=TOKEN(4)
C
C     ...MAJOR ORDERING
C
         IF(ARG1.EQ.'           ' .OR. ARG1.EQ.'PRIORITY  ' .OR.
```

```
     &        ARG1.EQ.'PREMISES  ' .OR. ARG1.EQ.'M_RECENT  ' .OR.
     &        ARG1.EQ.'L_RECENT  ' .OR. ARG1.EQ.'UNKNOWNS  ' .OR.
     &        ARG31.EQ.'RANDOM    '                              ) THEN
              ORD1EX=ARG1
           ELSE
              CALL UTEROR(6,REAL(ILINE),0.0)
           ENDIF
C
C       ...SEMI-MINOR ORDERING
C
           IF(ARG2.EQ.'          ' .OR. ARG2.EQ.'PRIORITY  ' .OR.
     &        ARG2.EQ.'PREMISES  ' .OR. ARG2.EQ.'M_RECENT  ' .OR.
     &        ARG2.EQ.'L_RECENT  ' .OR. ARG2.EQ.'UNKNOWNS  ' .OR.
     &        ARG1.EQ.'RANDOM    '                              ) THEN
              ORD2EX=ARG2
           ELSE
              CALL UTEROR(7,REAL(ILINE),0.0)
           ENDIF
C
C       ...MINOR ORDERING
C
           IF(ARG3.EQ.'          ' .OR. ARG3.EQ.'PRIORITY  ' .OR.
     &        ARG3.EQ.'PREMISES  ' .OR. ARG3.EQ.'M_RECENT  ' .OR.
     &        ARG3.EQ.'L_RECENT  ' .OR. ARG3.EQ.'UNKNOWNS  ' .OR.
     &        ARG1.EQ.'RANDOM    '                              ) THEN
              ORD3EX=ARG3
           ELSE
              CALL UTEROR(8,REAL(ILINE),0.0)
           ENDIF
C
C-------TRACE STATEMENT
C
         ELSEIF(STYPE.EQ.'TRACE     ') THEN
C
           IF(PTYPE.NE.'PROLOG    ') THEN
              CALL UTEROR(9,REAL(ILINE),0.0)
           ENDIF
C
C       MAKE SURE THAT THE ARGUMENT IS VALID
C
           READ(TOKEN(2),*) ITRCEX
C
C-------INITIAL STATEMENT
C
         ELSEIF(STYPE.EQ.'INITIAL   ') THEN
C
           IF(PTYPE.NE.'PROLOG    ') THEN
              CALL UTEROR(10,REAL(ILINE),0.0)
           ENDIF
C
C      FIND/PUT VARIABLE IN VARIABLE LIST
C
           CALL EXVLST(TOKEN(2),IVAR)
```

```
C
C     SET ITS VALUE
C
          READ(TOKEN(3),FMT='(G10.0)') VAR
          VALUEX(IVAR)=VAR
C
C-------RULE STATEMENT
C
        ELSEIF(STYPE.EQ.'RULE      ') THEN
C
          IF(PTYPE.NE.'PROLOG    ' .AND. PTYPE.NE.'ACTION    ') THEN
            CALL UTEROR(11,REAL(ILINE),0.0)
          ENDIF
C
C     CREATE NEW RULE
C
          IF(NRULEX.GE.MRULEX) THEN
            CALL UTEROR(12,REAL(ILINE),REAL(NRULEX))
          ENDIF
C
          NRULEX=NRULEX+1
C
C     STORE THE RULE NAME, STORE DEFAULT PRIORITY AND CONTEXT, AND
C        INITIALIZE TIME STAMP
C
          RULEEX(  NRULEX)=INSTRG(11:50)
          PRTYEX(  NRULEX)=0.00
          IRULEX(4,NRULEX)=0
          IRULEX(5,NRULEX)=0
C
C     PRE-SET POINTER TO FIRST PREMISE IN NEXT RULE
C
          IRULEX(1,NRULEX+1)=NPREMEX+1
C
          PTYPE='TOPRULE   '
C
C-------PRIORITY STATEMENT
C
        ELSEIF(STYPE.EQ.'PRIORITY  ') THEN
C
          IF(PTYPE.NE.'TOPRULE   ') THEN
            CALL UTEROR(13,REAL(ILINE),0.0)
          ENDIF
C
C     SET THE PRIORITY
C
          READ(TOKEN(2),FMT='(G10.0)') PRTYEX(NRULEX)
C
C-------CONTEXT STATEMENT
C
        ELSEIF(STYPE.EQ.'CONTEXT   ') THEN
C
          IF(PTYPE.NE.'TOPRULE   ') THEN
```

```
            CALL UTEROR(14,REAL(ILINE),0.0)
         ENDIF
C
C     FIND THE CONTEXT NUMBER OF THIS CONTEXT
C
         CALL EXCLST(TOKEN(2),ITOKEN)
         IRULEX(5,NRULEX)=ITOKEN
C
C-------IF STATEMENT
C
       ELSEIF(STYPE.EQ.'IF        ') THEN
C
         IF(PTYPE.NE.'TOPRULE   ') THEN
            CALL UTEROR(15,REAL(ILINE),0.0)
         ENDIF
C
C     CREATE NEW PREMISE TABLE ENTRY
C
         IF(NPRMEX.GE.MPRMEX) THEN
            CALL UTEROR(16,REAL(ILINE),REAL(NPRMEX))
         ENDIF
C
         NPRMEX=NPRMEX+1
C
C     FIND/PUT THE ARGUMENTS IN THE VARIABLE LIST
C
         CALL EXVLST(TOKEN(2),IARG1)
         CALL EXVLST(TOKEN(4),IARG2)
C
C     PUT THIS PREMISE INTO THE TABLE
C
         COMPEX(  NPRMEX)=TOKEN(3)
         IPRMEX(1,NPRMEX)=IARG1
         IPRMEX(2,NPRMEX)=IARG2
C
C     PRE-SET POINTER TO FIRST PREMISE IN NEXT RULE
C
         IRULEX(1,NRULEX+1)=NPRMEX+1
C
         PTYPE='PREMISE   '
C
C-------ANDIF STATEMENT
C
       ELSEIF(STYPE.EQ.'ANDIF     ') THEN
C
         IF(PTYPE.NE.'PREMISE   ') THEN
            CALL UTEROR(17,REAL(ILINE),0.0)
         ENDIF
C
C     CREATE NEW PREMISE TABLE ENTRY
C
         IF(NPRMEX.GE.MPRMEX) THEN
            CALL UTEROR(18,REAL(ILINE),REAL(NPRMEX))
```

```
            ENDIF
C
            NPRMEX=NPRMEX+1
C
C     FIND/PUT THE ARGUMENTS IN THE VARIABLE LIST
C
            CALL EXVLST(TOKEN(2),IARG1)
            CALL EXVLST(TOKEN(4),IARG2)
C
C     PUT THIS PREMISE INTO THE TABLE
C
            COMPEX(  NPRMEX)=TOKEN(3)
            IPRMEX(1,NPRMEX)=IARG1
            IPRMEX(2,NPRMEX)=IARG2
C
C     PRE-SET POINTER TO FIRST PREMISE IN NEXT RULE
C
            IRULEX(1,NRULEX+1)=NPRMEX+1
C
C-------THEN STATEMENT
C
          ELSEIF(STYPE.EQ.'THEN      ') THEN
C
            IF(PTYPE.NE.'PREMISE   ') THEN
               CALL UTEROR(19,REAL(ILINE),0.0)
            ENDIF
C
C     CREATE NEW ACTION TABLE ENTRY
C
            IF(NACTEX.GE.MACTEX) THEN
              CALL UTEROR(20,REAL(ILINE),REAL(NACTEX))
            ENDIF
C
            NACTEX=NACTEX+1
C
C     FIND/PUT THE ARGUMENTS IN THE VARIABLE LIST
C
            IF(TOKEN(2).EQ.'CONSIDER  ') THEN
               CALL EXCLST(TOKEN(3),IARG1)
            ELSE
               CALL EXVLST(TOKEN(3),IARG1)
            ENDIF
C
            CALL EXVLST(TOKEN(4),IARG2)
            CALL EXVLST(TOKEN(5),IARG3)
C
C     PUT THIS ACTION INTO THE TABLE
C
            ACTNEX(  NACTEX)=TOKEN(2)
            IACTEX(1,NACTEX)=IFLAG(3)*IARG1
            IACTEX(2,NACTEX)=IFLAG(4)*IARG2
            IACTEX(3,NACTEX)=IFLAG(5)*IARG3
C
```

```
C     PRE-SET POINTER TO FIRST ACTION IN NEXT RULE
C
           IRULEX(2,NRULEX+1)=NACTEX+1
C
           PTYPE='ACTION    '
C
C-------ANDTHEN STATEMENT
C
        ELSEIF(STYPE.EQ.'ANDTHEN   ') THEN
C
           IF(PTYPE.NE.'ACTION    ') THEN
              CALL UTEROR(21,REAL(ILINE),0.0)
           ENDIF
C
C     CREATE NEW ACTION TABLE ENTRY
C
           IF(NACTEX.GE.MACTEX) THEN
             CALL UTEROR(22,REAL(ILINE),REAL(NACTEX))
           ENDIF
C
           NACTEX=NACTEX+1
C
C     FIND/PUT THE ARGUMENTS IN THE VARIABLE LIST
C
           IF(TOKEN(2).EQ.'CONSIDER  ') THEN
              CALL EXCLST(TOKEN(3),IARG1)
           ELSE
              CALL EXVLST(TOKEN(3),IARG1)
           ENDIF
C
           CALL EXVLST(TOKEN(4),IARG2)
           CALL EXVLST(TOKEN(5),IARG3)
C
C     PUT THIS ACTION INTO THE TABLE
C
           ACTNEX( NACTEX)=TOKEN(2)
           IACTEX(1,NACTEX)=IFLAG(3)*IARG1
           IACTEX(2,NACTEX)=IFLAG(4)*IARG2
           IACTEX(3,NACTEX)=IFLAG(5)*IARG3
C
C     PRE-SET POINTER TO FIRST ACTION IN NEXT RULE
C
           IRULEX(2,NRULEX+1)=NACTEX+1
C
C-------END STATEMENT
C
        ELSEIF(STYPE.EQ.'END       ') THEN
           CLOSE(UNIT=IUNIT)
C
           RETURN
C
        ELSE
           CALL UTEROR(23,REAL(ILINE),0.0)
```

```
C
        ENDIF
C
        GOTO 10
C
        END
```

EXVLST

```
        SUBROUTINE EXVLST(LABL,
     &
     &                    IVAR)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE LOOKS FOR THE VARIABLE LABEL 'LABL' IN THE
C        VARIABLE LIST.  IF IT FINDS IT, IT RETURNS THE VARIABLE NUMBER.
C        IF NOT FOUND IN THE LIST, THEN IT IS ADDED TO END OF THE LIST
C        AND ITS NUMBER IS RETURNED
C
        CHARACTER*10 LABL
C
C*************************************************************************
C
        INCLUDE '[.EXPROC]EXCOMN.INC'
C
C*************************************************************************
C
C     LOOK FOR THIS VARIABLE IN THE LIST
C
        DO 10 I=1,NVAREX
        IF(LABL.NE.LABLEX(I)) GOTO 10
C
        IVAR=I
        RETURN
C
10      CONTINUE
C
C     VARIABLE IS NOT IN LIST, SO ADD TO END OF LIST
C
        IF(NVAREX.GE.MVAREX) THEN
           CALL UTEROR(1,REAL(NVAREX),REAL(MVAREX))
        ENDIF
C
        NVAREX=NVAREX+1
        LABLEX(NVAREX)=LABL
C
C     IF VARIABLE NAME STARTS WITH + OR -, USE ITS VALUE DIRECTLY
C
        IF(LABL(1:1).EQ.'+' .OR. LABL(1:1).EQ.'-') THEN
```

```
            READ(LABL,FMT='(G10.0)') VALUEX(NVAREX)
         ELSE
            VALUEX(NVAREX)=UKNOEX
         ENDIF
C
         IVAR=NVAREX
         RETURN
C
         END
```

B.3 EULR2D — Two-Dimensional Euler Integration Scheme

E2COMN

```
C-----SCALARS
C     CFLE2        CFL NUMBER FOR EULER INTEGRATION
C     AMUE2(1)     RELAXATION PARAMETER FOR SMOOTHING EQUATION
C     AMUE2(2)     COEFFICIENT OF LAPLACIAN IN SMOOTHING EQUATION
C     AMUE2(3)     BACKGROUND SMOOTHING LEVEL
C     AMUE2(4)     PRESSURE GRADIENT MULTIPLIER FOR SMOOTHING
C     U1RFE2       REFERENCE VALUE FOR 1ST DEPENDENT VARIABLE (RHO)
C     U2RFE2       REFERENCE VALUE FOR 2ND DEPENDENT VARIABLE (RHO*U)
C     U3RFE2       REFERENCE VALUE FOR 3RD DEPENDENT VARIABLE (RHO*V)
C     U4RFE2       REFERENCE VALUE FOR 4TH DEPENDENT VARIABLE (E)
C     GMOOE2       RATIO OF SPECIFIC HEATS
C     GMO1E2       GMOOE2*(GMOOE2-1.0)
C     GMO2E2       0.5*(GMOOE2-3.0)
C     GMO3E2       GMOOE2-1.0
C     GMO4E2       0.5*(GMOOE2-1.0)
C     GMO5E2       GMOOE2-3.0
C     GMO6E2       1.5*(GMOOE2-1.0)
C     GMO7E2       0.5*(GMOOE2+1.0)
C     GMO8E2       GMOOE2+1.0
C     GMO9E2       (GMOOE2*2.0)/(GMOOE2+1.0)
C     GM10E2       (GMOOE2-1.0)/(GMOOE2+1.0)
C     GM11E2       GMOOE2/(1.0-GMOOE2)
C     TOL1E2       CONVERGENCE TOLERANCE ON DU2ME2
C     TOL2E2       CONVERGENCE TOLERANCE ON DELTA CLFTE2
C     TOL3E2       CONVERGENCE TOLERANCE ON DELTA CDRGE2
C     NWNDE2       NUMBER OF ITERATIONS IN CONVERGENCE WINDOW
C     IHSTE2       UNIT NUMBER OF CONVERGENCE FILE
C     ITERE2       ITERATION NUMBER
C     TIMEE2       CPU TIME AT LAST ITERATION
C     CLFTE2       LIFT COEFFICIENT
C     CDRGE2       DRAG COEFFICIENT
C     CHRDE2       CHORD USE IN CLFTE2 AND CDRGE2 CALCULATIONS
```

```
C                 IF CHRDE2=0, THEN DO NOT CALCULATE CLFTE2 AND CDRGE2
C     DU2ME2      MAXIMUM CHANGE IN DPEN2 DURING LAST ITERATION
C     IDU2E2      NODE NUMBER OF NODE WITH CHANGE=DU2ME2
C     DU2AE2      AVERAGE CHANGE IN DPEN2 DURING LAST ITERATION
C     DU2RE2      RMS CHANGE IN DPEN2 DURING LAST ITERATION
C     SMINE2      MINIMUM VALUE OF DPEN5 DURING LAST ITERATION
C     SMAXE2      MAXIMUM VALUE OF DPEN2 DURING LAST ITERATION
C     TMO1E2      CPU TIME SPENT IN E2CONV
C     TMO2E2      CPU TIME SPENT IN E2WGHT
C     TMO3E2      CPU TIME SPENT IN E2SMTH
C     TMO4E2      CPU TIME SPENT IN E2TRAN
C     TMO5E2      CPU TIME SPENT IN E2ZERO
C     TMO6E2      CPU TIME SPENT IN E2DIST
C     TMO7E2      CPU TIME SPENT IN E2BCON
C     TMO8E2      CPU TIME SPENT IN E2INTP
C     TMO9E2      CPU TIME SPENT IN E2UPDT
C     TM1OE2      CPU TIME SPENT IN E2BFIX
C
C-----CHANGE ARRAY
C     CHNGE2(I,INODE)     INODE=1,NNODG2
C           I=1   CHANGE IN 1ST DEPENDENT VARIABLE
C           I=2   CHANGE IN 2ND DEPENDENT VARIABLE
C           I=3   CHANGE IN 3RD DEPENDENT VARIABLE
C           I=4   CHANGE IN 4TH DEPENDENT VARIABLE
C
C-----E2BCON INDEX AND CONTROL VECTORS
C     HDESE2      FREE-STREAM ENTHALPY FUNCTION
C     SDESE2      FREE-STREAM ENTROPY FUNCTION
C     CBCE2       FREE-STREAM SPEED OF SOUND
C     BCINE2(4,3) SUBSONIC INLET INFLUENCE MATRIX
C     BCEXE2(5,)  SUBSONIC EXIT  INFLUENCE MATRIX
C     VBCNE2(I,IBOUND)  IBOUND=1,NBNDG2
C                 BOUNDARY CONDITION INFO WHICH IS TYPE DEPENDENT
C
C-----E2DIST INDEX AND CONTROL VECTORS
C     VDISE2(I,ICELL)   ICELL=1,NCELG2
C           I=1   SOUTH FACE FLUX MULTIPLIER
C           I=2   EAST  FACE FLUX MULTIPLIER
C           I=3   NORTH FACE FLUX MULTIPLIER
C           I=4   WEST  FACE FLUX MULTIPLIER
C
C-----E2FORC CONTROL AND INDEX VECTORS
C     IFORE2(I,IFORC)   IFORC=1,NFORE2
C           I=1   LEFT  NODE FOR FORCE COEFFICIENT CONTRIBUTION
C           I=2   RIGHT NODE FOR FORCE COEFFICIENT CONTRIBUTION
C
C-----E2INTP CONTROL AND INDEX VECTORS
C     IITPE2(I,INODE)   INODE=1,NNODG2
C                 INTERPAOATION TRIPLET DESCRIPTION
C                    DPEN(I=2)=(DPEN(I=1)+DPEN(I=3))/2
C     LITPE2(I,ILEVEL)   ILEVEL=-MLVLG2,MLVLG2
C           I=1   FIRST TRIPLET (INTERIOR SIDE)
C           I=2   LAST  TRIPLET (INTERIOR SIDE)
```

```
C            I=3   FIRST TRIPLET (CENTER)
C            I=4   LAST  TRIPLET (CENTER)
C            I=5   FIRST TRIPLET (EXTERIOR SIDE)
C            I=6   LAST  TRIPLET (EXTERIOR SIDE)
C
C-----E2UPDT CONTROL VECTOR
C     CUPDE2(ILEVEL,INODE)   ILEVEL=-MLVLG2,MLVLG2, INODE=1,NNODG2
C                  SWITCH FOR UPDATING NODE AT THIS LEVEL
C                  NODE IS UPDATED ON A LEVEL IF AND(CUPDE2,MASK).NE.0
C                  WHERE MASK=2**(ILEVEL+MLVLG2)
C
C-----E2ZERO CONTROL VECTOR
C     CZROE2(ILEVEL,INODE)   ILEVEL=-MLVLG2,MLVLG2, INODE=1,NNODG2
C                  SWITCH FOR ZEROING NODE AT THIS LEVEL
C                  NODE IS ZEROED ON A LEVEL IF AND(CUPDE2,MASK).NE.0
C                  WHERE MASK=2**(ILEVEL+MLVLG2)
C
      PARAMETER (MDUMO1=                    1,
     &           MDUMO2=          4*MNODG2+1,
     &           MDUMO3=4*MCELG2+4*MNODG2+1,
     &           MDUMO4=4*MCELG2+7*MNODG2+1,
     &           MDUMO5=4*MCELG2+8*MNODG2+1)
C
      DIMENSION CHNGE2(4,MNODG2),
     &          VDISE2(4,MCELG2),
     &          IITPE2(3,MNODG2),
     &          CUPDE2(  MNODG2),
     &          CZROE2(  MNODG2)
C
      INTEGER CUPDE2,CZROE2
C
      EQUIVALENCE (DUMMG2(MDUMO1),CHNGE2(1,1)),
     &            (DUMMG2(MDUMO2),VDISE2(1,1)),
     &            (DUMMG2(MDUMO3),IITPE2(1,1)),
     &            (DUMMG2(MDUMO4),CUPDE2(  1)),
     &            (DUMMG2(MDUMO5),CZROE2(  1))
C
      COMMON /E2COMN/ CFLE2 ,AMUE2(4),
     &                U1RFE2,U2RFE2,U3RFE2,U4RFE2,
     &                GMOOE2,GMO1E2,GMO2E2,GMO3E2,GMO4E2,GMO5E2,
     &                GMO6E2,GMO7E2,GMO8E2,GMO9E2,GM10E2,GM11E2,
     &                TOL1E2,TOL2E2,TOL3E2,NWNDE2,
     &                IHSTE2,ITERE2,TIMEE2,CLFTE2,CDRGE2,CHRDE2,
     &                DU2ME2,IDU2E2,DU2AE2,DU2RE2,SMINE2,SMAXE2,
     &                TMO1E2,TMO2E2,TMO3E2,TMO4E2,TMO5E2,
     &                TMO6E2,TMO7E2,TMO8E2,TMO9E2,TM10E2,
C
     &                HDESE2,SDESE2,CBCE2,BCINE2(4,3),BCEXE2(6,3),
     &                VBCNE2(4,MBNDG2),
C
     &                IFORE2(2,MBNDG2),NFORE2,
C
     &                LITPE2(6,-MLVLG2:MLVLG2)
```

E2BCON

```
      SUBROUTINE E2BCON(IMGL
     &
     &                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE APPLIES THE BOUNDARY CONDITIONS.
C
C     BOUNDARY CONDITION TYPES:
C        1  RADIATION
C        2  DIRICHLET (HOLD INITIAL VALUES)
C        3  SOLID WALL (SIMPLE WAVE CONDITION)
C        4  FREE STREAM (CHARACTERISTICS)
C        5  CORNER - FLOW IN EAST/WEST DIRECTION
C        6  CORNER - FLOW IN NORTH/SOUTH DIRECTION
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
C     CALCULATE AUXILIARY CONSTANTS USED IN VORTEX COMPUTATION
C
      GAMM1=(GMOOE2-1.0)/GMOOE2
      GAMM2=1.0/GAMM1
      GAMM3=1.0/GMOOE2
      GAMM4=0.5*GAMM1
C
C     APPLY BOUNDARY CONDITIONS AT EACH BOUNDARY NODE
C
      DO 90 IBOUND=1,NBNDG2
C
C     BRANCH OUT ACCORDING TO TYPE
C
      INODE =IBNDG2(1,IBOUND)
      ITYPE =IBNDG2(5,IBOUND)
      ILEVEL=IBNDG2(6,IBOUND)
C
      IF(ILEVEL.LT.IMGL) GOTO 90
C
      GOTO (10,20,30,40,50,50), ITYPE
      GOTO  90
C
C*****RADIATION CONDITION
```

```
C
10      GOTO 90
C
C*****DIRICHLET CONDITION
C
20      CHNGE2(1,INODE)=0.0
        CHNGE2(2,INODE)=0.0
        CHNGE2(3,INODE)=0.0
        CHNGE2(4,INODE)=0.0
C
        GOTO 90
C
C*****SOLID WALL
C
30      COSWAL=VBCNE2(1,IBOUND)
        SINWAL=VBCNE2(2,IBOUND)
C
C     CALCULATE THE TANGENTIAL COMPONENT OF THE CHANGE IN MOMENTUM AND
C        BREAK IT INTO ITS COMPONENTS
C
        DRQ=CHNGE2(2,INODE)*COSWAL+CHNGE2(3,INODE)*SINWAL
C
        CHNGE2(2,INODE)=DRQ*COSWAL
        CHNGE2(3,INODE)=DRQ*SINWAL
C
        GOTO 90
C
C*****FREE STREAM, SPECIFIED CONDITIONS AND CHARACTERISTICS
C
C     COMPUTE PROJECTION OF VELOCITY ON INWARD NORMAL VECTOR TO
C        DETERMINE IF FLOW IS ENTERING OR EXITING
C
40      COSBND=VBCNE2(1,IBOUND)
        SINBND=VBCNE2(2,IBOUND)
        UINOUT=(-SINBND*DPENG2(2,INODE)+COSBND*DPENG2(3,INODE))
     &                    /SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)
        UMAG=SQRT(DPENG2(2,INODE)**2+DPENG2(3 INODE)**2)
     &                                                  /DPENG2(1,INODE)
C
C     BRANCH OUT DPENDING ON WHETHER THE VELOCITY NORMAL TO THE
C        BOUNDARY IS ENTERING OR EXITING, AND WHETHER THE FLOW
C        IS SUBSONIC OF SUPERSONIC
C
        IF(UINOUT.LT.-0.01 .AND. UMAG.GT.CBCE2) THEN
C
C.........SUPERSONIC OUTFLOW...ALL CHARACTERISTIC WAVES FROM INSIDE
C          THE DOMAIN, SO LEAVE CHANGES PREDICTED FROM DISTRIBUTION
C          UNALTERED
C
           CONTINUE
C
        ELSEIF(UINOUT.LT.-0.01) THEN
C
```

```
C.........SUBSONIC OUTFLOW...THREE CHARACTERISTICS WAVES FROM INSIDE
C         AND ONE FROM OUTSIDE THE DOMAIN.  PICK THE INCOMING
C         CHARACTERISTIC WAVE SUCH THAT THE STATIC PRESSURE IS HELD
C
C         COMPUTE THE DESIRED PRESSURE WHICH IS THAT GENERATED BY
C         THE PRESCRIBED UNIFORM FLOW + VORTEX
C
          VORCOS=VBCNE2(3,IBOUND)
          VORSIN=VBCNE2(4,IBOUND)
C
          RINF=BONDG2(5,IBOUND)
          UINF=BONDG2(6,IBOUND)/RINF
          VINF=BCNDG2(7,IBOUND)/RINF
          QINF=UINF*UINF+VINF*VINF
          PINF=GMO3E2*(BONDG2(8,IBOUND)-0.50*RINF*QINF)
C
          UFS=UINF+VORSIN*CLFTE2
          VFS=VINF-VORCOS*CLFTE2
          QFS=UFS*UFS+VFS*VFS
C
          PDES=(PINF**GAMM1+GAMM4*(RINF/PINF**GAMM3)*(QINF-QFS))**GAMM2
C
C         COMPUTE FLOW CONDITIONS AT THE OLD TIME
C
          ROLD=DPENG2(1,INODE)
          QOLD=SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)
          EOLD=DPENG2(4,INODE)
          POLD=GMO3E2*(EOLD-0.5*QOLD*QOLD/ROLD)
C
C         COMPUTE SINE AND COSINE OF FLOW DIRECTION SINCE BOUNDARY
C         CONDITIONS WILL BE APPLIED IN INTRINSIC COORDINATES
C
          COSFLO=DPENG2(2,INODE)/QOLD
          SINFLO=DPENG2(3,INODE)/QOLD
C
C         COPMUTE CHANGES PREDICTED BY DISTRIBUTION FORMULAE IN
C         INTRINSIC COORDINATES
C
          DR =CHNGE2(1,INODE)
          DQS=CHNGE2(2,INODE)*COSFLO+CHNGE2(3,INODE)*SINFLO
          DQN=CHNGE2(3,INODE)*COSFLO-CHNGE2(2,INODE)*SINFLO
          DE =CHNGE2(4,INODE)
C
C         CALCULATE THE FORCING FUNCTIONS AT THIS POINT
C
          DC4=BCEXE2(4,1)*DR +BCEXE2(4,2)*DQS+BCEXE2(4,3)*DE
          DC2=BCEXE2(5,1)*DR +BCEXE2(5,2)*DQS+BCEXE2(5,3)*DE
          DP =PDES-POLD
C
C         CALCULATE THE CHANGES IN THE DEPENDENT VARIABLES IN THE
C         INTRINSIC COORDINATE FRAME
C
          DR =BCEXE2(1,1)*DC4+BCEXE2(1,2)*DC2+BCEXE2(1,3)*DP
```

```
            DQS=BCEXE2(2,1)*DC4+BCEXE2(2,2)*DC2+BCEXE2(2,3)*DP
            DE =BCEXE2(3,1)*DC4+BCEXE2(3,2)*DC2+BCEXE2(3,3)*DP
C
C           STORE THE CHANGES IN THE CARTESIAN FRAME
C
            CHNGE2(1,INODE)=DR
            CHNGE2(2,INODE)=DQS*COSFLO-DQN*SINFLO
            CHNGE2(3,INODE)=DQN*COSFLO+DQS*SINFLO
            CHNGE2(4,INODE)=DE
C
          ELSEIF(UMAG.LT.CBCE2) THEN
C
C.....SUBSONIC INFLOW...ONE CHARACTERISTIC WAVE FROM INSIDE AND
C           THREE FROM OUTSIDE THE DOMAIN.  PICK THE INCOMING
C           CHARACTERISTIC WAVE SUCH THAT ANGLE, ENTROPY, AND STAGNATION
C           ENTHALPY ARE HELD
C
C           COMPUTE THE DESIRED FLOW ANGLE WHICH IS THAT GENERATED BY
C           THE PRESCRIBED UNIFORM FLOW + VORTEX
C
            UINF  =BONDG2(6,IBOUND)/BONDG2(5,IBOUND)
            VINF  =BONDG2(7,IBOUND)/BONDG2(5,IBOUND)
            VORCOS=VBCNE2(3,IBOUND)
            VORSIN=VBCNE2(4,IBOUND)
C
            TANDES=(VINF-VORCOS*CLFTE2)/(UINF+VORSIN*CLFTE2)
C
C           COMPUTE FLOW CONDITIONS AT THE OLD TIME
C
            ROLD=DPENG2(1,INODE)
            QOLD=SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)
            EOLD=DPENG2(4,INODE)
            POLD=GMO3E2*(EOLD-0.5*QOLD*QOLD/ROLD)
C
C           COMPUTE SINE, COSINE, AND TANGENT OF FLOW DIRECTION SINCE
C           BOUNDARY CONDITIONS WILL BE APPLIED IN INTRINSIC COORDINATES
C
            COSFLO=DPENG2(2,INODE)/QOLD
            SINFLO=DPENG2(3,INODE)/QOLD
            TANFLO=DPENG2(3,INODE)/DPENG2(2,INODE)
C
C           COPMUTE CHANGES PREDICTED BY DISTRIBUTION FORMULAE IN
C           INTRINSIC COORDINATES
C
            DR =CHNGE2(1,INODE)
            DQS=CHNGE2(2,INODE)*COSFLO+CHNGE2(3,INODE)*SINFLO
            DQN=CHNGE2(3,INODE)*COSFLO-CHNGE2(2,INODE)*SINFLO
            DE =CHNGE2(4,INODE)
C
C           CALCULATE THE FORCING FUNCTIONS AT THIS POINT
C
            DH =HDESE2-(0.5*(QOLD/ROLD)**2-GM11E2*POLD/ROLD)
            DS =(SDESE2-POLD/ROLD**GMOOE2)
```

```
            DC1=BCINE2(4,1)*DR +BCINE2(4,2)*DQS+BCINE2(4,3)*DE
C
C           CALCULATE THE CHANGES IN THE DEPENDENT VARIABLES IN THE
C           INTRINSIC COORDINATE FRAME
C
            DR =BCINE2(1,1)*DR +BCINE2(1,2)*DS +BCINE2(1,3)*DC1
            DQS=BCINE2(2,1)*DR +BCINE2(2,2)*DS +BCINE2(2,3)*DC1
            DE =BCINE2(3,1)*DR +BCINE2(3,2)*DS +BCINE2(3,3)*DC1
C
            DQN=(TANDES-TANFLO)/(1.0+TANDES*TANFLO)*(QOLD+DQS)
C
C           STORE THE CHANGES IN THE CARTESIAN FRAME
C
            CHNGE2(1,INODE)=DR
            CHNGE2(2,INODE)=DQS*COSFLO-DQN*SINFLO
            CHNGE2(3,INODE)=DQN*COSFLO+DQS*SINFLO
            CHNGE2(4,INODE)=DE
C
         ELSE
C
C.........SUPERSONIC INFLOW...ALL CHARACTERISTIC WAVES FROM OUTSIDE THE
C           DOMAIN, SO SET CHANGES AT THE NODE TO ZERO
C
            CHNGE2(1,INODE)=0.00
            CHNGE2(2,INODE)=0.00
            CHNGE2(3,INODE)=0.00
            CHNGE2(4,INODE)=0.00
C
         ENDIF
C
         GOTO 90
C
C*****CORNER
C
50       COSVAL=VBCNE2(1,IBOUND)
         SINVAL=VBCNE2(2,IBOUND)
C
C     COMPUTE PROJECTION OF VELOCITY ON INWARD NORMAL VECTOR TO
C        DETERMINE IF FLOW IS ENTERING OR EXITING
C
         UINOUT=(DPENG2(2,INODE)*COSVAL+DPENG2(3,INODE)*SINVAL)
     &                    /SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)
         UMAG=SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)
     &                                              /DPENG2(1,INODE)
C
C     BRANCH OUT DPENDING ON WEETHER THE VELOCITY NORMAL TO THE
C        BOUNDARY IS ENTERING OR EXITING, AND WHETHER THE NORMAL
C        COMPONENT IS SUBSONIC OF SUPERSONIC
C
         IF(UINOUT.LT.-0.01 .AND. UMAG.GT.CBCE2) THEN
C
C.........SUPERSONIC OUTFLOW...ALL CHARACTERISTIC WAVES FROM INSIDE
C           THE DOMAIN, SO LEAVE CHANGES PREDICTED FROM DISTRIBUTION
```

```
C          UNALTERED
C
           CONTINUE
C
         ELSEIF(UINOUT.LT.-0.01) THEN
C
C.........SUBSONIC OUTFLOW...THREE CHARACTERISTICS WAVES FROM INSIDE
C          AND ONE FROM OUTSIDE THE DOMAIN.  PICK THE INCOMING
C          CHARACTERISTIC WAVE SUCH THAT THE STATIC PRESSURE IS HELD
C
           RINF=BONDG2(6,IBOUND)
           UINF=BONDG2(6,IBOUND)/RINF
           VINF=BONDG2(7,IBOUND)/RINF
           QINF=UINF*UINF+VINF*VINF
           PINF=GMO3E2*(BONDG2(8,IBOUND)-0.50*RINF*QINF)
C
           UFS=UINF
           VFS=VINF
           QFS=UFS*UFS+VFS*VFS
C
           PDES=(PINF**GAMM1+GAMM4*(RINF/PINF**GAMM3)*(QINF-QFS))**GAMM2
C
C          COMPUTE FLOW CONDITIONS AT THE OLD TIME
C
           ROLD=DPENG2(1,INODE)
           QOLD=SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)
           EOLD=DPENG2(4,INODE)
           POLD=GMO3E2*(EOLD-0.5*QOLD*QOLD/ROLD)
C
C          COMPUTE SINE AND COSINE OF FLOW DIRECTION SINCE BOUNDARY
C          CONDITIONS WILL BE APPLIED IN INTRINSIC COORDINATES
C
           COSFLO=DPENG2(2,INODE)/QOLD
           SINFLO=DPENG2(3,INODE)/QOLD
C
C          COPMUTE CHANGES PREDICTED BY DISTRIBUTION FORMULAE IN
C          INTRINSIC COORDINATES
C
           DR =CHNGE2(1,INODE)
           DQS=CHNGE2(2,INODE)*COSFLO+CHNGE2(3,INODE)*SINFLO
           DQN=CHNGE2(3,INODE)*COSFLO-CHNGE2(2,INODE)*SINFLO
           DE =CHNGE2(4,INODE)
C
C          CALCULATE THE FORCING FUNCTIONS AT THIS POINT
C
           DC4=BCEXE2(4,1)*DR +BCEXE2(4,2)*DQS+BCEXE2(4,3)*DE
           DC2=BCEXE2(5,1)*DR +BCEXE2(5,2)*DQS+BCEXE2(5,3)*DE
           DP =PDES-POLD
C
C          CALCULATE THE CHANGES IN THE DEPENDENT VARIABLES IN THE
C          INTRINSIC COORDINATE FRAME
C
           DR =BCEXE2(1,1)*DC4+BCEXE2(1,2)*DC2+BCEXE2(1,3)*DP
```

```
            DQS=BCEXE2(2,1)*DC4+BCEXE2(2,2)*DC2+BCEXE2(2,3)*DP
            DE =BCEXE2(3,1)*DC4+BCEXE2(3,2)*DC2+BCEXE2(3,3)*DP
C
C           STORE THE CHANGES IN THE CARTESIAN FRAME
C
            CHNGE2(1,INODE)=DR
            CHNGE2(2,INODE)=DQS*COSFLO-DQN*SINFLO
            CHNGE2(3,INODE)=DQN*COSFLO+DQS*SINFLO
            CHNGE2(4,INODE)=DE
C
         ELSEIF(UMAG.LT.CBCE2) THEN
C
C.........SUBSONIC INFLOW...ONE CHARACTERISTIC WAVE FROM INSIDE AND
C           THREE FROM OUTSIDE THE DOMAIN.  PICK THE INCOMING
C           CHARACTERISTIC WAVE SUCH THAT ANGLE, ENTROPY, AND STAGNATION
C           ENTHALPY ARE HELD
C
            UINF  =BONDG2(6,IBOUND)/BONDG2(5,IBOUND)
            VINF  =BONDG2(7,IBOUND)/BONDG2(5,IBOUND)
C
            TANDES=SINVAL/COSVAL
C
C           COMPUTE FLOW CONDITIONS AT THE OLD TIME
C
            ROLD=DPENG2(1,INODE)
            QOLD=SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)
            EOLD=DPENG2(4,INODE)
            POLD=GMO3E2*(EOLD-0.5*QOLD*QOLD/ROLD)
C
C           COMPUTE SINE, COSINE, AND TANGENT OF FLOW DIRECTION SINCE
C           BOUNDARY CONDITIONS WILL BE APPLIED IN INTRINSIC COORDINATES
C
            COSFLO=DPENG2(2,INODE)/QOLD
            SINFLO=DPENG2(3,INODE)/QOLD
            TANFLO=DPENG2(3,INODE)/DPENG2(2,INODE)
C
C           COPMUTE CHANGES PREDICTED BY DISTRIBUTION FORMULAE IN
C           INTRINSIC COORDINATES
C
            DR =CHNGE2(1,INODE)
            DQS=CHNGE2(2,INODE)*COSFLO+CHNGE2(3,INODE)*SINFLO
            DQN=CHNGE2(3,INODE)*COSFLO-CHNGE2(2,INODE)*SINFLO
            DE =CHNGE2(4,INODE)
C
C           CALCULATE THE FORCING FUNCTIONS AT THIS POINT
C
            DH =HDESE2-(0.5*(QOLD/ROLD)**2-GM11E2*POLD/ROLD)
            DS =(SDESE2-POLD/ROLD**GMOOE2)
            DC1=BCINE2(4,1)*DR +BCINE2(4,2)*DQS+BCINE2(4,3)*DE
C
C           CALCULATE THE CHANGES IN THE DEPENDENT VARIABLES IN THE
C           INTRINSIC COORDINATE FRAME
C
```

```
            DR =BCINE2(1,1)*DH +BCINE2(1,2)*DS +BCINE2(1,3)*DC1
            DQS=BCINE2(2,1)*DH +BCINE2(2,2)*DS +BCINE2(2,3)*DC1
            DE =BCINE2(3,1)*DH +BCINE2(3,2)*DS +BCINE2(3,3)*DC1
C
            DQN=(TANDES-TANFLO)/(1.0+TANDES*TANFLO)*(QOLD+DQS)
C
C           STORE THE CHANGES IN THE CARTESIAN FRAME
C
            CHNGE2(1,INODE)=DR
            CHNGE2(2,INODE)=DQS*COSFLO-DQN*SINFLO
            CHNGE2(3,INODE)=DQN*COSFLO+DQS*SINFLO
            CHNGE2(4,INODE)=DE
C
         ELSE
C
C.........SUPERSONIC INFLOW...ALL CHARACTERISTIC WAVES FROM OUTSIDE THE
C           DOMAIN, SO SET CHANGES AT THE NODE TO ZERO
C
            CHNGE2(1,INODE)=0.00
            CHNGE2(2,INODE)=0.00
            CHNGE2(3,INODE)=0.00
            CHNGE2(4,INODE)=0.00
C
         ENDIF
C
         GOTO 90
C
C*****GO BACK FOR NEXT NODE
C
90       CONTINUE
C
         RETURN
         END
```

E2BFIX

```
         SUBROUTINE E2BFIX
C
         INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE FIXES THE WALL VELOCITY SO THAT IT IS TANGENT
C        TO ALL WALL NODES
C
C***********************************************************************
C
         INCLUDE '[.GRID2D]G2COMN.INC'
C
         INCLUDE '[.EULR2D]E2COMN.INC'
C
         INCLUDE '[.UTILITY]HEXCOD.INC'
```

```
C
C*************************************************************************
C
C     APPLY BOUNDARY CONDITIONS AT EACH BOUNDARY NODE
C
        DO 10 IBOUND=1,NBNDG2
C
C     BRANCH OUT ACCORDING TO TYPE
C
        INODE =IBNDG2(1,IBOUND)
        ITYPE =IBNDG2(5,IBOUND)
C
        IF(ITYPE.NE.3) GOTO 10
C
C*****SOLID WALL, SIMPLE WAVE CONDITION
C
C     NOTE: THIS ROUTINE ROTATES THE VELOCITY VECTOR TO BE
C           ALIGNED WITH THE WALL
C
        COSWAL=VBCNE2(1,IBOUND)
        SINWAL=VBCNE2(2,IBOUND)
C
C     CALCULATE THE TANGENTIAL COMPONENT OF THE CHANGE IN MOMENTUM AND
C       BREAK IT INTO ITS COMPONENTS
C
        RQ=DPENG2(2,INODE)*COSWAL+DPENG2(3,INODE)*SINWAL
C
        DPENG2(2,INODE)=RQ*COSWAL
        DPENG2(3,INODE)=RQ*SINWAL
C
10      CONTINUE
C
        RETURN
        END
```

E2COMP

```
        SUBROUTINE E2COMP(IUNIT
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CREATES A COMPARISON FILE CONTAINING:
C        '   X    '
C        'MACH NO.'
C        'PT LOSS '
C        '  -CP   '
C
C*************************************************************************
```

```
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
C*************************************************************************
C
C     WRITE OUT HEADER INFORMATION
C
      WRITE(IUNIT,10)
10    FORMAT('COMPUTED SOLUTION'                          /
     &       '   X      MACH NO.    -CP     PT LOSS   ')
C
C     LOOP THROUGH ALL THE SOLID SURFACE NODES
C
      DO 30 IFOR=1,NFORE2
      I=IFORE2(2,IFOR)
C
      X     =GEOMG2(1,I)
C
      TEMP =SQRT(DPENG2(2,I)**2+DPENG2(3,I)**2)/DPENG2(1,I)
      XMACH=TEMP/SQRT(GMO1E2*(DPENG2(4,I)/DPENG2(1,I)-0.5*TEMP**2))
C
      QREF =0.50*(U2RFE2*U2RFE2+U3RFE2*U3RFE2)/U1RFE2
      TEMP =0.50*(DPENG2(2,I)**2+DPENG2(3,I)**2)/DPENG2(1,I)
      CP   =-GMO3E2*((DPENG2(4,I)-TEMP)-(U4RFE2-QREF))/QREF
C
      EM2  =U2RFE2*U2RFE2+U3RFE2*U3RFE2
      FACT =U4RFE2*U1RFE2-0.5*EM2
      PTREF=(GMO3E2*FACT/U1RFE2)/(1.0+EM2/(2.0*GMOOE2*FACT))**GM11E2
      TEMP =DPENG2(2,I)**2+DPENG2(3,I)**2
      TEMP2=DPENG2(4,I)*DPENG2(1,I)-0.5*TEMP
      TEMP=(GMO3E2*TEMP2/DPENG2(1,I))
     &                          /(1.0+TEMP/(2.0*GMOOE2*TEMP2))**GM11E2
      PTLOS=(PTREF-TEMP)/PTREF
C
      WRITE(IUNIT,20) X,XMACH,CP,PTLOS
20    FORMAT(4F10.5)
C
30    CONTINUE
C
      RETURN
      END
```

E2CONV

```
      SUBROUTINE E2CONV(IOPT
     &
     &                  )
C
```

```
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE COMPUTES CONVERGENCE STATISTICS
C       IF IOPT =0 INITIALIZE
C               =1 COMPUTE STATISTICS
C
C*****************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************************
C
      GOTO (100,200), (IOPT+1)
      RETURN
C
C*****INITIALIZE STATISTICS BY SAVING RESULTS BEFORE ITERATION
C
100   DO 110 INODE=1,NNODG2
      WORKG2(INODE)=DPENG2(2,INODE)
110   CONTINUE
C
      TITLG2='********'
C
      RETURN
C
C*****COMPUTE CONVERGENCE STATISTICS AND WRITE CONVERGENCE STATISTICS TO
C       IHSTE2
C
C     INCREMENT NUMBER OF ITERATIONS AND FIND CPU TIME
C
200   ITERE2=ITERE2+1
      CALL UTTIME(TIMEE2)
C
C     COMPUTE THE FORCE COEFFICIENTS
C
      CALL E2FORC
C
C     COMPUTE MAXIMUM, AVERAGE, AND RMS CHANGES OF DPENG2(2), AND
C       SAVE THE CHANGES IN WORKG2
C
      DU2ME2=0.0
      DU2AE2=0.0
      DU2RE2=0.0
C
      DO 220 INODE=1,NNODG2
      WORKG2(INODE)=ABS(DPENG2(2,INODE)-WORKG2(INODE))
      IF(WORKG2(INODE).GT.DU2ME2) THEN
         DU2ME2=WORKG2(INODE)
         IDU2E2=INODE
```

```
        ENDIF
        DU2AE2=DU2AE2+WORKG2(INODE)
        DU2RE2=DU2RE2+WORKG2(INODE)*WORKG2(INODE)
220     CONTINUE
C
        DU2AE2=     DU2AE2/NNODG2
        DU2RE2=SQRT(DU2RE2/NNODG2)
C
        TITLG2='  DU2    '
C
C     STORE THE MAXIMUM AND MINIMUM VALUES OF THE SMOOTHING WEIGHTING
C
        SMINE2=DPENG2(5,1)
        SMAXE2=DPENG2(5,1)
C
        DO 230 INODE=1,NNODG2
        SMINE2=MIN(SMINE2,DPENG2(5,INODE))
        SMAXE2=MAX(SMAXE2,DPENG2(5,INODE))
230     CONTINUE
C
C     COMPUTE THE LOG OF THE DU'S
C
        DU2MLL=LOG10(DU2ME2)
        DU2ALL=LOG10(DU2AE2)
        DU2RLL=LOG10(DU2RE2)
C
        WRITE(IHSTE2,240) ITERE2,TIMEE2,DU2MLL,IDU2E2,DU2ALL,DU2RLL,
     &                    CLFTE2,CDRGE2,SMAXE2,SMINE2
240     FORMAT(I5,F10.2,F10.6,I5,6F10.6)
C
        RETURN
C
        END
```

E2CTRL

```
        SUBROUTINE E2CTRL(DUTOL,CLTOL,CDTOL
     &
     &                                    )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE SETS UP THE CONTROL AN INDEX VECTORS NEEDED IN
C       THE REMAINDER OF THE PROGRAM
C
C*****************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
```

```
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*************************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS NOT IN MEMORY
C
        IF(INBRG2.NE.0) THEN
           CALL UTEROR(+1,REAL(INBRG2),0.0)
        ENDIF
C
C     STORE THE TOLERANCES
C
        TOL1E2=DUTOL
        TOL2E2=CLTOL
        TOL3E2=CDTOL
C
C*****E2BCON(IMGL)*****************************************************
C
C     CALCULATE INVARIENT QUANTITIES FOR EACH BOUNDARY NODE
C
        DO 1990 IBOUND=1,NBNDG2
C
C     BRANCH OUT ACCORDING TO TYPE
C
        INODE =IBNDG2(1,IBOUND)
        ICLEFT=IBNDG2(2,IBOUND)
        ICRITE=IBNDG2(3,IBOUND)
        IEDGE =IBNDG2(4,IBOUND)
        ITYPE =IBNDG2(5,IBOUND)
C
        GOTO (1910,1100,1200,1300,1400,1500,1600), (ITYPE+1)
        GOTO 1900
C
C.....RADIATION CONDITION
C
1100    GOTO 1910
C
C.....DIRICHLET CONDITION
C
1200    GOTO 1910
C
C.....SOLID WALL
C
C     NOTE: THIS ROUTINE ASSUMES THAT THE INITIAL VELOCITY VECTOR
C           IS ALIGNED WITH THE WALL
C
C     WALL SLOPE IS ASSUMED TO BE AVERAGE SPLINE SLOPES
C
1300    DX=BONDG2(1,IBOUND)+BONDG2(3,IBOUND)
        DY=BONDG2(2,IBOUND)+BONDG2(4,IBOUND)
C
        DS=SQRT(DX*DX+DY*DY)
```

```
C
        COSWAL=DX/DS
        SINWAL=DY/DS
C
        VBCNE2(1,IBOUND)=COSWAL
        VBCNE2(2,IBOUND)=SINWAL
C
        GOTO 1910
C
C.....FREE STREAM, CHARACTERISTIC CONDITION (USAB)
C
C     CALCULATE THE INFINITY CONDITIONS (DOESN'T INCLUDE VORTEX)
C
1400    RINF=BOUNDG2(5,IBOUND)
        UINF=BOUNDG2(6,IBOUND)/RINF
        VINF=BOUNDG2(7,IBOUND)/RINF
        QINF=SQRT(UINF*UINF+VINF*VINF)
        PINF=GMO3E2*(BOUNDG2(8,IBOUND)-0.5O*RINF*QINF*QINF)
C
C     CALCULATE AUXILIARY CONSTANTS USED IN VORTEX COMPUTATION FOR
C        VORTEX CENTERED AT QUARTER-CHORD OF AIRFOIL
C
        DX   =GEOMG2(1,INODE)-0.25*CHRDE2
        DY   =GEOMG2(2,INODE)
        THETA=ATAN2(DY,DX)
        RAD  =SQRT(DX*DX+DY*DY)
        ALFA =ATAN2(VINF,UINF)
C
        XMCHSQ=(RINF*QINF*QINF)/(GMOOE2*PINF)
        IF(XMCHSQ.LT.1.0) THEN
           BETA=SQRT(1.0-XMCHSQ)
        ELSE
           BETA=0.00
        ENDIF
C
        DANG  =THETA-ALFA
        VORFAC=(QINF*CHRDE2*BETA)
     &       /(12.5663703*RAD*((COS(DANG))**2+(BETA*SIN(DANG))**2))
C
C     COMPUTE PROJECTION OF VELOCITY ON INWARD NORMAL VECTOR TO
C        DETERMINE IF FLOW IS ENTERING OR EXITING
C
        DX=BOUNDG2(1,IBOUND)+BONDG2(3,IBOUND)
        DY=BOUNDG2(2,IBOUND)+BONDG2(4,IBOUND)
C
        DS=SQRT(DX*DX+DY*DY)
C
        COSBND=DX/DS
        SINBND=DY/DS
C
        VBCNE2( 1,IBOUND)=COSBND
        VBCNE2( 2,IBOUND)=SINBND
        VBCNE2( 3,IBOUND)=VORFAC*COS(THETA)
```

```
        VBCNE2( 4,IBOUND)=VORFAC*SIN(THETA)
C
        GOTO 1910
C
C.....CORNER - FLOW IN EAST/WEST DIRECTION
C
1500    GOTO (1900,1525,1530,1900,1535,1900,1900,1510,
     &        1540,1900,1900,1505,1900,1520,1515,1900), (IEDGE+1)
        GOTO 1900
C
C     SOUTHWEST CORNER (OUTSIDE)
C
1505    JNODE=ICELG2(4,ICLEFT)
        GOTO 1650
C
C     SOUTHEAST CORNER (OUTSIDE)
C
1510    JNODE=ICELG2(2,ICLEFT)
        GOTO 1650
C
C     NORTHEAST CORNER (OUTSIDE)
C
1515    JNODE=ICELG2(8,ICLEFT)
        GOTO 1650
C
C     NORTHWEST CORNER (OUTSIDE)
C
1520    JNODE=ICELG2(6,ICLEFT)
        GOTO 1650
C
C     SOUTHWEST CORNER (INSIDE)
C
1525    JNODE=ICELG2(6,ICRITE)
        GOTO 1650
C
C     SOUTHEAST CORNER (INSIDE)
C
1530    JNODE=ICELG2(8,ICLEFT)
        GOTO 1650
C
C     NORTHEAST CORNER (INSIDE)
C
1535    JNODE=ICELG2(2,ICRITE)
        GOTO 1650
C
C     NORTHWEST CORNER (INSIDE)
C
1540    JNODE=ICELG2(4,ICLEFT)
        GOTO 1650
C
C.....CORNER - FLOW IN NORTH/SOUTH DIRECTION
C
1600    GOTO (1900,1525,1630,1900,1635,1900,1900,1610,
```

```
     &        1640,1900,1900,1605,1900,1620,1615,1900), (IEDGE+1)
         GOTO 1900
C
C     SOUTHWEST CORNER (OUTSIDE)
C
1605     JNODE=ICELG2(8,ICLEFT)
         GOTO 1650
C
C     SOUTHEAST CORNER (OUTSIDE)
C
1610     JNODE=ICELG2(6,ICLEFT)
         GOTO 1650
C
C     NORTHEAST CORNER (OUTSIDE)
C
1615     JNODE=ICELG2(4,ICLEFT)
         GOTO 1650
C
C     NORTHWEST CORNER (OUTSIDE)
C
1620     JNODE=ICELG2(2,ICLEFT)
         GOTO 1650
C
C     SOUTHWEST CORNER (INSIDE)
C
1625     JNODE=ICELG2(6,ICLEFT)
         GOTO 1650
C
C     SOUTHEAST CORNER (INSIDE)
C
1630     JNODE=ICELG2(8,ICRITE)
         GOTO 1650
C
C     NORTHEAST CORNER (INSIDE)
C
1635     JNODE=ICELG2(2,ICLEFT)
         GOTO 1650
C
C     NORTHWEST CORNER (INSIDE)
C
1640     JNODE=ICELG2(4,ICRITE)
         GOTO 1650
C
C     COMPUTE THE SINE AND COSINE OF THE WALL SLOPE (INWARD)
C
1650     DX=GEOMG2(1,JNODE)-GEOMG2(1,INODE)
         DY=GEOMG2(2,JNODE)-GEOMG2(2,INODE)
C
         DS=SQRT(DX*DX+DY*DY)
C
         COSWAL=DX/DS
         SINWAL=DY/DS
C
```

```
        VBCNE2(1,IBOUND)=COSWAL
        VBCNE2(2,IBOUND)=SINWAL
C
        GOTO 1910
C
C.....INVALID BOUNDARY CONDITION TYPE FOR THIS CELL TYPE
C
1900    CALL UTEROR(+2,REAL(IBOUND),REAL(IEDGE))
C
C.....GO BACK FOR NEXT NODE
C
1910    CONTINUE
C
1990    CONTINUE
C
C     SET UP THE LINIEARIZATION MATRICIES FOR FAR-FIELD BOUNDARY
C        CONDITIONS
C
C     ...USE INFINITY CONDITIONS
C
        R=U1RFE2
        Q=SQRT(U2RFE2+U2RFE2+U3RFE2+U3RFE2)
        E=U4RFE2
        P=GM03E2*(E-0.5*Q*Q/R)
        C=SQRT(GM00E2*P/R)
C
C     ...COMPUTE THE CONDITIONS TO BE HELD UPSTREAM
C
        HDE3E2=0.5*Q*Q/R/R-GM11E2*P/R
        SDE3E2 =P/R**GM00E2
        CBCE2  =C
C
C     ...SUBSONIC INLET INFLUENCE MATRIX
C
        BCINE2(1,1)=R/C/C
        BCINE2(1,2)=-GM00E2/C/C
        BCINE2(1,3)=0.
        BCINE2(2,1)=R/C+Q/C/C
        BCINE2(2,2)=-1./C-GM00E2*Q/R/C/C
        BCINE2(2,3)=-1./C
        BCINE2(3,1)=Q/C-GM04E2*Q*Q/R/C/C+GM00E2*E/C/C
        BCINE2(3,2)=-Q/R/C-GM00E2*E/R/C/C
        BCINE2(3,3)=-Q/R/C
C
        BCINE2(4,1)=GM04E2*Q*Q/R/R+C*Q/R
        BCINE2(4,2)=-GM03E2*Q/R-C
        BCINE2(4,3)=GM03E2
C
C     ...SUBSONIC EXIT INFLUENCE MATRIX
C
        BCEXE2(1,1)=0.
        BCEXE2(1,2)=-GM03E2/C/C
        BCEXE2(1,3)=1./C/C
```

```
        BCEXE2(2,1)=1./C
        BCEXE2(2,2)=-GM03E2*Q/R/C/C
        BCEXE2(2,3)=-1./C+Q/R/C/C
        BCEXE2(3,1)=Q/R/C
        BCEXE2(3,2)=-GM04E2*Q*Q/R/R/C/C
        BCEXE2(3,3)=-Q/R/C+GM00E2*E/R/C/C-GM04E2*Q*Q/R/R/C/C
C
        BCEXE2(4,1)=GM04E2*Q*Q/R/R-C*Q/R
        BCEXE2(4,2)=-GM03E2*Q/R+C
        BCEXE2(4,3)=GM03E2
        BCEXE2(5,1)=GM07E2*Q*Q/R/R-GM00E2*E/R
        BCEXE2(5,2)=-Q/R
        BCEXE2(5,3)=1.0
C
C*****E2DIST(IMGL)*************************************************************
C
C     LOOP THROUGH EACH MULTIPLE-GRID LEVEL
C
2000    DO 2160 IMGL=-MLVLG2,MLVLG2
C
C     LOOP THROUGH ALL FINE CELLS WHICH ARE NOT
C        JUST OUTSIDE AN EMBEDDED REGION AND NOT NEAR A BOUNDARY
C
        DO 2050 ICELL=ILVLG2(1,IMGL),ILVLG2(2,IMGL)
        VDISE2( 1,ICELL)=1.0
        VDISE2( 2,ICELL)=1.0
        VDISE2( 3,ICELL)=1.0
        VDISE2( 4,ICELL)=1.0
2050    CONTINUE
C
C     LOOP THROUGH ALL CELLS WHICH ARE JUST OUTSIDE AN EMBEDDED REGION
C        OR NEAR A BOUNDARY
C
        DO 2100 ICELL=ILVLG2(3,IMGL),ILVLG2(4,IMGL)
C
        ISW =ICELG2( 2,ICELL)
        ISE =ICELG2( 4,ICELL)
        INE =ICELG2( 6,ICELL)
        INW =ICELG2( 8,ICELL)
        IAUX=ICELG2(10,ICELL)
C
C     DETERMINE IF THIS CELL BOUNDS A SOLID BOUNDARY
C
        VDISE2(1,ICELL)=1.0
        IF(IAND(IAUX,HL0003).EQ.HL0003) THEN
           IHIT=0
           DO 2060 IBOUND=1,NBNDG2
           IF(ISW.EQ.IBNDG2(1,IBOUND) .AND.
     &             (IBNDG2(5,IBOUND).EQ.1 .OR.
     &              IBNDG2(5,IBOUND).EQ.3 .OR.
     &              IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
           IF(ISE.EQ.IBNDG2(1,IBOUND) .AND.
     &             (IBNDG2(5,IBOUND).EQ.1 .OR.
```

```
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
2060      CONTINUE
          IF(IHIT.EQ.2) VDISE2(1,ICELL)=2.0
        ENDIF
C
        VDISE2(2,ICELL)=1.0
        IF(IAND(IAUX,ELOOO6).EQ.ELOOO6) THEN
          IHIT=0
          DO 2070 IBOUND=1,NBNDG2
          IF(ISE.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
          IF(INE.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
2070      CONTINUE
          IF(IHIT.EQ.2) VDISE2(2,ICELL)=2.0
        ENDIF
C
        VDISE2(3,ICELL)=1.0
        IF(IAND(IAUX,ELOOOC).EQ.ELOOOC) THEN
          IHIT=0
          DO 2080 IBOUND=1,NBNDG2
          IF(INE.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
          IF(INW.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
2080      CONTINUE
          IF(IHIT.EQ.2) VDISE2(3,ICELL)=2.0
        ENDIF
C
        VDISE2(4,ICELL)=1.0
        IF(IAND(IAUX,ELOOO9).EQ.ELOOO9) THEN
          IHIT=0
          DO 2090 IBOUND=1,NBNDG2
          IF(INW.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
          IF(ISW.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
2090      CONTINUE
          IF(IHIT.EQ.2) VDISE2(4,ICELL)=2.0
        ENDIF
```

```
C
2100    CONTINUE
C
C     LOOP THROUGH ALL COARSE CELLS
C
        DO 2150 ICELL=ILVLG2(5,IMGL),ILVLG2(6,IMGL)
C
        ISW =ICELG2( 2,ICELL)
        ISE =ICELG2( 4,ICELL)
        INE =ICELG2( 6,ICELL)
        INW =ICELG2( 8,ICELL)
        IAUX=ICELG2(10,ICELL)
C
C     DETERMINE IF THIS CELL BOUNDS A SOLID BOUNDARY
C
        VDISE2(1,ICELL)=1.0
        IF(IAND(IAUX,HL0003).EQ.HL0003) THEN
           IHIT=0
           DO 2110 IBOUND=1,NBNDG2
           IF(ISW.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
           IF(ISE.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
2110       CONTINUE
           IF(IHIT.EQ.2) VDISE2(1,ICELL)=2.0
        ENDIF
C
        VDISE2(2,ICELL)=1.0
        IF(IAND(IAUX,HL0006).EQ.HL0006) THEN
           IHIT=0
           DO 2120 IBOUND=1,NBNDG2
           IF(ISE.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
           IF(INE.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
2120       CONTINUE
           IF(IHIT.EQ.2) VDISE2(2,ICELL)=2.0
        ENDIF
C
        VDISE2(3,ICELL)=1.0
        IF(IAND(IAUX,HL000C).EQ.HL000C) THEN
           IHIT=0
           DO 2130 IBOUND=1,NBNDG2
           IF(INE.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
```

```
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
           IF(INW.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.5     )) IHIT=IHIT+1
2130       CONTINUE
           IF(IHIT.EQ.2) VDISE2(3,ICELL)=2.0
        ENDIF
C
        VDISE2(4,ICELL)=1.0
        IF(IAND(IAUX,HLO009).EQ.HLO009) THEN
           IHIT=0
           DO 2140 IBOUND=1,NBNDG2
           IF(INW.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
           IF(ISW.EQ.IBNDG2(1,IBOUND) .AND.
     &              (IBNDG2(5,IBOUND).EQ.1 .OR.
     &               IBNDG2(5,IBOUND).EQ.3 .OR.
     &               IBNDG2(5,IBOUND).EQ.6     )) IHIT=IHIT+1
2140       CONTINUE
           IF(IHIT.EQ.2) VDISE2(4,ICELL)=2.0
        ENDIF
C
2150    CONTINUE
C
C     NEXT MULTIPLE-GRID LEVEL
C
2160    CONTINUE
C
C*****E2FORC(INODE)*****************************************************
C
3000    NFORE2=0
C
C     LOOP THROUGH ALL BOUNDARY POINTERS, PICKING UP PORTIONS OF THE
C       SOLID SURFACE
C
        DO 3080 IBOUND=1,NBNDG2
C
        INODE=IBNDG2(1,IBOUND)
        ILEFT=IBNDG2(2,IBOUND)
        IEDGE=IBNDG2(4,IBOUND)
        ITYPE=IBNDG2(5,IBOUND)
C
C     JUMP TO NEXT BOUNDARY NODE IF THIS COULD NOT BE PART OF A
C       SOLID SURFACE
C
        IF(ITYPE.EQ.0 .OR. ITYPE.EQ.2 .OR. ITYPE.EQ.4) GOTO 3080
C
C     BRANCH DEPENDING ON WHICH EDGE THE BOUNDARY CONDITION REFERS TO
C
```

```
      GOTO (3080,3010,3020,3010,3030,3030,3020,3010,
     &       3040,3040,3010,3040,3030,3030,3020,3010), (IEDGE+1)
C
C     ...SOUTH SIDE, SOUTHEAST CORNER (OUTSIDE), SOUTHWEST
C        CORNER (INSIDE), OR TRAILING EDGE SLIT
C
3010    JNODE=ICELG2(2,ILEFT)
        GOTO 3050
C
C     ...EAST SIDE, NORTHEAST CORNER (OUTSIDE), OR SOUTHEAST
C        CORNER (INSIDE)
C
3020    JNODE=ICELG2(4,ILEFT)
        GOTO 3050
C
C     ...NORTH SIDE, NORTHWEST CORNER (OUTSIDE), NORTHEAST
C        CORNER (INSIDE), OR LEADING EDGE SLIT
C
3030    JNODE=ICELG2(6,ILEFT)
        GOTO 3050
C
C     ...WEST SIDE, SOUTHWEST CORNER (OUTSIDE), OR NORTHWEST
C        CORNER (INSIDE)
C
3040    JNODE=ICELG2(8,ILEFT)
C
C     FIND THE BOUNDARY POINTER FOR JNODE
C
3050    DO 3060 JBOUND=1,NBNDG2
        IF(IBNDG2(1,JBOUND).EQ.JNODE) THEN
           JTYPE=IBNDG2(5,JBOUND)
           IF(JTYPE.EQ.0 .OR. JTYPE.EQ.2 .OR. JTYPE.EQ.4) GOTO 3080
           GOTO 3070
        ENDIF
3060    CONTINUE
C
C     ERROR, COULD NOT FIND OTHER NODE IN THIS PAIR IN BOUNDARY POINTERS
C
        CALL UTEROR(+3,REAL(JNODE),REAL(INODE))
C
C     ADD THIS PAIR OF NODES TO THE LIST
C
3070    NFORE2=NFORE2+1
        IFORE2(1,NFORE2)=JNODE
        IFORE2(2,NFORE2)=INODE
C
3080    CONTINUE
C
C*****E2INTP(IMGL)*************************************************
C
C     INITIALIZE NUMBER OF INTERPOLATION TRIPLETS
C
4000    IBEG=1
```

```
        IEND=0
C
C     LOOP THROUGH EVERY MULTIGRID LEVEL BETWEEN COARSEST AND FINEST
C
        DO 4180 IMGL=-MLVLG2,MLVLG2-1
C
C     LOOP THROUGH ALL COARSE CELLS ON THIS LEVEL, ADDING ANY INTERIOR
C        SIDE NODES TO STACK IF NOT ON IT ALREADY
C
        DO 4080 ICELL=ILVLG2(5,IMGL),ILVLG2(6,IMGL)
        IAUX=ICELG2(10,ICELL)
C
        IF(IAND(IAUX,HL0300).EQ.HL0300) GOTO 4020
C
        DO 4010 ISTACK=IBEG,IEND
        IF(ICELG2(3,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4020
4010    CONTINUE
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(2,ICELL)
        IITPE2(2,IEND)=ICELG2(3,ICELL)
        IITPE2(3,IEND)=ICELG2(4,ICELL)
C
4020    IF(IAND(IAUX,HL0600).EQ.HL0600) GOTO 4040
C
        DO 4030 ISTACK=IBEG,IEND
        IF(ICELG2(5,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4040
4030    CONTINUE
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(4,ICELL)
        IITPE2(2,IEND)=ICELG2(5,ICELL)
        IITPE2(3,IEND)=ICELG2(6,ICELL)
C
4040    IF(IAND(IAUX,HL0C00).EQ.HL0C00) GOTO 4060
C
        DO 4050 ISTACK=IBEG,IEND
        IF(ICELG2(7,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4060
4050    CONTINUE
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(6,ICELL)
        IITPE2(2,IEND)=ICELG2(7,ICELL)
        IITPE2(3,IEND)=ICELG2(8,ICELL)
C
4060    IF(IAND(IAUX,HL0900).EQ.HL0900) GOTO 4080
C
        DO 4070 ISTACK=IBEG,IEND
        IF(ICELG2(9,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4080
4070    CONTINUE
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(8,ICELL)
```

```
        IITPE2(2,IEND)=ICELG2(9,ICELL)
        IITPE2(3,IEND)=ICELG2(2,ICELL)
C
4080    CONTINUE
C
C     STORE FIRST AND LAST SIDE FOR THIS LEVEL
C
        LITPE2(1,IMGL)=IBEG
        LITPE2(2,IMGL)=IEND
C
        IBEG=IEND+1
C
C     LOOP THROUGH ALL COARSE CELLS ON THIS LEVEL, ADDING ALL CENTER
C       NODES TO LIST
C
        DO 4090 ICELL=ILVLG2(5,IMGL),ILVLG2(6,IMGL)
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(3,ICELL)
        IITPE2(2,IEND)=ICELG2(1,ICELL)
        IITPE2(3,IEND)=ICELG2(7,ICELL)
C
4090    CONTINUE
C
C     STORE FIRST AND LAST SIDE FOR THIS LEVEL
C
        LITPE2(3,IMGL)=IBEG
        LITPE2(4,IMGL)=IEND
C
        IBEG=IEND+1
C
C     LOOP THROUGH ALL COARSE CELLS ON THIS LEVEL, ADDING ANY EXTERIOR
C       SIDE NODES TO STACK IF NOT ON IT ALREADY
C
        DO 4170 ICELL=ILVLG2(5,IMGL+1),ILVLG2(6,IMGL+1)
        IAUX=ICELG2(10,ICELL)
C
        IF(IAND(IAUX,HL0300).NE.HL0300) GOTO 4110
C
        DO 4100 ISTACK=IBEG,IEND
        IF(ICELG2(3,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4110
4100    CONTINUE
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(2,ICELL)
        IITPE2(2,IEND)=ICELG2(3,ICELL)
        IITPE2(3,IEND)=ICELG2(4,ICELL)
C
4110    IF(IAND(IAUX,HL0600).NE.HL0600) GOTO 4130
C
        DO 4120 ISTACK=IBEG,IEND
        IF(ICELG2(5,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4130
4120    CONTINUE
```

```
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(4,ICELL)
        IITPE2(2,IEND)=ICELG2(5,ICELL)
        IITPE2(3,IEND)=ICELG2(6,ICELL)
C
4130    IF(IAND(IAUX,HLOC00).NE.HLOC00) GOTO 4150
C
        DO 4140 ISTACK=IBEG,IEND
        IF(ICELG2(7,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4150
4140    CONTINUE
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(6,ICELL)
        IITPE2(2,IEND)=ICELG2(7,ICELL)
        IITPE2(3,IEND)=ICELG2(8,ICELL)
C
4150    IF(IAND(IAUX,HL0900).NE.HL0900) GOTO 4170
C
        DO 4160 ISTACK=IBEG,IEND
        IF(ICELG2(9,ICELL).EQ.IITPE2(2,ISTACK)) GOTO 4170
4160    CONTINUE
C
        IEND=IEND+1
        IITPE2(1,IEND)=ICELG2(8,ICELL)
        IITPE2(2,IEND)=ICELG2(9,ICELL)
        IITPE2(3,IEND)=ICELG2(2,ICELL)
C
4170    CONTINUE
C
C     STORE FIRST AND LAST SIDE FOR THIS LEVEL
C
        LITPE2(5,IMGL)=IBEG
        LITPE2(6,IMGL)=IEND
C
        IBEG=IEND+1
C
C     NEXT MULTIPLE-GRID LEVEL
C
4180    CONTINUE
C
C*****E2UPDT(IMGL)*************************************************
C
C     INITIALLY SET ALL CONTROL BITS OFF
C
5000    CONTINUE
        DO 5010 INODE=1,NNODG2
        CUPDE2(INODE)=0
5010    CONTINUE
C
C     LOOP THROUGH EACH MULTIPLE-GRID LEVEL
C
        DO 5030 IMGL=-MLVLG2,MLVLG2
```

```
        IMASK=2**(IMGL+MLVLG2)
C
C     STEP THROUGH EACH CELL AT THIS AND ALL FINER LEVELS
C
        DO 5020 ICELL=ILVLG2(1,IMGL),NCELG2
        IAUX=ICELG2(10,ICELL)
C
C     SET CONTROL BIT ON FOR EACH NODE IF IT IS AN INTERIOR NODE
C
        IF(IAND(IAUX,HL1000).EQ.0)
     &       CUPDE2(ICELG2(2,ICELL))=IOR(CUPDE2(ICELG2(2,ICELL)),IMASK)
        IF(IAND(IAUX,HL2000).EQ.0)
     &       CUPDE2(ICELG2(4,ICELL))=IOR(CUPDE2(ICELG2(4,ICELL)),IMASK)
        IF(IAND(IAUX,HL4000).EQ.0)
     &       CUPDE2(ICELG2(6,ICELL))=IOR(CUPDE2(ICELG2(6,ICELL)),IMASK)
        IF(IAND(IAUX,HL8000).EQ.0)
     &       CUPDE2(ICELG2(8,ICELL))=IOR(CUPDE2(ICELG2(8,ICELL)),IMASK)
C
C     UPDATE SIDE NODES FROM JUST INSIDE (COARSE)
C
        IF(IAND(IAUX,HL0300).EQ.HL0300)
     &       CUPDE2(ICELG2(3,ICELL))=IOR(CUPDE2(ICELG2(3,ICELL)),IMASK)
        IF(IAND(IAUX,HL0600).EQ.HL0600)
     &       CUPDE2(ICELG2(5,ICELL))=IOR(CUPDE2(ICELG2(5,ICELL)),IMASK)
        IF(IAND(IAUX,HL0C00).EQ.HL0C00)
     &       CUPDE2(ICELG2(7,ICELL))=IOR(CUPDE2(ICELG2(7,ICELL)),IMASK)
        IF(IAND(IAUX,HL0900).EQ.HL0900)
     &       CUPDE2(ICELG2(9,ICELL))=IOR(CUPDE2(ICELG2(9,ICELL)),IMASK)
C
5020    CONTINUE
C
C     NEXT MULTIPLE-GRID LEVEL
C
5030    CONTINUE
C
C*****E2ZERO(IMGL)*************************************************
C
C     INITIALLY SET ALL CONTROL BITS OFF
C
6000    CONTINUE
        DO 6010 INODE=1,NNODG2
        CZROE2(INODE)=0
6010    CONTINUE
C
C     LOOP THROUGH EACH MULTIPLE-GRID LEVEL
C
        DO 6030 IMGL=-MLVLG2,MLVLG2
        IMASK=2**(IMGL+MLVLG2)
C
C     STEP THROUGH EACH COARSE CELL AT THIS LEVEL
C
        DO 6020 ICELL=ILVLG2(5,IMGL),ILVLG2(6,IMGL)
        IAUX=ICELG2(10,ICELL)
```

```
C
C     SET CONTROL BIT ON FOR EACH NODE IF NOT AN EDGE OF MESH
C        ON THE NEXT FINER LEVEL
C
        IF(IAND(IAUX,HLO100).EQ.0)
     &       CZROE2(ICELG2(2,ICELL))=IOR(CZROE2(ICELG2(2,ICELL)),IMASK)
        IF(IAND(IAUX,HLO200).EQ.0)
     &       CZROE2(ICELG2(4,ICELL))=IOR(CZROE2(ICELG2(4,ICELL)),IMASK)
        IF(IAND(IAUX,HLO400).EQ.0)
     &       CZROE2(ICELG2(6,ICELL))=IOR(CZROE2(ICELG2(6,ICELL)),IMASK)
        IF(IAND(IAUX,HLO800).EQ.0)
     &       CZROE2(ICELG2(8,ICELL))=IOR(CZROE2(ICELG2(8,ICELL)),IMASK)
C
6020    CONTINUE
C
C     NEXT MULTIPLE-GRID LEVEL
C
6030    CONTINUE
C
C*****END PROCESSING*****************************************************
C
9999    RETURN
        END
```

E2DATA

```
        SUBROUTINE E2DATA(IVAR
     &
     &                    )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CREATES OUTPUT ARRAYS
C        IF IVAR =  1      DATA$ = X-LOCATION
C                =  2              Y-LOCATION
C                = 11              DEPENDENT VARIABLE, U1
C                = 12              DEPENDENT VARIABLE, U2
C                = 13              DEPENDENT VARIABLE, U3
C                = 14              DEPENDENT VARIABLE, U4
C                = 15              DEPENDENT VARIABLE, U5
C                = 21              CHANGE IN U1
C                = 22              CHANGE IN U2
C                = 23              CHANGE IN U3
C                = 24              CHANGE IN U4
C                = 31              VELOCITY
C                = 32              FLOW ANGLE
C                = 33              MACH NUMBER
C                = 34              P/PT
C                = 35              P/PREF
C                = 36              ABS(PT-PTREF)/PTREF
```

```
C                 = 37              (S-SREF)/CV
C                 = 38              -CP = -(P-PREF)/QREF
C                 = 39              (PTREF-PT)/PTREF
C
C*************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
C*************************************************************************
C
C     BRANCH OUT BASED ON INPUT PARAMETER
C
      IF(IVAR.EQ. 1) GOTO  100
      IF(IVAR.EQ. 2) GOTO  200
      IF(IVAR.EQ.11) GOTO 1100
      IF(IVAR.EQ.12) GOTO 1200
      IF(IVAR.EQ.13) GOTO 1300
      IF(IVAR.EQ.14) GOTO 1400
      IF(IVAR.EQ.15) GOTO 1500
      IF(IVAR.EQ.21) GOTO 2100
      IF(IVAR.EQ.22) GOTO 2200
      IF(IVAR.EQ.23) GOTO 2300
      IF(IVAR.EQ.24) GOTO 2400
      IF(IVAR.EQ.31) GOTO 3100
      IF(IVAR.EQ.32) GOTO 3200
      IF(IVAR.EQ.33) GOTO 3300
      IF(IVAR.EQ.34) GOTO 3400
      IF(IVAR.EQ.35) GOTO 3500
      IF(IVAR.EQ.36) GOTO 3600
      IF(IVAR.EQ.37) GOTO 3700
      IF(IVAR.EQ.38) GOTO 3800
      IF(IVAR.EQ.39) GOTO 3900
                     RETURN
C
C*****X-POSITION
C
100   DO 110 I=1,NNODG2
      WORKG2(I)=GEOMG2(1,I)
110   CONTINUE
C
      TITLG2='   X    '
C
      RETURN
C
C*****Y-POSITION
C
200   DO 210 I=1,NNODG2
      WORKG2(I)=GEOMG2(2,I)
210   CONTINUE
C
      TITLG2='   Y    '
```

```
C
        RETURN
C
C*****DEPENDENT VARIABLE, U1
C
1100    DO 1110 I=1,NNODG2
        WORKG2(I)=DPENG2(1,I)
1110    CONTINUE
C
        TITLG2=' RHO   '
C
        RETURN
C
C*****DEPENDENT VARIABLE, U2
C
1200    DO 1210 I=1,NNODG2
        WORKG2(I)=DPENG2(2,I)
1210    CONTINUE
C
        TITLG2=' RHO*U  '
C
        RETURN
C
C*****DEPENDENT VARIABLE, U3
C
1300    DO 1310 I=1,NNODG2
        WORKG2(I)=DPENG2(3,I)
1310    CONTINUE
C
        TITLG2=' RHO*V  '
C
        RETURN
C
C*****DEPENDENT VARIABLE, U4
C
1400    DO 1410 I=1,NNODG2
        WORKG2(I)=DPENG2(4,I)
1410    CONTINUE
C
        TITLG2=' RHO*E  '
C
        RETURN
C
C*****DEPENDENT VARIABLE, U5
C
1500    DO 1510 I=1,NNODG2
        WORKG2(I)=DPENG2(5,I)
1510    CONTINUE
C
        TITLG2='SMTH WT.'
C
        RETURN
C
```

```
C*****CHANGE IN DEPENDENT VARIABLE, DU1
C
2100    DO 2110 I=1,NNODG2
        WORKG2(I)=CHNGE2(1,I)
2110    CONTINUE
C
        TITLG2=' D(RHO) '
C
        RETURN
C
C*****CHANGE IN DEPENDENT VARIABLE, DU2
C
2200    DO 2210 I=1,NNODG2
        WORKG2(I)=CHNGE2(2,I)
2210    CONTINUE
C
        TITLG2='D(RHO*U)'
C
        RETURN
C
C*****CHANGE IN DEPENDENT VARIABLE, DU3
C
2300    DO 2310 I=1,NNODG2
        WORKG2(I)=CHNGE2(3,I)
2310    CONTINUE
C
        TITLG2='D(RHO*V)'
C
        RETURN
C
C*****CHANGE IN DEPENDENT VARIABLE, DU4
C
2400    DO 2410 I=1,NNODG2
        WORKG2(I)=CHNGE2(4,I)
2410    CONTINUE
C
        TITLG2='D(RHO*E)'
C
        RETURN
C
C*****VELOCITY
C
3100    DO 3110 I=1,NNODG2
        WORKG2(I)=SQRT(DPENG2(2,I)**2+DPENG2(3,I)**2)/DPENG2(1,I)
3110    CONTINUE
C
        TITLG2='VELOCITY'
C
        RETURN
C
C*****FLOW ANGLE
C
3200    DO 3210 I=1,NNODG2
```

```
      TEMP=DPENG2(2,I)**2+DPENG2(3,I)**2
      IF(TEMP.LE.0.000001) THEN
         WORKG2(I)=0.00
      ELSE
         WORKG2(I)=ATAN2(DPENG2(3,I),DPENG2(2,I))
      ENDIF
3210  CONTINUE
C
      TITLG2=' ANGLE  '
C
      RETURN
C
C*****MACH NUMBER
C
3300  DO 3310 I=1,NNODG2
      TEMP=SQRT(DPENG2(2,I)**2+DPENG2(3,I)**2)/DPENG2(1,I)
      WORKG2(I)=TEMP/SQRT(GM01E2*(DPENG2(4,I)/DPENG2(1,I)
     &                                          -0.5*TEMP**2))
3310  CONTINUE
C
      TITLG2='MACH NO.'
C
      RETURN
C
C*****LOCAL PRESSURE RATIO
C
3400  DO 3410 I=1,NNODG2
      TEMP=DPENG2(2,I)**2+DPENG2(3,I)**2
      WORKG2(I)=(1.0+TEMP
     &        /(2.0*GM00E2*(DPENG2(4,I)*DPENG2(1,I)-0.5*TEMP)))**GM11E2
3410  CONTINUE
C
      TITLG2='  P/PT  '
C
      RETURN
C
C*****REFERENCE PRESSURE RATIO
C
3500  RHOUU=(U2RFE2*U2RFE2+U3RFE2*U3RFE2)/U1RFE2
      PREF=GM03E2*(U4RFE2-0.5*RHOUU)
C
      DO 3510 I=1,NNODG2
      TEMP=(DPENG2(2,I)**2+DPENG2(3,I)**2)/DPENG2(1,I)
      WORKG2(I)=GM03E2*(DPENG2(4,I)-0.5*TEMP)/PREF
3510  CONTINUE
C
      TITLG2=' P/PREF '
C
      RETURN
C
C*****ABSOLUTE VALUE OF TOTAL PRESSURE LOSS
C
3600  EM2=U2RFE2*U2RFE2+U3RFE2*U3RFE2
```

```
        FACT=U4RFE2*U1RFE2-0.5*EM2
        PTREF=(GM03E2*FACT/U1RFE2)/(1.0+EM2/(2.0*GM00E2*FACT))**GM11E2
C
        DO 3610 I=1,NNODG2
        TEMP=DPENG2(2,I)**2+DPENG2(3,I)**2
        TEMP2=DPENG2(4,I)*DPENG2(1,I)-0.5*TEMP
        TEMP=(GM03E2*TEMP2/DPENG2(1,I))
     &                          /(1.0+TEMP/(2.0*GM00E2*TEMP2))**GM11E2
        WORKG2(I)=ABS(TEMP-PTREF)/PTREF
3610    CONTINUE
C
        TITLG2='ABS.LOSS'
C
        RETURN
C
C*****ENTROPY RISE
C
3700    F1REF=2.0*U4RFE2-(U2RFE2*U2RFE2+U3RFE2*U3RFE2)/U1RFE2
        F2REF=U1RFE2
C
        DO 3710 I=1,NNODG2
        TEMP=2.0*DPENG2(4,I)-(DPENG2(2,I)**2+DPENG2(3,I)**2)/DPENG2(1,I)
        WORKG2(I)=LOG((TEMP/F1REF)/((DPENG2(1,I)/F2REF)**GM00E2))
3710    CONTINUE
C
        TITLG2=' S-SREF '
C
        RETURN
C
C*****PRESSURE COEFFICIENT (NEGATIVE)
C
3800    QREF=0.50*(U2RFE2*U2RFE2+U3RFE2*U3RFE2)/U1RFE2
C
        DO 3810 I=1,NNODG2
        TEMP=0.50*(DPENG2(2,I)**2+DPENG2(3,I)**2)/DPENG2(1,I)
        WORKG2(I)=-GM03E2*((DPENG2(4,I)-TEMP)-(U4RFE2-QREF))/QREF
3810    CONTINUE
C
        TITLG2='  -CP    '
C
        RETURN
C
C*****TOTAL PRESSURE LOSS
C
3900    EM2=U2RFE2*U2RFE2+U3RFE2*U3RFE2
        FACT=U4RFE2*U1RFE2-0.5*EM2
        PTREF=(GM03E2*FACT/U1RFE2)/(1.0+EM2/(2.0*GM00E2*FACT))**GM11E2
C
        DO 3910 I=1,NNODG2
        TEMP=DPENG2(2,I)**2+DPENG2(3,I)**2
        TEMP2=DPENG2(4,I)*DPENG2(1,I)-0.5*TEMP
        TEMP=(GM03E2*TEMP2/DPENG2(1,I))
     &                          /(1.0+TEMP/(2.0*GM00E2*TEMP2))**GM11E2
```

```
        WORKG2(I)=(PTREF-TEMP)/PTREF
3910    CONTINUE
C
        TITLG2='PT LOSS '
C
        RETURN
C
        END
```

E2DIST

```
        SUBROUTINE E2DIST(IMGL
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE STEPS THROUGH EACH CELL ON THIS LEVEL AND APPLIES
C       NI'S SCHEME BY:
C             -CALCULATING TIME STEP
C             -CALCULATING CHANGE IN U
C             -CALCULATING JACOBIANS AND CHANGES OF F AND G
C             -DISTRIBUTING CHANGES TO NODES
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2CDMN.INC'
C
        INCLUDE '[.EULR2D]E2CDMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*************************************************************************
C
C     STEP THROUGH EACH FINE CELL AT THIS LEVEL WHICH IS NEITHER JUST
C       OUTSIDE AND EMBEDDED REGION OR NEAR A BOUNDARY
C
        DO 10 ICELL=ILVLG2(1,IMGL),ILVLG2(2,IMGL)
C
C     SET UP NODE POINTERS FOR THIS CELL
C
        ISW=ICELG2(2,ICELL)
        ISE=ICELG2(4,ICELL)
        INE=ICELG2(6,ICELL)
        INW=ICELG2(8,ICELL)
C
C     GEOMETRY OF ALL CELL CORNERS
C
        XSW=GEOMG2(1,ISW)
        YSW=GEOMG2(2,ISW)
```

```
      XSE=GEONG2(1,ISE)
      YSE=GEONG2(2,ISE)
      XNE=GEONG2(1,INE)
      YNE=GEONG2(2,INE)
      XNW=GEONG2(1,INW)
      YNW=GEONG2(2,INW)
C
C     COMPUTE PROJECTIONS OF CELL FACES
C
      DXEW=0.5*(XNE+XSE-XNW-XSW)
      DYEW=0.5*(YNE+YSE-YNW-YSW)
      DXNS=0.5*(XNW+XNE-XSW-XSE)
      DYNS=0.5*(YNW+YNE-YSW-YSE)
C
      DEW= SQRT(DXEW*DXEW+DYEW*DYEW)
      DNS= SQRT(DXNS*DXNS+DYNS*DYNS)
C
C     SAVE DEPENDENT VARIABLES AND COMPUTE THE FLUX VECTORS AT ALL
C       CELL CORNERS
C
C     ...SOUTHWEST
C
      U1SW=DPENG2(1,ISW)
      U2SW=DPENG2(2,ISW)
      U3SW=DPENG2(3,ISW)
      U4SW=DPENG2(4,ISW)
C
      U21SW=U2SW/U1SW
      U31SW=U3SW/U1SW
      PSW  =GMO3E2*(U4SW-0.5*(U2SW*U21SW+U3SW*U31SW))
      HSW  =GMOOE2*(U4SW/U1SW)-GMO4E2*(U21SW*U21SW+U31SW*U31SW)
C
      F1SW=U2SW
      F2SW=U2SW*U21SW
      F3SW=U2SW*U31SW
      F4SW=U2SW*HSW
C
      G1SW=U3SW
      G2SW=U3SW*U21SW
      G3SW=U3SW*U31SW
      G4SW=U3SW*HSW
C
C     ...SOUTHEAST
C
      U1SE=DPENG2(1,ISE)
      U2SE=DPENG2(2,ISE)
      U3SE=DPENG2(3,ISE)
      U4SE=DPENG2(4,ISE)
C
      U21SE=U2SE/U1SE
      U31SE=U3SE/U1SE
      PSE  =GMO3E2*(U4SE-0.5*(U2SE*U21SE+U3SE*U31SE))
      HSE  =GMOOE2*(U4SE/U1SE)-GMO4E2*(U21SE*U21SE+U31SE*U31SE)
```

```
C
        F1SE=U2SE
        F2SE=U2SE*U21SE
        F3SE=U2SE*U31SE
        F4SE=U2SE*HSE
C
        G1SE=U3SE
        G2SE=U3SE*U21SE
        G3SE=U3SE*U31SE
        G4SE=U3SE*HSE
C
C     ...NORTHEAST
C
        U1NE=DPENG2(1,INE)
        U2NE=DPENG2(2,INE)
        U3NE=DPENG2(3,INE)
        U4NE=DPENG2(4,INE)
C
        U21NE=U2NE/U1NE
        U31NE=U3NE/U1NE
        PNE  =GM03E2*(U4NE-0.5*(U2NE*U21NE+U3NE*U31NE))
        HNE  =GM00E2*(U4NE/U1NE)-GM04E2*(U21NE*U21NE+U31NE*U31NE)
C
        F1NE=U2NE
        F2NE=U2NE*U21NE
        F3NE=U2NE*U31NE
        F4NE=U2NE*HNE
C
        G1NE=U3NE
        G2NE=U3NE*U21NE
        G3NE=U3NE*U31NE
        G4NE=U3NE*HNE
C
C     ...NORTHWEST
C
        U1NW=DPENG2(1,INW)
        U2NW=DPENG2(2,INW)
        U3NW=DPENG2(3,INW)
        U4NW=DPENG2(4,INW)
C
        U21NW=U2NW/U1NW
        U31NW=U3NW/U1NW
        PNW  =GM03E2*(U4NW-0.5*(U2NW*U21NW+U3NW*U31NW))
        HNW  =GM00E2*(U4NW/U1NW)-GM04E2*(U21NW*U21NW+U31NW*U31NW)
C
        F1NW=U2NW
        F2NW=U2NW*U21NW
        F3NW=U2NW*U31NW
        F4NW=U2NW*HNW
C
        G1NW=U3NW
        G2NW=U3NW*U21NW
        G3NW=U3NW*U31NW
```

```
        G4NW=U3NW*HNW
C
C     VISIT EACH CELL FACE, COMPUTING THE FLUX THROUGH THE FACE
C
C      ...SOUTH FACE
C
        DU1=    ((F1SW+F1SE)          )*(YSW-YSE)
     &         -((G1SW+G1SE)          )*(XSW-XSE)
        DU2=    ((F2SW+F2SE)+(PSW+PSE))*(YSW-YSE)
     &         -((G2SW+G2SE)          )*(XSW-XSE)
        DU3=    ((F3SW+F3SE)          )*(YSW-YSE)
     &         -((G3SW+G3SE)+(PSW+PSE))*(XSW-XSE)
        DU4=    ((F4SW+F4SE)          )*(YSW-YSE)
     &         -((G4SW+G4SE)          )*(XSW-XSE)
C
C      ...NORTH FACE
C
        DU1=DU1+((F1NE+F1NW)          )*(YNE-YNW)
     &         -((G1NE+G1NW)          )*(XNE-XNW)
        DU2=DU2+((F2NE+F2NW)+(PNE+PNW))*(YNE-YNW)
     &         -((G2NE+G2NW)          )*(XNE-XNW)
        DU3=DU3+((F3NE+F3NW)          )*(YNE-YNW)
     &         -((G3NE+G3NW)+(PNE+PNW))*(XNE-XNW)
        DU4=DU4+((F4NE+F4NW)          )*(YNE-YNW)
     &         -((G4NE+G4NW)          )*(XNE-XNW)
C
C      ...EAST FACE
C
        DU1=DU1+((F1SE+F1NE)          )*(YSE-YNE)
     &         -((G1SE+G1NE)          )*(XSE-XNE)
        DU2=DU2+((F2SE+F2NE)+(PSE+PNE))*(YSE-YNE)
     &         -((G2SE+G2NE)          )*(XSE-XNE)
        DU3=DU3+((F3SE+F3NE)          )*(YSE-YNE)
     &         -((G3SE+G3NE)+(PSE+PNE))*(XSE-XNE)
        DU4=DU4+((F4SE+F4NE)          )*(YSE-YNE)
     &         -((G4SE+G4NE)          )*(XSE-XNE)
C
C      ...WEST FACE
C
        DU1=DU1+((F1NW+F1SW)          )*(YNW-YSW)
     &         -((G1NW+G1SW)          )*(XNW-XSW)
        DU2=DU2+((F2NW+F2SW)+(PNW+PSW))*(YNW-YSW)
     &         -((G2NW+G2SW)          )*(XNW-XSW)
        DU3=DU3+((F3NW+F3SW)          )*(YNW-YSW)
     &         -((G3NW+G3SW)+(PNW+PSW))*(XNW-XSW)
        DU4=DU4+((F4NW+F4SW)          )*(YNW-YSW)
     &         -((G4NW+G4SW)          )*(XNW-XSW)
C
C     COMPUTE THE DEPENDENT VARIABLES AT THE CELL CENTER
C
        U1C=0.25*(U1SW+U1SE+U1NE+U1NW)
        U2C=0.25*(U2SW+U2SE+U2NE+U2NW)
        U3C=0.25*(U3SW+U3SE+U3NE+U3NW)
```

```
        U4C=0.25*(U4SW+U4SE+U4NE+U4NW)
C
C     COMPUTE COMBINATIONS OF THE DEPENDENT VARIABLES AT THE CELL CENTER
C
        U21C=U2C/U1C
        U31C=U3C/U1C
        U41C=U4C/U1C
C
C     COMPUTE THE MAXIMUM STABLE TIME STEP FOR THIS CELL
C
        RHO  =U1C
        PRES =GM03E2*(U4C-0.5*(U2C*U21C+U3C*U31C))
        SOUND=SQRT(GM00E2*PRES/RHO)
        UCNS =ABS(U21C*DYNS-U31C*DXNS)+SOUND*DNS
        UCEW =ABS(U21C*DYEW-U31C*DXEW)+SOUND*DEW
        DTVOL=CFLE2/MAX(UCNS,UCEW)
C
C     ADJUST FLUX BALANCES BY DTVOL
C
        DU1=0.5*DTVOL*DU1
        DU2=0.5*DTVOL*DU2
        DU3=0.5*DTVOL*DU3
        DU4=0.5*DTVOL*DU4
C
C     SET UP THE JACOBIAN OF F-MATRIX
C
        DFDU11= 0.00
        DFDU12= 1.00
        DFDU13= 0.00
        DFDU14 =0.00
C
        DFDU21= GM02E2*U21C*U21C+GM04E2*U31C*U31C
        DFDU22=-GM05E2*U21C
        DFDU23=-GM03E2*U31C
        DFDU24= GM03E2
C
        DFDU31=-U21C*U31C
        DFDU32= U31C
        DFDU33= U21C
        DFDU34= 0.00
C
        DFDU41=-GM00E2*U21C*U41C+GM03E2*U21C*(U21C*U21C+U31C*U31C)
        DFDU42= GM00E2*U41C      -GM06E2*U21C*U21C-GM04E2*U31C*U31C
        DFDU43=-GM03E2*U21C*U31C
        DFDU44= GM00E2*U21C
C
C     SET UP THE JACOBIAN OF G-MATRIX
C
        DGDU11= 0.00
        DGDU12= 0.00
        DGDU13= 1.00
        DGDU14 =0.00
C
```

```
      DGDU21=-U21C*U31C
      DGDU22= U31C
      DGDU23= U21C
      DGDU24= 0.00
C
      DGDU31= GM02E2*U31C*U31C+GM04E2*U21C*U21C
      DGDU32=-GM03E2*U21C
      DGDU33=-GM05E2*U31C
      DGDU34= GM03E2
C
      DGDU41=-GM00E2*U31C*U41C+GM03E2*U31C*(U21C*U21C+U31C*U31C)
      DGDU42=-GM03E2*U21C*U31C
      DGDU43= GM00E2*U41C     -GM06E2*U31C*U31C-GM04E2*U21C*U21C
      DGDU44= GM00E2*U31C
C
C     COMPUTE CHANGE IN F AND CHANGE IN G
C
      DF1=DFDU11*DU1+DFDU12*DU2+DFDU13*DU3+DFDU14*DU4
      DF2=DFDU21*DU1+DFDU22*DU2+DFDU23*DU3+DFDU24*DU4
      DF3=DFDU31*DU1+DFDU32*DU2+DFDU33*DU3+DFDU34*DU4
      DF4=DFDU41*DU1+DFDU42*DU2+DFDU43*DU3+DFDU44*DU4
C
      DG1=DGDU11*DU1+DGDU12*DU2+DGDU13*DU3+DGDU14*DU4
      DG2=DGDU21*DU1+DGDU22*DU2+DGDU23*DU3+DGDU24*DU4
      DG3=DGDU31*DU1+DGDU32*DU2+DGDU33*DU3+DGDU34*DU4
      DG4=DGDU41*DU1+DGDU42*DU2+DGDU43*DU3+DGDU44*DU4
C
C     PERFORM LOCAL TRANSFORMATION OF CONVECTIVE TERMS FOR THIS CELL
C
      DDF1=DTVOL*( DF1*DYNS-DG1*DXNS)
      DDF2=DTVOL*( DF2*DYNS-DG2*DXNS)
      DDF3=DTVOL*( DF3*DYNS-DG3*DXNS)
      DDF4=DTVOL*( DF4*DYNS-DG4*DXNS)
C
      DDG1=DTVOL*(-DF1*DYEW+DG1*DXEW)
      DDG2=DTVOL*(-DF2*DYEW+DG2*DXEW)
      DDG3=DTVOL*(-DF3*DYEW+DG3*DXEW)
      DDG4=DTVOL*(-DF4*DYEW+DG4*DXEW)
C
C     DISTRIBUTE CHANGES
C
      CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.25*(DU1-DDF1-DDG1)
      CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.25*(DU2-DDF2-DDG2)
      CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.25*(DU3-DDF3-DDG3)
      CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.25*(DU4-DDF4-DDG4)
C
      CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.25*(DU1+DDF1-DDG1)
      CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.25*(DU2+DDF2-DDG2)
      CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.25*(DU3+DDF3-DDG3)
      CHNGE2(4,ISE)=CHNGE2(4,ISE)+0.25*(DU4+DDF4-DDG4)
C
      CHNGE2(1,INE)=CHNGE2(1,INE)+0.25*(DU1+DDF1+DDG1)
      CHNGE2(2,INE)=CHNGE2(2,INE)+0.25*(DU2+DDF2+DDG2)
```

```
        CHNGE2(3,INE)=CHNGE2(3,INE)+0.25*(DU3+DDF3+DDG3)
        CHNGE2(4,INE)=CHNGE2(4,INE)+0.25*(DU4+DDF4+DDG4)
C
        CHNGE2(1,INW)=CHNGE2(1,INW)+0.25*(DU1-DDF1+DDG1)
        CHNGE2(2,INW)=CHNGE2(2,INW)+0.25*(DU2-DDF2+DDG2)
        CHNGE2(3,INW)=CHNGE2(3,INW)+0.25*(DU3-DDF3+DDG3)
        CHNGE2(4,INW)=CHNGE2(4,INW)+0.25*(DU4-DDF4+DDG4)
C
C     GO BACK FOR NEXT CELL ON THIS LEVEL
C
10      CONTINUE
C
C     STEP THROUGH EACH FINE CELL AT THIS LEVEL WHICH IS EITHER AT
C        A BOUNDARY OR JUST OUTSIDE AN EMBEDDED REGION
C
        DO 30 ICELL=ILVLG2(3,IMGL),ILVLG2(4,IMGL)
C
C     SET UP NODE POINTERS FOR THIS CELL
C
        ISW=ICELG2(2,ICELL)
        IS =ICELG2(3,ICELL)
        ISE=ICELG2(4,ICELL)
        IE =ICELG2(5,ICELL)
        INE=ICELG2(6,ICELL)
        IN =ICELG2(7,ICELL)
        INW=ICELG2(8,ICELL)
        IW =ICELG2(9,ICELL)
C
C     GEOMETRY OF ALL CELL CORNERS
C
        XSW=GEOMG2(1,ISW)
        YSW=GEOMG2(2,ISW)
        XSE=GEOMG2(1,ISE)
        YSE=GEOMG2(2,ISE)
        XNE=GEOMG2(1,INE)
        YNE=GEOMG2(2,INE)
        XNW=GEOMG2(1,INW)
        YNW=GEOMG2(2,INW)
C
C     COMPUTE PROJECTIONS OF CELL FACES
C
        DXEW=0.5*(XNE+XSE-XNW-XSW)
        DYEW=0.5*(YNE+YSE-YNW-YSW)
        DXNS=0.5*(XNW+XNE-XSW-XSE)
        DYNS=0.5*(YNW+YNE-YSW-YSE)
C
        DEW= SQRT(DXEW*DXEW+DYEW*DYEW)
        DNS= SQRT(DXNS*DXNS+DYNS*DYNS)
C
C     SAVE DEPENDENT VARIABLES AND COMPUTE THE FLUX VECTORS AT ALL
C        CELL CORNERS
C
C     ...SOUTHWEST
```

```
C
      U1SW=DPENG2(1,ISW)
      U2SW=DPENG2(2,ISW)
      U3SW=DPENG2(3,ISW)
      U4SW=DPENG2(4,ISW)
C
      U21SW=U2SW/U1SW
      U31SW=U3SW/U1SW
      PSW  =GM03E2*(U4SW-0.5*(U2SW*U21SW+U3SW*U31SW))
      HSW  =GM00E2*(U4SW/U1SW)-GM04E2*(U21SW*U21SW+U31SW*U31SW)
C
      F1SW=U2SW
      F2SW=U2SW*U21SW
      F3SW=U2SW*U31SW
      F4SW=U2SW*HSW
C
      G1SW=U3SW
      G2SW=U3SW*U21SW
      G3SW=U3SW*U31SW
      G4SW=U3SW*HSW
C
C     ...SOUTHEAST
C
      U1SE=DPENG2(1,ISE)
      U2SE=DPENG2(2,ISE)
      U3SE=DPENG2(3,ISE)
      U4SE=DPENG2(4,ISE)
C
      U21SE=U2SE/U1SE
      U31SE=U3SE/U1SE
      PSE  =GM03E2*(U4SE-0.5*(U2SE*U21SE+U3SE*U31SE))
      HSE  =GM00E2*(U4SE/U1SE)-GM04E2*(U21SE*U21SE+U31SE*U31SE)
C
      F1SE=U2SE
      F2SE=U2SE*U21SE
      F3SE=U2SE*U31SE
      F4SE=U2SE*HSE
C
      G1SE=U3SE
      G2SE=U3SE*U21SE
      G3SE=U3SE*U31SE
      G4SE=U3SE*HSE
C
C     ...NORTHEAST
C
      U1NE=DPENG2(1,INE)
      U2NE=DPENG2(2,INE)
      U3NE=DPENG2(3,INE)
      U4NE=DPENG2(4,INE)
C
      U21NE=U2NE/U1NE
      U31NE=U3NE/U1NE
      PNE  =GM03E2*(U4NE-0.5*(U2NE*U21NE+U3NE*U31NE))
```

```
      HNE  =GMOOE2*(U4NE/U1NE)-GMO4E2*(U21NE*U21NE+U31NE*U31NE)
C
      F1NE=U2NE
      F2NE=U2NE*U21NE
      F3NE=U2NE*U31NE
      F4NE=U2NE*HNE
C
      G1NE=U3NE
      G2NE=U3NE*U21NE
      G3NE=U3NE*U31NE
      G4NE=U3NE*HNE
C
C     ...NORTHWEST
C
      U1NW=DPENG2(1,INW)
      U2NW=DPENG2(2,INW)
      U3NW=DPENG2(3,INW)
      U4NW=DPENG2(4,INW)
C
      U21NW=U2NW/U1NW
      U31NW=U3NW/U1NW
      PNW  =GMO3E2*(U4NW-0.5*(U2NW*U21NW+U3NW*U31NW))
      HNW  =GMOOE2*(U4NW/U1NW)-GMO4E2*(U21NW*U21NW+U31NW*U31NW)
C
      F1NW=U2NW
      F2NW=U2NW*U21NW
      F3NW=U2NW*U31NW
      F4NW=U2NW*HNW
C
      G1NW=U3NW
      G2NW=U3NW*U21NW
      G3NW=U3NW*U31NW
      G4NW=U3NW*HNW
C
C     VISIT EACH CELL FACE, COMPUTING THE FLUX THROUGH THE FACE AS
C       WELL AS THE AVERAGE VALUES OF THE DEPENDENT VARIABLES IN
C       THE CELL.   FIRST ASSUME THAT THE CELL IS NOT JUST OUTSIDE,
C       AND THEN CORRECT IF IT IS
C
C     ...SOUTH FACE - STANDARD
C
      WALL=2.0-VDISE2(1,ICELL     )
C
      U1C=    (U1SW+U1SE)
      U2C=    (U2SW+U2SE)
      U3C=    (U3SW+U3SE)
      U4C=    (U4SW+U4SE)
C
      DU1=    ((F1SW+F1SE)*WALL          )*(YSW-YSE)
     &       -((G1SW+G1SE)*WALL          )*(XSW-XSE)
      DU2=    ((F2SW+F2SE)*WALL+(PSW+PSE))*(YSW-YSE)
     &       -((G2SW+G2SE)*WALL          )*(XSW-XSE)
      DU3=    ((F3SW+F3SE)*WALL          )*(YSW-YSE)
```

```
     &          -((G3SW+G3SE)*WALL+(PSW+PSE))*(XSW-XSE)
        DU4=     ((F4SW+F4SE)*WALL          )*(YSW-YSE)
     &          -((G4SW+G4SE)*WALL          )*(XSW-XSE)
C
C      ...SOUTH FACE - CORRECT IF JUST OUTSIDE EMBEDDED REGION
C
        IF(IS.NE.0) THEN
C
           U1S=DPENG2(1,IS)
           U2S=DPENG2(2,IS)
           U3S=DPENG2(3,IS)
           U4S=DPENG2(4,IS)
C
              U21S=U2S/U1S
              U31S=U3S/U1S
              PS  =GM03E2*(U4S-0.5*(U2S*U21S+U3S*U31S))
              HS  =GM00E2*(U4S/U1S)-GM04E2*(U21S*U21S+U31S*U31S)
C
              F1S=U2S
              F2S=U2S*U21S
              F3S=U2S*U31S
              F4S=U2S*HS
C
              G1S=U3S
              G2S=U3S*U21S
              G3S=U3S*U31S
              G4S=U3S*HS
C
              U1C=U1C-(0.5*U1SW-U1S+0.5*U1SE)
              U2C=U2C-(0.5*U2SW-U2S+0.5*U2SE)
              U3C=U3C-(0.5*U3SW-U3S+0.5*U3SE)
              U4C=U4C-(0.5*U4SW-U4S+0.5*U4SE)
C
              DU1=DU1-((0.5*F1SW-F1S+0.5*F1SE))*(YSW-YSE)
     &               +((0.5*G1SW-G1S+0.5*G1SE))*(XSW-XSE)
              DU2=DU2-((0.5*F2SW-F2S+0.5*F2SE)
     &                +(0.5*PSW -PS +0.5*PSE ))*(YSW-YSE)
     &               +((0.5*G2SW-G2S+0.5*G2SE))*(XSW-XSE)
              DU3=DU3-((0.5*F3SW-F3S+0.5*F3SE))*(YSW-YSE)
     &               +((0.5*G3SW-G3S+0.5*G3SE)
     &                +(0.5*PSW -PS +0.5*PSE ))*(XSW-XSE)
              DU4=DU4-((0.5*F4SW-F4S+0.5*F4SE))*(YSW-YSE)
     &               +((0.5*G4SW-G4S+0.5*G4SE))*(XSW-XSE)
C
        ENDIF
C
C      ...NORTH FACE - STANDARD
C
        WALL=2.0-VDISE2(3,ICELL     )
C
        U1C=U1C+(U1NE+U1NW)
        U2C=U2C+(U2NE+U2NW)
        U3C=U3C+(U3NE+U3NW)
```

```
      U4C=U4C+(U4NE+U4NW)
C
      DU1=DU1+((F1NE+F1NW)*WALL          )*(YNE-YNW)
     &       -((G1NE+G1NW)*WALL          )*(XNE-XNW)
      DU2=DU2+((F2NE+F2NW)*WALL+(PNE+PNW))*(YNE-YNW)
     &       -((G2NE+G2NW)*WALL          )*(XNE-XNW)
      DU3=DU3+((F3NE+F3NW)*WALL          )*(YNE-YNW)
     &       -((G3NE+G3NW)*WALL+(PNE+PNW))*(XNE-XNW)
      DU4=DU4+((F4NE+F4NW)*WALL          )*(YNE-YNW)
     &       -((G4NE+G4NW)*WALL          )*(XNE-XNW)
C
C     ...NORTH FACE - CORRECT IF JUST OUTSIDE EMBEDDED REGION
C
      IF(IN.NE.0) THEN
C
         U1N=DPENG2(1,IN)
         U2N=DPENG2(2,IN)
         U3N=DPENG2(3,IN)
         U4N=DPENG2(4,IN)
C
            U21N=U2N/U1N
            U31N=U3N/U1N
            PN  =GM03E2*(U4N-0.5*(U2N*U21N+U3N*U31N))
            HN  =GM00E2*(U4N/U1N)-GM04E2*(U21N*U21N+U31N*U31N)
C
            F1N=U2N
            F2N=U2N*U21N
            F3N=U2N*U31N
            F4N=U2N*HN
C
            G1N=U3N
            G2N=U3N*U21N
            G3N=U3N*U31N
            G4N=U3N*HN
C
            U1C=U1C-(0.5*U1NE-U1N+0.5*U1NW)
            U2C=U2C-(0.5*U2NE-U2N+0.5*U2NW)
            U3C=U3C-(0.5*U3NE-U3N+0.5*U3NW)
            U4C=U4C-(0.5*U4NE-U4N+0.5*U4NW)
C
            DU1=DU1-((0.5*F1NE-F1N+0.5*F1NW))*(YNE-YNW)
     &             +((0.5*G1NE-G1N+0.5*G1NW))*(XNE-XNW)
            DU2=DU2-((0.5*F2NE-F2N+0.5*F2NW)
     &              +(0.5*PNE -PN +0.5*PNW ))*(YNE-YNW)
     &             +((0.5*G2NE-G2N+0.5*G2NW))*(XNE-XNW)
            DU3=DU3-((0.5*F3NE-F3N+0.5*F3NW))*(YNE-YNW)
     &             +((0.5*G3NE-G3N+0.5*G3NW)
     &              +(0.5*PNE -PN +0.5*PNW ))*(XNE-XNW)
            DU4=DU4-((0.5*F4NE-F4N+0.5*F4NW))*(YNE-YNW)
     &             +((0.5*G4NE-G4N+0.5*G4NW))*(XNE-XNW)
C
      ENDIF
C
```

```
C      ...EAST FACE - STANDARD
C
         WALL=2.0-VDISE2(2,ICELL     )
C
         U1C=U1C+(U1SE+U1NE)
         U2C=U2C+(U2SE+U2NE)
         U3C=U3C+(U3SE+U3NE)
         U4C=U4C+(U4SE+U4NE)
C
         DU1=DU1+((F1SE+F1NE)*WALL          )*(YSE-YNE)
     &          -((G1SE+G1NE)*WALL          )*(XSE-XNE)
         DU2=DU2+((F2SE+F2NE)*WALL+(PSE+PNE))*(YSE-YNE)
     &          -((G2SE+G2NE)*WALL          )*(XSE-XNE)
         DU3=DU3+((F3SE+F3NE)*WALL          )*(YSE-YNE)
     &          -((G3SE+G3NE)*WALL+(PSE+PNE))*(XSE-XNE)
         DU4=DU4+((F4SE+F4NE)*WALL          )*(YSE-YNE)
     &          -((G4SE+G4NE)*WALL          )*(XSE-XNE)
C
C      ...EAST FACE - CORRECT IF JUST OUTSIDE EMBEDDED REGION
C
         IF(IE.NE.0) THEN
C
            U1E=DPENG2(1,IE)
            U2E=DPENG2(2,IE)
            U3E=DPENG2(3,IE)
            U4E=DPENG2(4,IE)
C
               U21E=U2E/U1E
               U31E=U3E/U1E
               PE  =GM03E2*(U4E-0.5*(U2E*U21E+U3E*U31E))
               HE  =GM00E2*(U4E/U1E)-GM04E2*(U21E*U21E+U31E*U31E)
C
               F1E=U2E
               F2E=U2E*U21E
               F3E=U2E*U31E
               F4E=U2E*HE
C
               G1E=U3E
               G2E=U3E*U21E
               G3E=U3E*U31E
               G4E=U3E*HE
C
               U1C=U1C-(0.5*U1SE-U1E+0.5*U1NE)
               U2C=U2C-(0.5*U2SE-U2E+0.5*U2NE)
               U3C=U3C-(0.5*U3SE-U3E+0.5*U3NE)
               U4C=U4C-(0.5*U4SE-U4E+0.5*U4NE)
C
               DU1=DU1-((0.5*F1SE-F1E+0.5*F1NE))*(YSE-YNE)
     &                +((0.5*G1SE-G1E+0.5*G1NE))*(XSE-XNE)
               DU2=DU2-((0.5*F2SE-F2E+0.5*F2NE)
     &                 +(0.5*PSE -PE +0.5*PNE ))*(YSE-YNE)
     &                +((0.5*G2SE-G2E+0.5*G2NE))*(XSE-XNE)
               DU3=DU3-((0.5*F3SE-F3E+0.5*F3NE))*(YSE-YNE)
```

```
     &               +((0.5*G3SE-G3E+0.5*G3NE)
     &                +(0.5*PSE -PE +0.5*PNE ))*(XSE-XNE)
               DU4=DU4-((0.5*F4SE-F4E+0.5*F4NE))*(YSE-YNE)
     &               +((0.5*G4SE-G4E+0.5*G4NE))*(XSE-XNE)
C
        ENDIF
C
C     ...WEST FACE - STANDARD
C
        WALL=2.0-VDISE2(4,ICELL     )
C
        U1C=U1C+(U1NW+U1SW)
        U2C=U2C+(U2NW+U2SW)
        U3C=U3C+(U3NW+U3SW)
        U4C=U4C+(U4NW+U4SW)
C
        DU1=DU1+((F1NW+F1SW)*WALL           )*(YNW-YSW)
     &         -((G1NW+G1SW)*WALL           )*(XNW-XSW)
        DU2=DU2+((F2NW+F2SW)*WALL+(PNW+PSW))*(YNW-YSW)
     &         -((G2NW+G2SW)*WALL           )*(XNW-XSW)
        DU3=DU3+((F3NW+F3SW)*WALL           )*(YNW-YSW)
     &         -((G3NW+G3SW)*WALL+(PNW+PSW))*(XNW-XSW)
        DU4=DU4+((F4NW+F4SW)*WALL           )*(YNW-YSW)
     &         -((G4NW+G4SW)*WALL           )*(XNW-XSW)
C
C     ...WEST FACE - CORRECT IF JUST OUTSIDE EMBEDDED REGION
C
        IF(IW.NE.0) THEN
C
           U1W=DPENG2(1,IW)
           U2W=DPENG2(2,IW)
           U3W=DPENG2(3,IW)
           U4W=DPENG2(4,IW)
C
              U21W=U2W/U1W
              U31W=U3W/U1W
              PW  =GM03E2*(U4W-0.5*(U2W*U21W+U3W*U31W))
              HW  =GM00E2*(U4W/U1W)-GM04E2*(U21W*U21W+U31W*U31W)
C
              F1W=U2W
              F2W=U2W*U21W
              F3W=U2W*U31W
              F4W=U2W*HW
C
              G1W=U3W
              G2W=U3W*U21W
              G3W=U3W*U31W
              G4W=U3W*HW
C
              U1C=U1C-(0.5*U1NW-U1W+0.5*U1SW)
              U2C=U2C-(0.5*U2NW-U2W+0.5*U2SW)
              U3C=U3C-(0.5*U3NW-U3W+0.5*U3SW)
              U4C=U4C-(0.5*U4NW-U4W+0.5*U4SW)
```

```
C
            DU1=DU1-((0.5*F1NW-F1W+0.5*F1SW))*(YNW-YSW)
     &             +((0.5*G1NW-G1W+0.5*G1SW))*(XNW-XSW)
            DU2=DU2-((0.5*F2NW-F2W+0.5*F2SW)
     &              +(0.5*PNW -PW +0.5*PSW ))*(YNW-YSW)
     &             +((0.5*G2NW-G2W+0.5*G2SW))*(XNW-XSW)
            DU3=DU3-((0.5*F3NW-F3W+0.5*F3SW))*(YNW-YSW)
     &             +((0.5*G3NW-G3W+0.5*G3SW)
     &              +(0.5*PNW -PW +0.5*PSW ))*(XNW-XSW)
            DU4=DU4-((0.5*F4NW-F4W+0.5*F4SW))*(YNW-YSW)
     &             +((0.5*G4NW-G4W+0.5*G4SW))*(XNW-XSW)
C
         ENDIF
C
C     SET UP FINAL VALUES OF DEPENDENT VARIABLES AT THE CELL CENTER
C
         U1C=0.125*U1C
         U2C=0.125*U2C
         U3C=0.125*U3C
         U4C=0.125*U4C
C
C     COMPUTE COMBINATIONS OF THE DEPENDENT VARIABLES AT THE CELL CENTER
C
         U21C=U2C/U1C
         U31C=U3C/U1C
         U41C=U4C/U1C
C
C     COMPUTE THE MAXIMUM STABLE TIME STEP FOR THIS CELL
C
         RHO  =U1C
         PRES =GM03E2*(U4C-0.5*(U2C*U21C+U3C*U31C))
         SOUND=SQRT(GM00E2*PRES/RHO)
         UCNS =ABS(U21C*DYNS-U31C*DXNS)+SOUND*DNS
         UCEW =ABS(U21C*DYEW-U31C*DXEW)+SOUND*DEW
         DTVOL=CFLE2/MAX(UCNS,UCEW)
C
C     ADJUST FLUX BALANCES BY DTVOL
C
         DU1=0.5*DTVOL*DU1
         DU2=0.5*DTVOL*DU2
         DU3=0.5*DTVOL*DU3
         DU4=0.5*DTVOL*DU4
C
C     SET UP THE JACOBIAN OF F-MATRIX
C
         DFDU11= 0.00
         DFDU12= 1.00
         DFDU13= 0.00
         DFDU14 =0.00
C
         DFDU21= GM02E2*U21C*U21C+GM04E2*U31C*U31C
         DFDU22=-GM05E2*U21C
         DFDU23=-GM03E2*U31C
```

```
      DFDU24= GM03E2
C
      DFDU31=-U21C*U31C
      DFDU32= U31C
      DFDU33= U21C
      DFDU34= 0.00
C
      DFDU41=-GM00E2*U21C*U41C+GM03E2*U21C*(U21C*U21C+U31C*U31C)
      DFDU42= GM00E2*U41C      -GM06E2*U21C*U21C-GM04E2*U31C*U31C
      DFDU43=-GM03E2*U21C*U31C
      DFDU44= GM00E2*U21C
C
C     SET UP THE JACOBIAN OF G-MATRIX
C
      DGDU11= 0.00
      DGDU12= 0.00
      DGDU13= 1.00
      DGDU14 =0.00
C
      DGDU21=-U21C*U31C
      DGDU22= U31C
      DGDU23= U21C
      DGDU24= 0.00
C
      DGDU31= GM02E2*U31C*U31C+GM04E2*U21C*U21C
      DGDU32=-GM03E2*U21C
      DGDU33=-GM06E2*U31C
      DGDU34= GM03E2
C
      DGDU41=-GM00E2*U31C*U41C+GM03E2*U31C*(U21C*U21C+U31C*U31C)
      DGDU42=-GM03E2*U21C*U31C
      DGDU43= GM00E2*U41C      -GM06E2*U31C*U31C-GM04E2*U21C*U21C
      DGDU44= GM00E2*U31C
C
C     COMPUTE CHANGE IN F AND CHANGE IN G
C
      DF1=DFDU11*DU1+DFDU12*DU2+DFDU13*DU3+DFDU14*DU4
      DF2=DFDU21*DU1+DFDU22*DU2+DFDU23*DU3+DFDU24*DU4
      DF3=DFDU31*DU1+DFDU32*DU2+DFDU33*DU3+DFDU34*DU4
      DF4=DFDU41*DU1+DFDU42*DU2+DFDU43*DU3+DFDU44*DU4
C
      DG1=DGDU11*DU1+DGDU12*DU2+DGDU13*DU3+DGDU14*DU4
      DG2=DGDU21*DU1+DGDU22*DU2+DGDU23*DU3+DGDU24*DU4
      DG3=DGDU31*DU1+DGDU32*DU2+DGDU33*DU3+DGDU34*DU4
      DG4=DGDU41*DU1+DGDU42*DU2+DGDU43*DU3+DGDU44*DU4
C
C     PERFORM LOCAL TRANSFORMATION OF CONVECTIVE TERMS FOR THIS CELL
C
      DDF1=DTVOL*( DF1*DYDS-DG1*DXDS)
      DDF2=DTVOL*( DF2*DYDS-DG2*DXDS)
      DDF3=DTVOL*( DF3*DYDS-DG3*DXDS)
      DDF4=DTVOL*( DF4*DYDS-DG4*DXDS)
C
```

```
        DDG1=DTVOL*(-DF1*DYEW+DG1*DXEW)
        DDG2=DTVOL*(-DF2*DYEW+DG2*DXEW)
        DDG3=DTVOL*(-DF3*DYEW+DG3*DXEW)
        DDG4=DTVOL*(-DF4*DYEW+DG4*DXEW)
C
C     DISTRIBUTE CHANGES
C
          COEF=0.25*VDISE2(4,ICELL     )*VDISE2(1,ICELL     )
C
          CHNGE2(1,ISW)=CHNGE2(1,ISW)+COEF*(DU1-DDF1-DDG1)
          CHNGE2(2,ISW)=CHNGE2(2,ISW)+COEF*(DU2-DDF2-DDG2)
          CHNGE2(3,ISW)=CHNGE2(3,ISW)+COEF*(DU3-DDF3-DDG3)
          CHNGE2(4,ISW)=CHNGE2(4,ISW)+COEF*(DU4-DDF4-DDG4)
C
        IF(IS.NE.0) THEN
          COEF=0.25*VDISE2(1,ICELL     )
C
          CHNGE2(1,IS )=CHNGE2(1,IS )+COEF*(DU1     -DDG1)
          CHNGE2(2,IS )=CHNGE2(2,IS )+COEF*(DU2     -DDG2)
          CHNGE2(3,IS )=CHNGE2(3,IS )+COEF*(DU3     -DDG3)
          CHNGE2(4,IS )=CHNGE2(4,IS )+COEF*(DU4     -DDG4)
        ENDIF
C
          COEF=0.25*VDISE2(1,ICELL     )*VDISE2(2,ICELL     )
C
          CHNGE2(1,ISE)=CHNGE2(1,ISE)+COEF*(DU1+DDF1-DDG1)
          CHNGE2(2,ISE)=CHNGE2(2,ISE)+COEF*(DU2+DDF2-DDG2)
          CHNGE2(3,ISE)=CHNGE2(3,ISE)+COEF*(DU3+DDF3-DDG3)
          CHNGE2(4,ISE)=CHNGE2(4,ISE)+COEF*(DU4+DDF4-DDG4)
C
        IF(IE.NE.0) THEN
          COEF=0.25*VDISE2(2,ICELL     )
C
          CHNGE2(1,IE )=CHNGE2(1,IE )+COEF*(DU1+DDF1     )
          CHNGE2(2,IE )=CHNGE2(2,IE )+COEF*(DU2+DDF2     )
          CHNGE2(3,IE )=CHNGE2(3,IE )+COEF*(DU3+DDF3     )
          CHNGE2(4,IE )=CHNGE2(4,IE )+COEF*(DU4+DDF4     )
        ENDIF
C
          COEF=0.25*VDISE2(2,ICELL     )*VDISE2(3,ICELL     )
C
          CHNGE2(1,INE)=CHNGE2(1,INE)+COEF*(DU1+DDF1+DDG1)
          CHNGE2(2,INE)=CHNGE2(2,INE)+COEF*(DU2+DDF2+DDG2)
          CHNGE2(3,INE)=CHNGE2(3,INE)+COEF*(DU3+DDF3+DDG3)
          CHNGE2(4,INE)=CHNGE2(4,INE)+COEF*(DU4+DDF4+DDG4)
C
        IF(IN.NE.0) THEN
          COEF=0.25*VDISE2(3,ICELL     )
C
          CHNGE2(1,IN )=CHNGE2(1,IN )+COEF*(DU1     +DDG1)
          CHNGE2(2,IN )=CHNGE2(2,IN )+COEF*(DU2     +DDG2)
          CHNGE2(3,IN )=CHNGE2(3,IN )+COEF*(DU3     +DDG3)
          CHNGE2(4,IN )=CHNGE2(4,IN )+COEF*(DU4     +DDG4)
```

```
        ENDIF
C
          COEF=0.25*VDISE2(3,ICELL     )*VDISE2(4,ICELL     )
C
          CHNGE2(1,INW)=CHNGE2(1,INW)+COEF*(DU1-DDF1+DDG1)
          CHNGE2(2,INW)=CHNGE2(2,INW)+COEF*(DU2-DDF2+DDG2)
          CHNGE2(3,INW)=CHNGE2(3,INW)+COEF*(DU3-DDF3+DDG3)
          CHNGE2(4,INW)=CHNGE2(4,INW)+COEF*(DU4-DDF4+DDG4)
C
        IF(IW.NE.0) THEN
          COEF=0.25*VDISE2(4,ICELL     )
C
          CHNGE2(1,IW )=CHNGE2(1,IW )+COEF*(DU1-DDF1     )
          CHNGE2(2,IW )=CHNGE2(2,IW )+COEF*(DU2-DDF2     )
          CHNGE2(3,IW )=CHNGE2(3,IW )+COEF*(DU3-DDF3     )
          CHNGE2(4,IW )=CHNGE2(4,IW )+COEF*(DU4-DDF4     )
        ENDIF
C
C     GO BACK FOR NEXT CELL ON THIS LEVEL
C
30      CONTINUE
C
C     STEP THROUGH COARSE EACH CELL AT THIS LEVEL
C
        DO 50 ICELL=ILVLG2(5,IMGL),ILVLG2(6,IMGL)
C
C     SET UP NODE POINTERS FOR THIS CELL
C
          ICENT=ICELG2(1,ICELL)
          ISW  =ICELG2(2,ICELL)
          ISE  =ICELG2(4,ICELL)
          INE  =ICELG2(6,ICELL)
          INW  =ICELG2(8,ICELL)
C
C     GEOMETRY OF ALL CELL CORNERS
C
          XSW=GEOMG2(1,ISW)
          YSW=GEOMG2(2,ISW)
          XSE=GEOMG2(1,ISE)
          YSE=GEOMG2(2,ISE)
          XNE=GEOMG2(1,INE)
          YNE=GEOMG2(2,INE)
          XNW=GEOMG2(1,INW)
          YNW=GEOMG2(2,INW)
C
C     COMPUTE PROJECTIONS OF CELL FACES
C
          DXEW=0.5*(XNE+XSE-XNW-XSW)
          DYEW=0.5*(YNE+YSE-YNW-YSW)
          DXNS=0.5*(XNW+XNE-XSW-XSE)
          DYNS=0.5*(YNW+YNE-YSW-YSE)
C
          DEW=SQRT(DXEW*DXEW+DYEW*DYEW)
```

```
        DNS=SQRT(DXNS*DXNS+DYNS*DYNS)
C
C     TRANSPORT CHANGES FROM FINE MESH TO CENTER OF THE CELL.  ALSO
C       SAVE THE VALUES OF THE DEPENDENT VARIABLES AT THE CELL CENTER
C
        U1C=DPENG2(1,ICENT)
        U2C=DPENG2(2,ICENT)
        U3C=DPENG2(3,ICENT)
        U4C=DPENG2(4,ICENT)
C
        DU1=CHNGE2(1,ICENT)
        DU2=CHNGE2(2,ICENT)
        DU3=CHNGE2(3,ICENT)
        DU4=CHNGE2(4,ICENT)
C
C     COMPUTE COMBINATIONS OF THE DEPENDENT VARIABLES AT THE CELL CENTER
C
        U21C=U2C/U1C
        U31C=U3C/U1C
        U41C=U4C/U1C
C
C     COMPUTE THE MAXIMUM STABLE TIME STEP FOR THIS CELL
C
        RHO  =U1C
        PRES =GM03E2*(U4C-0.5*(U2C*U21C+U3C*U31C))
        SOUND=SQRT(GM00E2*PRES/RHO)
        UCNS =ABS(U21C*DYNS-U31C*DXNS)+SOUND*DNS
        UCEW =ABS(U21C*DYEW-U31C*DXEW)+SOUND*DEW
        DTVOL=CFLE2/MAX(UCNS,UCEW)
C
C     SET UP THE JACOBIAN OF F-MATRIX
C
        DFDU11= 0.00
        DFDU12= 1.00
        DFDU13= 0.00
        DFDU14 =0.00
C
        DFDU21= GM02E2*U21C*U21C+GM04E2*U31C*U31C
        DFDU22=-GM05E2*U21C
        DFDU23=-GM03E2*U31C
        DFDU24= GM03E2
C
        DFDU31=-U21C*U31C
        DFDU32= U31C
        DFDU33= U21C
        DFDU34= 0.00
C
        DFDU41=-GM00E2*U21C*U41C+GM03E2*U21C*(U21C*U21C+U31C*U31C)
        DFDU42= GM00E2*U41C      -GM06E2*U21C*U21C-GM04E2*U31C*U31C
        DFDU43=-GM03E2*U21C*U31C
        DFDU44= GM00E2*U21C
C
C     SET UP THE JACOBIAN OF G-MATRIX
```

```
C
        DGDU11= 0.00
        DGDU12= 0.00
        DGDU13= 1.00
        DGDU14 =0.00
C
        DGDU21=-U21C*U31C
        DGDU22= U31C
        DGDU23= U21C
        DGDU24= 0.00
C
        DGDU31= GM02E2*U31C*U31C+GM04E2*U21C*U21C
        DGDU32=-GM03E2*U21C
        DGDU33=-GM05E2*U31C
        DGDU34= GM03E2
C
        DGDU41=-GM00E2*U31C*U41C+GM03E2*U31C*(U21C*U21C+U31C*U31C)
        DGDU42=-GM03E2*U21C*U31C
        DGDU43= GM00E2*U41C     -GM06E2*U31C*U31C-GM04E2*U21C*U21C
        DGDU44= GM00E2*U31C
C
C     COMPUTE CHANGE IN F AND CHANGE IN G
C
        DF1=DFDU11*DU1+DFDU12*DU2+DFDU13*DU3+DFDU14*DU4
        DF2=DFDU21*DU1+DFDU22*DU2+DFDU23*DU3+DFDU24*DU4
        DF3=DFDU31*DU1+DFDU32*DU2+DFDU33*DU3+DFDU34*DU4
        DF4=DFDU41*DU1+DFDU42*DU2+DFDU43*DU3+DFDU44*DU4
C
        DG1=DGDU11*DU1+DGDU12*DU2+DGDU13*DU3+DGDU14*DU4
        DG2=DGDU21*DU1+DGDU22*DU2+DGDU23*DU3+DGDU24*DU4
        DG3=DGDU31*DU1+DGDU32*DU2+DGDU33*DU3+DGDU34*DU4
        DG4=DGDU41*DU1+DGDU42*DU2+DGDU43*DU3+DGDU44*DU4
C
C     PERFORM LOCAL TRANSFORMATION OF CONVECTIVE TERMS FOR THIS CELL
C
        DDF1=DTVOL*( DF1*DYNS-DG1*DXNS)
        DDF2=DTVOL*( DF2*DYNS-DG2*DXNS)
        DDF3=DTVOL*( DF3*DYNS-DG3*DXNS)
        DDF4=DTVOL*( DF4*DYNS-DG4*DXNS)
C
        DDG1=DTVOL*(-DF1*DYEW+DG1*DXEW)
        DDG2=DTVOL*(-DF2*DYEW+DG2*DXEW)
        DDG3=DTVOL*(-DF3*DYEW+DG3*DXEW)
        DDG4=DTVOL*(-DF4*DYEW+DG4*DXEW)
C
C     DISTRIBUTE CHANGES
C
        COEF=0.25*VDISE2(4,ICELL     )*VDISE2(1,ICELL     )
C
        CHNGE2(1,ISW)=CHNGE2(1,ISW)+COEF*(DU1-DDF1-DDG1)
        CHNGE2(2,ISW)=CHNGE2(2,ISW)+COEF*(DU2-DDF2-DDG2)
        CHNGE2(3,ISW)=CHNGE2(3,ISW)+COEF*(DU3-DDF3-DDG3)
        CHNGE2(4,ISW)=CHNGE2(4,ISW)+COEF*(DU4-DDF4-DDG4)
```

```
C
        COEF=0.25*VDISE2(1,ICELL     )*VDISE2(2,ICELL     )
C
        CHNGE2(1,ISE)=CHNGE2(1,ISE)+COEF*(DU1+DDF1-DDG1)
        CHNGE2(2,ISE)=CHNGE2(2,ISE)+COEF*(DU2+DDF2-DDG2)
        CHNGE2(3,ISE)=CHNGE2(3,ISE)+COEF*(DU3+DDF3-DDG3)
        CHNGE2(4,ISE)=CHNGE2(4,ISE)+COEF*(DU4+DDF4-DDG4)
C
        COEF=0.25*VDISE2(2,ICELL     )*VDISE2(3,ICELL     )
C
        CHNGE2(1,INE)=CHNGE2(1,INE)+COEF*(DU1+DDF1+DDG1)
        CHNGE2(2,INE)=CHNGE2(2,INE)+COEF*(DU2+DDF2+DDG2)
        CHNGE2(3,INE)=CHNGE2(3,INE)+COEF*(DU3+DDF3+DDG3)
        CHNGE2(4,INE)=CHNGE2(4,INE)+COEF*(DU4+DDF4+DDG4)
C
        COEF=0.25*VDISE2(3,ICELL     )*VDISE2(4,ICELL     )
C
        CHNGE2(1,INW)=CHNGE2(1,INW)+COEF*(DU1-DDF1+DDG1)
        CHNGE2(2,INW)=CHNGE2(2,INW)+COEF*(DU2-DDF2+DDG2)
        CHNGE2(3,INW)=CHNGE2(3,INW)+COEF*(DU3-DDF3+DDG3)
        CHNGE2(4,INW)=CHNGE2(4,INW)+COEF*(DU4-DDF4+DDG4)
C
C     GO BACK FOR NEXT CELL ON THIS LEVEL
C
50      CONTINUE
C
        RETURN
C
        END
```

E2FORC

```
        SUBROUTINE E2FORC
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE COMPUTES THE LIFT AND DRAG COEFFICIENT FOR A
C        PROFILE WHOSE SURFACE IS GIVEN BY BOUNDARY-NODE PAIRS WHICH
C        ARE NEITHER IN THE FREE-STREAM NOR DIRICHLET
C
C*****************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
C*****************************************************************
C
        IF(CHRDE2.LE.0.0) THEN
           CLFTE2=0.00
```

```
            CDRGE2=0.00
            RETURN
          ENDIF
C
C     CALCULATE THE REFERENCE DYNAMIC PRESSURE
C
          QREF=0.50*(U2RFE2*U2RFE2+U3RFE2*U3RFE2)/U1RFE2
C
C     INITIALIZE INTEGRAL CALCULATIONS
C
          CX=0.00
          CY=0.00
C
C     LOOP THRUOGH THE NODE-PAIRS, COLLECTING THE INTEGRALS
C
          DO 10 II=1,NFORE2
C
          ILEFT=IFORE2(1,II)
          XLEFT=GEOMG2(1,ILEFT)
          YLEFT=GEOMG2(2,ILEFT)
          QLEFT=0.5*(DPENG2(2,ILEFT)**2+DPENG2(3,ILEFT)**2)
     &                                                  /DPENG2(1,ILEFT)
          CPLEFT=GMO3E2*((DPENG2(4,ILEFT)-QLEFT)-(U4RFE2-QREF))/QREF
C
          IRITE=IFORE2(2,II)
          XRITE=GEOMG2(1,IRITE)
          YRITE=GEOMG2(2,IRITE)
          QRITE=0.5*(DPENG2(2,IRITE)**2+DPENG2(3,IRITE)**2)
     &                                                  /DPENG2(1,IRITE)
          CPRITE=GMO3E2*((DPENG2(4,IRITE)-QRITE)-(U4RFE2-QREF))/QREF
C
C     ACCUMULATE THE INTEGRALS
C
          CX=CX+0.50*(CPLEFT+CPRITE)*(YRITE-YLEFT)
          CY=CY-0.50*(CPLEFT+CPRITE)*(XRITE-XLEFT)
C
10        CONTINUE
C
C     CALCULATE THE NORMALIZING QUANTITIES
C
          COSANG=U2RFE2/SQRT(U2RFE2*U2RFE2+U3RFE2*U3RFE2)
          SINANG=U3RFE2/SQRT(U2RFE2*U2RFE2+U3RFE2*U3RFE2)
C
C     CALCULATE THE LIFT AND DRAG COEFFICIENTS
C
          CLFTE2=(CY*COSANG-CX*SINANG)/CHRDE2
          CDRGE2=(CY*SINANG+CX*COSANG)/CHRDE2
C
          RETURN
          END
```

E2HSPL

```
      SUBROUTINE E2HSPL(IXY
     &
     &                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PLOTS HISTORY INFORMATION WHERE IXY INDICATES
C       THE ABSCISSA AND ORDINATE BY:
C           1 => ITERATN
C           2 => CPU TIME
C           3 => MAX DU2
C           4 => IMAX DU2
C           5 => AVG DU2
C           6 => RMS DU2          IF IXY.LT.0
C           7 => C-LIFT               AUTO-HARD
C           8 => C-DRAG
C           9 => SMTH (MAX)
C
C*********************************************************************
C
      INCLUDE '[.UTILITY]UTCOMN.INC'
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
      DIMENSION CONV(10),ILTYPE(3),ISTYPE(3),NPERLN(3)
C
      CHARACTER*47 PLTIT
      CHARACTER*27 PLTITL
      CHARACTER*8  PLTITS(9)
C
      PARAMETER (MFLAGS=20)
C
      DATA ILTYPE / 0, 0, 1/
      DATA ISTYPE / 2, 0, 0/
C
      DATA PLTITL/'CONVERGENCE HISTORY'/
C
      DATA PLTITS/'ITERATN ','CPU TIME',' MAX DU2','IMAX DU2',
     &            ' AVG DU2',' RMS DU2',' C-LIFT ',' C-DRAG ',
     &            'SMTH MAX'                                  /
C
C*********************************************************************
C
C*****PLOT CONVERGENCE HISTORY
C
```

```
      JOPT=ABS(IXY)
C
      IOPTX=     JOPT /10
      IOPTY=JOPT-IOPTX*10
C
      IF(IOPTX.LT.1 .OR. IOPTX.GT.9) RETURN
      IF(IOPTY.LT.1 .OR. IOPTY.GT.9) RETURN
C
      CALL UTBASE(+1,20000,IXPLOT)
      CALL UTBASE(+1,20000,IYPLOT)
C
C     READ HISTORY FILE
C
      REWIND IHSTE2
C
      NRECS =MFLAGS
      NFLAGS=0
C
      DO 20 I=1,(20000-MFLAGS)
C
      READ(IHSTE2,10,END=30) CONV
10    FORMAT(F5.0,2F10.0,F5.0,6F10.0)
C
C     IF ITERATION NUMBER IS 0, THEN DUMP PREVIOUS ITERATION'S
C        SOLUTION INTO THE FIRST MFLAGS SLOTS, OTHERWISE ADD RECORD TO
C        END OF THE LIST
C
      IF(CONV(1).EQ.0.0) THEN
         NFLAGS=MIN((NFLAGS+1),MFLAGS)
         VARUT(NFLAGS+IXPLOT)=VARUT(NRECS+IXPLOT)
         VARUT(NFLAGS+IYPLOT)=VARUT(NRECS+IYPLOT)
      ELSE
         NRECS=NRECS+1
         VARUT(NRECS+IXPLOT)=CONV(IOPTX)
         VARUT(NRECS+IYPLOT)=CONV(IOPTY)
      ENDIF
C
20    CONTINUE
C
C     REPEAT FIRST POINT INTO REMAINDER OF FIRST MFLAGS SLOTS
C
30    DO 40 I=NFLAGS+1,MFLAGS
      VARUT(I+IXPLOT)=VARUT(MFLAGS+1+IXPLOT)
      VARUT(I+IYPLOT)=VARUT(MFLAGS+1+IYPLOT)
40    CONTINUE
C
C     SET UP THE NUMBER OF POINTS IN EACH LINE
C
      NPERLN(1)=NFLAGS
      NPERLN(2)=MFLAGS-NFLAGS
      NPERLN(3)=NRECS -MFLAGS
C
C     PLOT
```

```
C
      PLTIT=PLTITS(IOPTX)//PLTITS(IOPTY)//PLTITL
C
      IF(IXY.GT.0) THEN
         INDGR=29
      ELSE
         INDGR=13
      ENDIF
C
      CALL GRLINE(ILTYPE,ISTYPE,3,PLTIT,INDGR,
     &                        VARUT(1+IXPLOT),VARUT(1+IYPLOT),NPERLN)
C
C     RELEASE BASED VARIABLES
C
      CALL UTBASE(-1,0,IXPLOT)
      CALL UTBASE(-1,0,IYPLOT)
C
      RETURN
      END
```

E2HSPR

```
      SUBROUTINE E2HSPR(IDIV
     &
     &                        )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PRINTS HISTORY INFORMATION TO JTERMO AND TO
C       JPRINT IF THE ITERATION IS EVENLY DIVISIBLE BY IDIV
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
      IF(ITERE2/IDIV*IDIV.NE.ITERE2) RETURN
C
      WRITE(JTERMO,10) ITERE2,DU2ME2,IDU2E2,DU2AE2,CLFTE2,CDRGE2
10    FORMAT(' ITER',I5,' MAX(DU2)=',E12.5,' (',I5,') AVG=',E12.5,
     &       ' CL=',F7.4,' CD=',F7.4)
C
      WRITE(JPRINT,20) ITERE2,DU2ME2,IDU2E2,DU2AE2,
     &                 CLFTE2,CDRGE2,SMINE2,SMAXE2
20    FORMAT(' ITER',I5,' MAX(DU2)=',E12.5,' (',I5,') AVG=',E12.5,
```

```
     &         ' CL=',F7.4,' CD=',F7.4,' SMTH MN:MX=',2F8.5        )
C
      RETURN
      END
```

E2INIT

```
      SUBROUTINE E2INIT(IOPT
     &
     &                        )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS THE CONSTANTS FOR THE EULER EQUATION
C       SOLVER AND COMPUTES THE NEEDED COMBINATIONS OF THE RATIO
C       OF SPECIFIC HEATS
C
C       IOPT =0 READ FROM JCARDS AND PRINT
C            =1 READ FROM JTERMI AND PRINT
C            =2 COMPLETE AUXILIARY INITIALIZATION
C
C     RECOMMENDATIONS FOR FREE STREAM INPUTS:
C       U1RFE2=1.00
C       U2RFE2=COS(ALPHA)
C       U3RFE2=SIN(ALPHA)
C       U4RFE2=0.50+1.0/(GMOOE2*(GMOOE2-1.00)*MACH*MACH)
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
      IF(IOPT.EQ.1) GOTO 20
C
C     READ INPUTS FROM INPUT FILE
C
      READ(JCARDS,10)U1RFE2,U2RFE2,U3RFE2,U4RFE2,CLFTE2,CDRGE2,CHRDE2,
     &               AMUE2 ,GMOOE2,CFLE2 ,NWNDE2
10    FORMAT(7F10.0/6F10.0,I5)
C
      ITERE2=0
      GOTO 40
C
C     CHANGE INPUTS FROM TERMINAL
C
```

```
20      WRITE(JTERMO,30) U1RFE2,    GMOOE2,
     &                   U2RFE2,    CLFTE2,
     &                   U3RFE2,    CDRGE2,
     &                   U4RFE2,    CHRDE2,
     &                   AMUE2(1),  CFLE2,
     &                   AMUE2(2),  ITERE2,
     &                   AMUE2(3),  NWNDE2,
     &                   AMUE2(4)
30      FORMAT(' 1. U1RFE2 =',F10.5,'     10. GMOOE2 =',F10.5/
     &         ' 2. U2RFE2 =',F10.5,'     11. CLFTE2 =',F10.5/
     &         ' 3. U3RFE2 =',F10.5,'     12. CDRGE2 =',F10.5/
     &         ' 4. U4RFE2 =',F10.5,'     13. CHRDE2 =',F10.5/
     &         ' 5. AMUE2.1=',F10.5,'     14. CFLE2  =',F10.5/
     &         ' 6. AMUE2.2=',F10.5,'     15. ITERE2 =', I10 /
     &         ' 7. AMUE2.3=',F10.5,'     16. NWNDE2 =', I10 /
     &         ' 8. AMUE2.4=',F10.5                          /)
C
        CALL UTINPI('VARIABLE NUMBER',IVAR)
C
        IF(IVAR.EQ.0) GOTO 40
C
        IF(IVAR.EQ. 1) CALL UTINPF('U1RFE2 ',U1RFE2  )
        IF(IVAR.EQ. 2) CALL UTINPF('U2RFE2 ',U2RFE2  )
        IF(IVAR.EQ. 3) CALL UTINPF('U3RFE2 ',U3RFE2  )
        IF(IVAR.EQ. 4) CALL UTINPF('U4RFE2 ',U4RFE2  )
        IF(IVAR.EQ. 5) CALL UTINPF('AMUE2.1',AMUE2(1))
        IF(IVAR.EQ. 6) CALL UTINPF('AMUE2.2',AMUE2(2))
        IF(IVAR.EQ. 7) CALL UTINPF('AMUE2.3',AMUE2(3))
        IF(IVAR.EQ. 8) CALL UTINPF('AMUE2.4',AMUE2(4))
        IF(IVAR.EQ.10) CALL UTINPF('GMOOE2 ',GMOOE2  )
        IF(IVAR.EQ.11) CALL UTINPF('CLFTE2 ',CLFTE2  )
        IF(IVAR.EQ.12) CALL UTINPF('CDRGE2 ',CDRGE2  )
        IF(IVAR.EQ.13) CALL UTINPF('CHRDE2 ',CHRDE2  )
        IF(IVAR.EQ.14) CALL UTINPF('CFLE2  ',CFLE2   )
        IF(IVAR.EQ.15) CALL UTINPI('ITERE2 ',ITERE2  )
        IF(IVAR.EQ.16) CALL UTINPI('NWNDE2 ',NWNDE2  )
C
        GOTO 20
C
C     PRINT OUT PARAMETERS
C
40      WRITE(JPRINT,50) U1RFE2,   GMOOE2,
     &                   U2RFE2,   CLFTE2,
     &                   U3RFE2,   CDRGE2,
     &                   U4RFE2,   CHRDE2,
     &                   AMUE2(1), CFLE2 ,
     &                   AMUE2(2), ITERE2,
     &                   AMUE2(3), NWNDE2,
     &                   AMUE2(4)
50      FORMAT(' 2-D EULER EQUATION INITIALIZED'/
     &        10X,' U1RFE2 = ',F10.5,10X,' GMOOE2 = ',F10.5/
     &        10X,' U2RFE2 = ',F10.5,10X,' CLFTE2 = ',F10.5/
     &        10X,' U3RFE2 = ',F10.5,10X,' CDRGE2 = ',F10.5/
```

```
     &         10X,' U4RFE2 = ',F10.5,10X,' CHRDE2 = ',F10.5/
     &         10X,' AMUE2.1= ',F10.5,10X,' CFLE2  = ',F10.5/
     &         10X,' AMUE2.2= ',F10.5,10X,' ITERE2 = ', I10 /
     &         10X,' AMUE2.3= ',F10.5,10X,' NWNDE2 = ', I10 /
     &         10X,' AMUE2.4= ',F10.5                       //)
C
C     CALCULATE THE NEEDED COMBINATIONS OF GAMMA
C
        GM01E2=GM00E2*(GM00E2-1.0)
        GM02E2=   0.5*(GM00E2-3.0)
        GM03E2=       GM00E2-1.0
        GM04E2=   0.5*(GM00E2-1.0)
        GM05E2=       GM00E2-3.0
        GM06E2=   1.5*(GM00E2-1.0)
        GM07E2=   0.5*(GM00E2+1.0)
        GM08E2=       GM00E2+1.0
        GM09E2=(GM00E2*2.0)/(GM00E2+1.0)
        GM10E2=(GM00E2-1.0)/(GM00E2+1.0)
        GM11E2=GM00E2/(1.0-GM00E2)
C
        IHSTE2=JFIL12
C
C     INITIALIZE THE TIMES
C
        CALL E2TIME(0)
C
        RETURN
        END
```

E2INTP

```
        SUBROUTINE E2INTP(IMGL
     &
     &                   )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE INTERPOLATES THE CALCULATED CHANGES TO
C        THE FINEST MESH BY A BI-LINEAR INTERPOLATION
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCGD.INC'
C
C*********************************************************************
C
```

```
C     LOOP THROUGH EVERY MULTIGRID LEVEL BETWEEN CURRENT AND FINEST
C
        DO 30 JMGL=IMGL,MLVLG2-1
C
C     LOOP THROUGH THREE TIMES TO INTERPOLATE FOR...
C     ...1.  ALL INTERIOR SIDES ON THIS LEVEL
C     ...2.  ALL CENTERS ON THIS LEVEL
C     ...3.  ALL INTERFACES SIDES AT THE NEXT FINER LEVEL
C
        DO 20 ITIME=1,5,2
C
C     INTERPOLATE FOR ALL INTERIOR SIDES ON THIS LEVEL
C
        DO 10 ITRIP=LITPE2(ITIME,JMGL),LITPE2(ITIME+1,JMGL)
        ILEFT=IITPE2(1,ITRIP)
        ICENT=IITPE2(2,ITRIP)
        IRITE=IITPE2(3,ITRIP)
C
        CHNGE2(1,ICENT)=0.50*(CHNGE2(1,ILEFT)+CHNGE2(1,IRITE))
        CHNGE2(2,ICENT)=0.50*(CHNGE2(2,ILEFT)+CHNGE2(2,IRITE))
        CHNGE2(3,ICENT)=0.50*(CHNGE2(3,ILEFT)+CHNGE2(3,IRITE))
        CHNGE2(4,ICENT)=0.50*(CHNGE2(4,ILEFT)+CHNGE2(4,IRITE))
10      CONTINUE
C
20      CONTINUE
C
C     NEXT FINER MULTIPLE-GRID LEVEL
C
30      CONTINUE
C
        RETURN
        END
```

E2MAIN

```
        SUBROUTINE E2MAIN(
     &
     &                    DU2MAX,CLIFT,CDRAG)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS IS THE MAIN CONTROLLING ROUTINE FOR NI'S TECHNIQUE FOR
C        TWO-DIMENSIONAL EULER EQUATION.  EMBEDDED MESHES CAN
C        BE HANDLED BY USING A NODE/CELL POINTER SYSTEM
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
```

```
C
C*****************************************************************************
C
C     INITIALIZE FOR CONVERGENCE
C
          CALL UTTIME(TBEG)
        CALL E2CONV(0)
          CALL UTTIME(TEND)
          TM01E2=TM01E2+(TEND-TBEG)
C
C     APPLY PRE-SMOOTHING
C
          CALL UTTIME(TBEG)
        CALL E2WGHT
          CALL UTTIME(TEND)
          TM02E2=TM02E2+(TEND-TBEG)
C
          CALL UTTIME(TBEG)
        CALL E2SMTH
          CALL UTTIME(TEND)
          TM03E2=TM03E2+(TEND-TBEG)
C
C     LOOP THROUGH EACH MULTIGRID LEVEL, STARTING AT THE FINEST
C
        DO 10 IMGL=MLVLG2,-MLVLG2,-1
        IF(ILVLG2(1,IMGL).GT.ILVLG2(6,IMGL)) GOTO 10
C
C     IF ON A COARSE LEVEL, TRANSPORT CHANGES FROM FINER LEVEL TO
C        CELL CENTERS
C
          CALL UTTIME(TBEG)
        CALL E2TRAN(IMGL)
          CALL UTTIME(TEND)
          TM04E2=TM04E2+(TEND-TBEG)
C
C     INITIALIZE CHANGES FOR ALL NODES AT THIS LEVEL
C
          CALL UTTIME(TBEG)
        CALL E2ZERO(IMGL)
          CALL UTTIME(TEND)
          TM05E2=TM05E2+(TEND-TBEG)
C
C     CALCULATE CHANGES AND DISTRIBUTE FOR ALL CELLS
C
          CALL UTTIME(TBEG)
        CALL E2DIST(IMGL)
          CALL UTTIME(TEND)
          TM06E2=TM06E2+(TEND-TBEG)
C
C     APPLY BOUNDARY CONDITIONS
C
          CALL UTTIME(TBEG)
        CALL E2BCON(IMGL)
```

```
            CALL UTTIME(TEND)
            TM07E2=TM07E2+(TEND-TBEG)
C
C     INTERPOLATE TO ALL FINER MESHES
C
            CALL UTTIME(TBEG)
          CALL E2INTP(IMGL)
            CALL UTTIME(TEND)
            TM08E2=TM08E2+(TEND-TBEG)
C
C     APPLY BOUNDARY CONDITIONS
C
            CALL UTTIME(TBEG)
          CALL E2BCON(IMGL)
            CALL UTTIME(TEND)
            TM07E2=TM07E2+(TEND-TBEG)
C
C     UPDATE ALL NODES
C
            CALL UTTIME(TBEG)
          CALL E2UPDT(IMGL)
            CALL UTTIME(TEND)
            TM09E2=TM09E2+(TEND-TBEG)
C
C     LOOP BACK FOR NEXT MULTIGRID LEVEL
C
10        CONTINUE
C
            CALL UTTIME(TBEG)
          CALL E2BFIX
            CALL UTTIME(TEND)
            TM10E2=TM10E2+(TEND-TBEG)
C
C     COMPUTE THE CONVERGENCE STATISTICS
C
            CALL UTTIME(TBEG)
          CALL E2CONV(1)
            CALL UTTIME(TEND)
            TM01E2=TM01E2+(TEND-TBEG)
C
          DU2MAX=DU2ME2
          CLIFT =CLFTE2
          CDRAG =CDRGE2
C
          RETURN
          END
```

E2PRNT

```
          SUBROUTINE E2PRNT
```

```
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PRINTS THE SOLUTION ALONG THE SOLID WALLS
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
        CHARACTER*30 TABTIT
C
C*************************************************************************
C
C     WRITE OUT THE HEADER FOR THE TABLE
C
        WRITE(TABTIT,10)
10      FORMAT('2-D EULER SOLUTION')
C
        CALL UTHEAD(TABTIT)
C
        WRITE(JPRINT,20)
20      FORMAT('   INODE      GEOM1       GEOM2       DPEN1    ',
     &                    '    DPEN2       DPEN3       DPEN4    ',
     &                    '  MACH NO.      ANGL        -CP      ',
     &                    '   PT LOSS  '                          /)
C
C     LOOP THROUGH ALL THE SOLID SURFACE NODES
C
        DO 40 IFOR=1,NFORE2
        INODE=IFORE2(2,IFOR)
C
C     COMPUTE THE APPROPRIATE AERODYNAMIC PROPERTIES
C
C     ...VELOCITY
C
        VEL=SQRT(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)/DPENG2(1,INODE)
C
C     ...MACH NUMBER
C
        XMACH=VEL
     &       /SQRT(GMO1E2*(DPENG2(4,INODE)/DPENG2(1,INODE)-0.5*VEL*VEL))
C
C     ...FLOW ANGLE
C
        IF(VEL.LE.0.000001) THEN
           ANGLE=0.00
        ELSE
           ANGLE=ATAN2(DPENG2(3,INODE),DPENG2(2,INODE))
        ENDIF
```

```
C
C     ...TOTAL PRESSURE LOSS
C
        EM2=U2RFE2*U2RFE2+U3RFE2*U3RFE2
        FACT=U4RFE2*U1RFE2-0.5*EM2
        PTREF=(GM03E2*FACT/U1RFE2)/(1.0+EM2/(2.0*GM00E2*FACT))**GM11E2
C
        EM2=DPENG2(2,INODE)**2+DPENG2(3,INODE)**2
        FACT=DPENG2(4,INODE)*DPENG2(1,INODE)-0.5*EM2
        PT=(GM03E2*FACT/DPENG2(1,INODE))
     &    /(1.0+EM2/(2.0*GM00E2*FACT))**GM11E2
        PTLOSS=ABS(PT-PTREF)/PTREF
C
C     ...PRESSURE COEFFICIENT (NEGATIVE)
C
        QREF=0.50*(U2RFE2*U2RFE2+U3RFE2*U3RFE2)/U1RFE2
C
        Q=0.50*(DPENG2(2,INODE)**2+DPENG2(3,INODE)**2)/DPENG2(1,INODE)
        CPRESS=-GM03E2*((DPENG2(4,INODE)-Q)-(U4RFE2-QREF))/QREF
C
C     PRINT THE TABLE
C
        WRITE(JPRINT,30) INODE,GEOMG2(1,INODE),GEOMG2(2,INODE),
     &                         DPENG2(1,INODE),DPENG2(2,INODE),
     &                         DPENG2(3,INODE),DPENG2(4,INODE),
     &                         XMACH          ,ANGLE          ,
     &                         CPRESS         ,PTLOSS
30      FORMAT(1X,I6,2X,10(F10.5,2X))
C
40      CONTINUE
C
C     PRINT A SUMMARY
C
        CALL G2SUMY(JPRINT)
C
C     PRINT THE FORCE COEFFICIENTS
C
        WRITE(JPRINT,50) CLFTE2,CDRGE2
50      FORMAT(' COMPUTED LIFT COEFFICIENT=',F7.4/
     &         ' COMPUTED DRAG COEFFICIENT=',F7.4)
C
        RETURN
        END
```

E2SMT1

```
        SUBROUTINE E2SMT1
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
```

```
C     THIS SUBROUTINE APPLIES POST-SMOOTHING WITH THE SMOOTHING
C        COEFFICIENT DISTRIBUTION GIVEN IN DPENG2(5,I).  THIS ROUTINE
C        ONLY HANDLES THE FINE CELLS NOT NEAR BOUNDARIES AND NOT
C        JUST OUTISDE EMBEDDED REGIONS
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
C*************************************************************************
C
C     STEP THROUGH CELL WHICH IS NOT NEAR A BOUNDARY AND IS NOT JUST
C        OUTSIDE AN EMBEDDED DOMAIN TO ACCUMULATE SMOOTHING
C
        DO 20 IMGL=0,NLVLG2
C
        DO 10 ICELL=ILVLG2(1,IMGL),ILVLG2(2,IMGL)
C
C     SET UP THE NODE POINTERS
C
          ISW=ICELG2(2,ICELL)
          ISE=ICELG2(4,ICELL)
          INE=ICELG2(6,ICELL)
          INW=ICELG2(8,ICELL)
C
C     SAVE DEPENDENT VARIABLES AND SMOOTHING WEIGHTING FACTOR
C        AT ALL CELL CORNERS
C
          U1SW=DPENG2(1,ISW)
          U2SW=DPENG2(2,ISW)
          U3SW=DPENG2(3,ISW)
          U4SW=DPENG2(4,ISW)
          WTSW=DPENG2(5,ISW)
C
          U1SE=DPENG2(1,ISE)
          U2SE=DPENG2(2,ISE)
          U3SE=DPENG2(3,ISE)
          U4SE=DPENG2(4,ISE)
          WTSE=DPENG2(5,ISE)
C
          U1NE=DPENG2(1,INE)
          U2NE=DPENG2(2,INE)
          U3NE=DPENG2(3,INE)
          U4NE=DPENG2(4,INE)
          WTNE=DPENG2(5,INE)
C
          U1NW=DPENG2(1,INW)
          U2NW=DPENG2(2,INW)
          U3NW=DPENG2(3,INW)
          U4NW=DPENG2(4,INW)
          WTNW=DPENG2(5,INW)
```

```
C
C     COMPUTE THE WEIGHTING FUNCTIONS
C
        WSWSE=MAX(WTSW,WTSE)
        WSENE=MAX(WTSE,WTNE)
        WNENW=MAX(WTNE,WTNW)
        WNWSW=MAX(WTNW,WTSW)
        WSWNE=MAX(WTSW,WTNE)
        WSENW=MAX(WTSE,WTNW)
C
C     SOUTHWEST SMOOTHING
C
        CHNG1=0.125*((U1SE-U1SW)*WSWSE+(U1NE-U1SW)*WSWNE
     &                                +(U1NW-U1SW)*WNWSW)
        CHNG2=0.125*((U2SE-U2SW)*WSWSE+(U2NE-U2SW)*WSWNE
     &                                +(U2NW-U2SW)*WNWSW)
        CHNG3=0.125*((U3SE-U3SW)*WSWSE+(U3NE-U3SW)*WSWNE
     &                                +(U3NW-U3SW)*WNWSW)
        CHNG4=0.125*((U4SE-U4SW)*WSWSE+(U4NE-U4SW)*WSWNE
     &                                +(U4NW-U4SW)*WNWSW)
C
        CHNGE2(1,ISW)=CHNGE2(1,ISW)+CHNG1
        CHNGE2(2,ISW)=CHNGE2(2,ISW)+CHNG2
        CHNGE2(3,ISW)=CHNGE2(3,ISW)+CHNG3
        CHNGE2(4,ISW)=CHNGE2(4,ISW)+CHNG4
C
C     SOUTHEAST SMOOTHING
C
        CHNG1=0.125*((U1NE-U1SE)*WSENE+(U1NW-U1SE)*WSENW
     &                                +(U1SW-U1SE)*WSWSE)
        CHNG2=0.125*((U2NE-U2SE)*WSENE+(U2NW-U2SE)*WSENW
     &                                +(U2SW-U2SE)*WSWSE)
        CHNG3=0.125*((U3NE-U3SE)*WSENE+(U3NW-U3SE)*WSENW
     &                                +(U3SW-U3SE)*WSWSE)
        CHNG4=0.125*((U4NE-U4SE)*WSENE+(U4NW-U4SE)*WSENW
     &                                +(U4SW-U4SE)*WSWSE)
C
        CHNGE2(1,ISE)=CHNGE2(1,ISE)+CHNG1
        CHNGE2(2,ISE)=CHNGE2(2,ISE)+CHNG2
        CHNGE2(3,ISE)=CHNGE2(3,ISE)+CHNG3
        CHNGE2(4,ISE)=CHNGE2(4,ISE)+CHNG4
C
C     NORTHEAST SMOOTHING
C
        CHNG1=0.125*((U1NW-U1NE)*WNENW+(U1SW-U1NE)*WSWNE
     &                                +(U1SE-U1NE)*WSENE)
        CHNG2=0.125*((U2NW-U2NE)*WNENW+(U2SW-U2NE)*WSWNE
     &                                +(U2SE-U2NE)*WSENE)
        CHNG3=0.125*((U3NW-U3NE)*WNENW+(U3SW-U3NE)*WSWNE
     &                                +(U3SE-U3NE)*WSENE)
        CHNG4=0.125*((U4NW-U4NE)*WNENW+(U4SW-U4NE)*WSWNE
     &                                +(U4SE-U4NE)*WSENE)
C
```

```
        CHNGE2(1,INE)=CHNGE2(1,INE)+CHNG1
        CHNGE2(2,INE)=CHNGE2(2,INE)+CHNG2
        CHNGE2(3,INE)=CHNGE2(3,INE)+CHNG3
        CHNGE2(4,INE)=CHNGE2(4,INE)+CHNG4
C
C     NORTHWEST SMOOTHING
C
        CHNG1=0.1250*((U1SW-U1NW)*WNWSW+(U1SE-U1NW)*WSENW
     &                                  +(U1NE-U1NW)*WNENW)
        CHNG2=0.1250*((U2SW-U2NW)*WNWSW+(U2SE-U2NW)*WSENW
     &                                  +(U2NE-U2NW)*WNENW)
        CHNG3=0.1250*((U3SW-U3NW)*WNWSW+(U3SE-U3NW)*WSENW
     &                                  +(U3NE-U3NW)*WNENW)
        CHNG4=0.1250*((U4SW-U4NW)*WNWSW+(U4SE-U4NW)*WSENW
     &                                  +(U4NE-U4NW)*WNENW)
C
        CHNGE2(1,INW)=CHNGE2(1,INW)+CHNG1
        CHNGE2(2,INW)=CHNGE2(2,INW)+CHNG2
        CHNGE2(3,INW)=CHNGE2(3,INW)+CHNG3
        CHNGE2(4,INW)=CHNGE2(4,INW)+CHNG4
C
C     GO BACK FOR NEXT CELL
C
10      CONTINUE
C
20      CONTINUE
C
        RETURN
        END
```

E2SMT2

```
        SUBROUTINE E2SMT2
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE APPLIES POST-SMOOTHING WITH THE SMOOTHING
C        COEFFICIENT DISTRIBUTION GIVEN IN DPENG2(5,I).  THIS ROUTINE
C        ONLY HANDLES CELLS ALONG BOUNDARIES AND JUST OUTSIDE
C        EMBEDDED REGIONS
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************************
```

```
C
C     STATEMENT FUNCTION FOR WEIGHTING OF SMOOTHING COEFFICIENTS
C
        W(X,Y)=MAX(X,Y)
C
C     STEP THROUGH CELL WHICH IS NEAR A BOUNDARY OR IS JUST
C        OUTSIDE AN EMBEDDED DOMAIN TO ACCUMULATE SMOOTHING
C
        DO 10 IMGL=0,MLVLG2
        DO 10 ICELL=ILVLG2(3,IMGL),ILVLG2(4,IMGL)
C
C     SET UP NODE POINTERS FOR THIS CELL
C
        ICENT=ICELG2(1,ICELL)
        ISW  =ICELG2(2,ICELL)
        IS   =ICELG2(3,ICELL)
        ISE  =ICELG2(4,ICELL)
        IE   =ICELG2(5,ICELL)
        INE  =ICELG2(6,ICELL)
        IN   =ICELG2(7,ICELL)
        INW  =ICELG2(8,ICELL)
        IW   =ICELG2(9,ICELL)
C
        IAUX =ICELG2(10,ICELL)
C
C     SAVE DEPENDENT VARIABLES AND SMOOTHING WEIGHTING FACTOR
C        AT ALL CELL CORNERS
C
        U1SW=DPENG2(1,ISW)
        U2SW=DPENG2(2,ISW)
        U3SW=DPENG2(3,ISW)
        U4SW=DPENG2(4,ISW)
        WSW =DPENG2(5,ISW)
C
        U1SE=DPENG2(1,ISE)
        U2SE=DPENG2(2,ISE)
        U3SE=DPENG2(3,ISE)
        U4SE=DPENG2(4,ISE)
        WSE =DPENG2(5,ISE)
C
        U1NE=DPENG2(1,INE)
        U2NE=DPENG2(2,INE)
        U3NE=DPENG2(3,INE)
        U4NE=DPENG2(4,INE)
        WNE =DPENG2(5,INE)
C
        U1NW=DPENG2(1,INW)
        U2NW=DPENG2(2,INW)
        U3NW=DPENG2(3,INW)
        U4NW=DPENG2(4,INW)
        WNW =DPENG2(5,INW)
C
        IF(IS.EQ.0) THEN
```

```
            U1S=0.5*(U1SW+U1SE)
            U2S=0.5*(U2SW+U2SE)
            U3S=0.5*(U3SW+U3SE)
            U4S=0.5*(U4SW+U4SE)
            WS =0.5*( WSW+ WSE)
         ELSE
            U1S=DPENG2(1,IS)
            U2S=DPENG2(2,IS)
            U3S=DPENG2(3,IS)
            U4S=DPENG2(4,IS)
            WS =DPENG2(5,IS)
         ENDIF
C
         IF(IE.EQ.0) THEN
            U1E=0.5*(U1SE+U1NE)
            U2E=0.5*(U2SE+U2NE)
            U3E=0.5*(U3SE+U3NE)
            U4E=0.5*(U4SE+U4NE)
            WE =0.5*( WSE+ WNE)
         ELSE
            U1E=DPENG2(1,IE)
            U2E=DPENG2(2,IE)
            U3E=DPENG2(3,IE)
            U4E=DPENG2(4,IE)
            WE =DPENG2(5,IE)
         ENDIF
C
         IF(IN.EQ.0) THEN
            U1N=0.5*(U1NE+U1NW)
            U2N=0.5*(U2NE+U2NW)
            U3N=0.5*(U3NE+U3NW)
            U4N=0.5*(U4NE+U4NW)
            WN =0.5*( WNE+ WNW)
         ELSE
            U1N=DPENG2(1,IN)
            U2N=DPENG2(2,IN)
            U3N=DPENG2(3,IN)
            U4N=DPENG2(4,IN)
            WN =DPENG2(5,IN)
         ENDIF
C
         IF(IW.EQ.0) THEN
            U1W=0.5*(U1SW+U1NW)
            U2W=0.5*(U2SW+U2NW)
            U3W=0.5*(U3SW+U3NW)
            U4W=0.5*(U4SW+U4NW)
            WW =0.5*( WSW+ WNW)
         ELSE
            U1W=DPENG2(1,IW)
            U2W=DPENG2(2,IW)
            U3W=DPENG2(3,IW)
            U4W=DPENG2(4,IW)
            WW =DPENG2(5,IW)
```

```
        ENDIF
C
C     SET UP DEPENDENT VARIABLES AT THE CELL CENTER
C
        IF(ICENT.EQ.0) THEN
           U1C=0.25*(U1SW+U1SE+U1NE+U1NW)
           U2C=0.25*(U2SW+U2SE+U2NE+U2NW)
           U3C=0.25*(U3SW+U3SE+U3NE+U3NW)
           U4C=0.25*(U4SW+U4SE+U4NE+U4NW)
           WC =0.25*( WSW+ WSE+ WNE+ WNW)
        ELSE
           U1C=DPENG2(1,ICENT)
           U2C=DPENG2(2,ICENT)
           U3C=DPENG2(3,ICENT)
           U4C=DPENG2(4,ICENT)
           WC =DPENG2(5,ICENT)
        ENDIF
C
C     SOUTHWEST SMOOTHING
C
        IF(IAND(IAUX,HL0001).EQ.HL0000) THEN
           IF(IAND(IAUX,HL0010).EQ.HL0000) THEN
C
C             ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
              CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.1250*((U1SE-U1SW)*W(WSE,WSW)
     &                                           +(U1NE-U1SW)*W(WNE,WSW)
     &                                           +(U1NW-U1SW)*W(WNW,WSW))
              CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.1250*((U2SE-U2SW)*W(WSE,WSW)
     &                                           +(U2NE-U2SW)*W(WNE,WSW)
     &                                           +(U2NW-U2SW)*W(WNW,WSW))
              CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.1250*((U3SE-U3SW)*W(WSE,WSW)
     &                                           +(U3NE-U3SW)*W(WNE,WSW)
     &                                           +(U3NW-U3SW)*W(WNW,WSW))
              CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.1250*((U4SE-U4SW)*W(WSE,WSW)
     &                                           +(U4NE-U4SW)*W(WNE,WSW)
     &                                           +(U4NW-U4SW)*W(WNW,WSW))
C
           ELSE
C
C             ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
              CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.1250*((U1S -U1SW)*W(WS ,WSW)
     &                                           +(U1C -U1SW)*W(WC ,WSW)
     &                                           +(U1W -U1SW)*W(WW ,WSW))
              CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.1250*((U2S -U2SW)*W(WS ,WSW)
     &                                           +(U2C -U2SW)*W(WC ,WSW)
     &                                           +(U2W -U2SW)*W(WW ,WSW))
              CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.1250*((U3S -U3SW)*W(WS ,WSW)
     &                                           +(U3C -U3SW)*W(WC ,WSW)
     &                                           +(U3W -U3SW)*W(WW ,WSW))
              CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.1250*((U4S -U4SW)*W(WS ,WSW)
     &                                           +(U4C -U4SW)*W(WC ,WSW)
     &                                           +(U4W -U4SW)*W(WW ,WSW))
C
```

```
         ENDIF
       ELSEIF(IAND(IAUX,HL0008).EQ.HL0008) THEN
C
C        ---CORNER BOUNDARY CELL (SOUTHWEST CORNER OF DOMAIN)
         CONTINUE
C
       ELSEIF(IAND(IAUX,HL0009).EQ.HL0009) THEN
         IF(IAND(IAUX,HL0010).EQ.HL0000) THEN
C
C        ---BOUNDARY ON WESTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
           CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.50*((U1NW-U1SW)*W(WNW,WSW))
           CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.50*((U2NW-U2SW)*W(WNW,WSW))
           CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.50*((U3NW-U3SW)*W(WNW,WSW))
           CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.50*((U4NW-U4SW)*W(WNW,WSW))
C
         ELSE
C
C        ---BOUNDARY ON WESTERN EDGE, JUST OUTSIDE EMBEDDED REGION
           CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.50*((U1W -U1SW)*W(WW ,WSW))
           CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.50*((U2W -U2SW)*W(WW ,WSW))
           CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.50*((U3W -U3SW)*W(WW ,WSW))
           CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.50*((U4W -U4SW)*W(WW ,WSW))
C
         ENDIF
       ELSEIF(IAND(IAUX,HL0003).EQ.HL0003) THEN
         IF(IAND(IAUX,HL0010).EQ.HL0000) THEN
C
C        ---BOUNDARY ON SOUTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
           CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.50*((U1SE-U1SW)*W(WSE,WSW))
           CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.50*((U2SE-U2SW)*W(WSE,WSW))
           CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.50*((U3SE-U3SW)*W(WSE,WSW))
           CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.50*((U4SE-U4SW)*W(WSE,WSW))
C
         ELSE
C
C        ---BOUNDARY ON SOUTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
           CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.50*((U1S -U1SW)*W(WS ,WSW))
           CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.50*((U2S -U2SW)*W(WS ,WSW))
           CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.50*((U3S -U3SW)*W(WS ,WSW))
           CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.50*((U4S -U4SW)*W(WS ,WSW))
C
         ENDIF
C
       ELSEIF(IAND(IAUX,HL0001).EQ.HL0001) THEN
C        ---INSIDE CORNER BOUNDARY
           CHNGE2(1,ISW)=CHNGE2(1,ISW)+0.50*((U1SE-U1SW)*W(WSE,WSW)
     &                                    +(U1NW-U1SW)*W(WNW,WSW))
           CHNGE2(2,ISW)=CHNGE2(2,ISW)+0.50*((U2SE-U2SW)*W(WSE,WSW)
     &                                    +(U2NW-U2SW)*W(WNW,WSW))
           CHNGE2(3,ISW)=CHNGE2(3,ISW)+0.50*((U3SE-U3SW)*W(WSE,WSW)
     &                                    +(U3NW-U3SW)*W(WNW,WSW))
           CHNGE2(4,ISW)=CHNGE2(4,ISW)+0.50*((U4SE-U4SW)*W(WSE,WSW)
     &                                    +(U4NW-U4SW)*W(WNW,WSW))
```

```
C
        ELSE
           CALL UTEROR(+1,REAL(ICELL),REAL(IAUX))
        ENDIF
C
C     SOUTH SMOOTHING
C
        IF(IS.NE.0) THEN
          CHNGE2(1,IS)=CHNGE2(1,IS)+0.1250*(    (U1SE-U1S )*W(WSE,WS )
     &                                      +    (U1E -U1S )*W(WE ,WS )
     &                                     +2.0*(U1C -U1S )*W(WC ,WS )
     &                                      +    (U1W -U1S )*W(WW ,WS )
     &                                      +    (U1SW-U1S )*W(WSW,WS ))
          CHNGE2(2,IS)=CHNGE2(2,IS)+0.1250*(    (U2SE-U2S )*W(WSE,WS )
     &                                      +    (U2E -U2S )*W(WE ,WS )
     &                                     +2.0*(U2C -U2S )*W(WC ,WS )
     &                                      +    (U2W -U2S )*W(WW ,WS )
     &                                      +    (U2SW-U2S )*W(WSW,WS ))
          CHNGE2(3,IS)=CHNGE2(3,IS)+0.1250*(    (U3SE-U3S )*W(WSE,WS )
     &                                      +    (U3E -U3S )*W(WE ,WS )
     &                                     +2.0*(U3C -U3S )*W(WC ,WS )
     &                                      +    (U3W -U3S )*W(WW ,WS )
     &                                      +    (U3SW-U3S )*W(WSW,WS ))
          CHNGE2(4,IS)=CHNGE2(4,IS)+0.1250*(    (U4SE-U4S )*W(WSE,WS )
     &                                      +    (U4E -U4S )*W(WE ,WS )
     &                                     +2.0*(U4C -U4S )*W(WC ,WS )
     &                                      +    (U4W -U4S )*W(WW ,WS )
     &                                      +    (U4SW-U4S )*W(WSW,WS ))
        ENDIF
C
C     SOUTHEAST SMOOTHING
C
        IF(IAND(IAUX,HL0002).EQ.HL0000) THEN
           IF(IAND(IAUX,HL0020).EQ.HL0000) THEN
C
C            ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
             CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.1250*((U1NE-U1SE)*W(WNE,WSE)
     &                                         +(U1NW-U1SE)*W(WNW,WSE)
     &                                         +(U1SW-U1SE)*W(WSW,WSE))
             CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.1250*((U2NE-U2SE)*W(WNE,WSE)
     &                                         +(U2NW-U2SE)*W(WNW,WSE)
     &                                         +(U2SW-U2SE)*W(WSW,WSE))
             CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.1250*((U3NE-U3SE)*W(WNE,WSE)
     &                                         +(U3NW-U3SE)*W(WNW,WSE)
     &                                         +(U3SW-U3SE)*W(WSW,WSE))
             CHNGE2(4,ISE)=CHNGE2(4,ISE)+0.1250*((U4NE-U4SE)*W(WNE,WSE)
     &                                         +(U4NW-U4SE)*W(WNW,WSE)
     &                                         +(U4SW-U4SE)*W(WSW,WSE))
C
           ELSE
C
C            ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
             CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.1250*((U1E -U1SE)*W(WE ,WSE)
```

```
     &                                          +(U1C -U1SE)*W(WC ,WSE)
     &                                          +(U1S -U1SE)*W(WS ,WSE))
           CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.1250*((U2E -U2SE)*W(WE ,WSE)
     &                                          +(U2C -U2SE)*W(WC ,WSE)
     &                                          +(U2S -U2SE)*W(WS ,WSE))
           CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.1250*((U3E -U3SE)*W(WE ,WSE)
     &                                          +(U3C -U3SE)*W(WC ,WSE)
     &                                          +(U3S -U3SF)*W(WS ,WSE))
           CHNGE2(4,ISE)=CHNGE2(4,ISE)+0.1250*((U4E -U4SE)*W(WE ,WSE)
     &                                          +(U4C -U4SE)*W(WC ,WSE)
     &                                          +(U4S -U4SE)*W(WS ,WSE))
C
         ENDIF
       ELSEIF(IAND(IAUX,HL0007).EQ.HL0007) THEN
C
C        ---CORNER BOUNDARY CELL (SOUTHEAST CORNER OF DOMAIN)
         CONTINUE
C
       ELSEIF(IAND(IAUX,HL0003).EQ.HL0003) THEN
         IF(IAND(IAUX,HL0020).EQ.HL0000) THEN
C
C        ---BOUNDARY ON SOUTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
           CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.50*((U1SW-U1SE)*W(WSW,WSE))
           CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.50*((U2SW-U2SE)*W(WSW,WSE))
           CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.50*((U3SW-U3SE)*W(WSW,WSE))
           CHNGE2(4,ISE)=CHNGE2(4,ISE)+0.50*((U4SW-U4SE)*W(WSW,WSE))
C
         ELSE
C
C        ---BOUNDARY ON SOUTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
           CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.50*((U1S -U1SE)*W(WS ,WSE))
           CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.50*((U2S -U2SE)*W(WS ,WSE))
           CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.50*((U3S -U3SE)*W(WS ,WSE))
           CHNGE2(4,ISE)=CHNGE2(4,ISE)+0.50*((U4S -U4SE)*W(WS ,WSE))
C
         ENDIF
       ELSEIF(IAND(IAUX,HL0006).EQ.HL0006) THEN
         IF(IAND(IAUX,HL0020).EQ.HL0000) THEN
C
C        ---BOUNDARY ON EASTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
           CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.50*((U1NE-U1SE)*W(WNE,WSE))
           CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.50*((U2NE-U2SE)*W(WNE,WSE))
           CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.50*((U3NE-U3SE)*W(WNE,WSE))
           CHNGE2(4,ISE)=CHNGE2(4,ISE)+0.50*((U4NE-U4SE)*W(WNE,WSE))
C
         ELSE
C
C        ---BOUNDARY ON EASTERN EDGE, JUST OUTSIDE EMBEDDED REGION
           CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.50*((U1E -U1SE)*W(WE ,WSE))
           CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.50*((U2E -U2SE)*W(WE ,WSE))
           CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.50*((U3E -U3SE)*W(WE ,WSE))
           CHNGE2(4,ISE)=CHNGE2(4,ICE)+0.50*((U4E -U4SE)*W(WE ,WSE))
C
```

```
            ENDIF
C
          ELSEIF(IAND(IAUX,HL0002).EQ.HL0002) THEN
C           ---INSIDE CORNER BOUNDARY
              CHNGE2(1,ISE)=CHNGE2(1,ISE)+0.50*((U1SW-U1SE)*W(WSW,WSE)
     &                                         +(U1NE-U1SE)*W(WNE,WSE))
              CHNGE2(2,ISE)=CHNGE2(2,ISE)+0.50*((U2SW-U2SE)*W(WSW,WSE)
     &                                         +(U2NE-U2SE)*W(WNE,WSE))
              CHNGE2(3,ISE)=CHNGE2(3,ISE)+0.50*((U3SW-U3SE)*W(WSW,WSE)
     &                                         +(U3NE-U3SE)*W(WNE,WSE))
              CHNGE2(4,ISE)=CHNGE2(4,ISE)+0.50*((U4SW-U4SE)*W(WSW,WSE)
     &                                         +(U4NE-U4SE)*W(WNE,WSE))
C
            ELSE
              CALL UTEROR(+2,REAL(ICELL),REAL(IAUX))
            ENDIF
C
C     EAST SMOOTHING
C
            IF(IE.NE.0) THEN
              CHNGE2(1,IE)=CHNGE2(1,IE)+0.1250*(    (U1NE-U1E )*W(WNE,WE )
     &                                         +    (U1N -U1E )*W(WN ,WE )
     &                                        +2.0*(U1C -U1E )*W(WC ,WE )
     &                                         +    (U1S -U1E )*W(WS ,WE )
     &                                         +    (U1SE-U1E )*W(WSE,WE ))
              CHNGE2(2,IE)=CHNGE2(2,IE)+0.1250*(    (U2NE-U2E )*W(WNE,WE )
     &                                         +    (U2N -U2E )*W(WN ,WE )
     &                                        +2.0*(U2C -U2E )*W(WC ,WE )
     &                                         +    (U2S -U2E )*W(WS ,WE )
     &                                         +    (U2SE-U2E )*W(WSE,WE ))
              CHNGE2(3,IE)=CHNGE2(3,IE)+0.1250*(    (U3NE-U3E )*W(WNE,WE )
     &                                         +    (U3N -U3E )*W(WN ,WE )
     &                                        +2.0*(U3C -U3E )*W(WC ,WE )
     &                                         +    (U3S -U3E )*W(WS ,WE )
     &                                         +    (U3SE-U3E )*W(WSE,WE ))
              CHNGE2(4,IE)=CHNGE2(4,IE)+0.1250*(    (U4NE-U4E )*W(WNE,WE )
     &                                         +    (U4N -U4E )*W(WN ,WE )
     &                                        +2.0*(U4C -U4E )*W(WC ,WE )
     &                                         +    (U4S -U4E )*W(WS ,WE )
     &                                         +    (U4SE-U4E )*W(WSE,WE ))
            ENDIF
C
C     NORTHEAST SMOOTHING
C
            IF(IAND(IAUX,HL0004).EQ.HL0000) THEN
              IF(IAND(IAUX,HL0040).EQ.HL0000) THEN
C
C             ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
                CHNGE2(1,INE)=CHNGE2(1,INE)+0.1250*((U1NW-U1NE)*W(WNW,WNE)
     &                                             +(U1SW-U1NE)*W(WSW,WNE)
     &                                             +(U1SE-U1NE)*W(WSE,WNE))
                CHNGE2(2,INE)=CHNGE2(2,INE)+0.1250*((U2NW-U2NE)*W(WNW,WNE)
     &                                             +(U2SW-U2NE)*W(WSW,WNE)
```

```
     &                                          +(U2SE-U2NE)*W(WSE,WNE))
            CHNGE2(3,INE)=CHNGE2(3,INE)+0.1250*((U3NW-U3NE)*W(WNW,WNE)
     &                                          +(U3SW-U3NE)*W(WSW,WNE)
     &                                          +(U3SE-U3NE)*W(WSE,WNE))
            CHNGE2(4,INE)=CHNGE2(4,INE)+0.1250*((U4NW-U4NE)*W(WNW,WNE)
     &                                          +(U4SW-U4NE)*W(WSW,WNE)
     &                                          +(U4SE-U4NE)*W(WSE,WNE))
C
         ELSE
C
C        ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
            CHNGE2(1,INE)=CHNGE2(1,INE)+0.1250*((U1N -U1NE)*W(WN ,WNE)
     &                                          +(U1C -U1NE)*W(WC ,WNE)
     &                                          +(U1E -U1NE)*W(WE ,WNE))
            CHNGE2(2,INE)=CHNGE2(2,INE)+0.1250*((U2N -U2NE)*W(WN ,WNE)
     &                                          +(U2C -U2NE)*W(WC ,WNE)
     &                                          +(U2E -U2NE)*W(WE ,WNE))
            CHNGE2(3,INE)=CHNGE2(3,INE)+0.1250*((U3N -U3NE)*W(WN ,WNE)
     &                                          +(U3C -U3NE)*W(WC ,WNE)
     &                                          +(U3E -U3NE)*W(WE ,WNE))
            CHNGE2(4,INE)=CHNGE2(4,INE)+0.1250*((U4N -U4NE)*W(WN ,WNE)
     &                                          +(U4C -U4NE)*W(WC ,WNE)
     &                                          +(U4E -U4NE)*W(WE ,WNE))
C
         ENDIF
      ELSEIF(IAND(IAUX,HL000E).EQ.HL000E) THEN
C
C        ---CORNER BOUNDARY CELL (NORTHEAST CORNER OF DOMAIN)
            CONTINUE
C
      ELSEIF(IAND(IAUX,HL0006).EQ.HL0006) THEN
         IF(IAND(IAUX,HL0040).EQ.HL0000) THEN
C
C        ---BOUNDARY ON EASTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
            CHNGE2(1,INE)=CHNGE2(1,INE)+0.50*((U1SE-U1NE)*W(WSE,WNE))
            CHNGE2(2,INE)=CHNGE2(2,INE)+0.50*((U2SE-U2NE)*W(WSE,WNE))
            CHNGE2(3,INE)=CHNGE2(3,INE)+0.50*((U3SE-U3NE)*W(WSE,WNE))
            CHNGE2(4,INE)=CHNGE2(4,INE)+0.50*((U4SE-U4NE)*W(WSE,WNE))
C
         ELSE
C
C        ---BOUNDARY ON EASTERN EDGE, JUST OUTSIDE EMBEDDED REGION
            CHNGE2(1,INE)=CHNGE2(1,INE)+0.50*((U1E -U1NE)*W(WE ,WNE))
            CHNGE2(2,INE)=CHNGE2(2,INE)+0.50*((U2E -U2NE)*W(WE ,WNE))
            CHNGE2(3,INE)=CHNGE2(3,INE)+0.50*((U3E -U3NE)*W(WE ,WNE))
            CHNGE2(4,INE)=CHNGE2(4,INE)+0.50*((U4E -U4NE)*W(WE ,WNE))
C
         ENDIF
      ELSEIF(IAND(IAUX,HL000C).EQ.HL000C) THEN
         IF(IAND(IAUX,HL0040).EQ.HL0000) THEN
C
C        ---BOUNDARY ON NORTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
            CHNGE2(1,INE)=CHNGE2(1,INE)+0.50*((U1NW-U1NE)*W(WNW,WNE))
```

```
            CHNGE2(2,INE)=CHNGE2(2,INE)+0.50*((U2NW-U2NE)*W(WNW,WNE))
            CHNGE2(3,INE)=CHNGE2(3,INE)+0.50*((U3NW-U3NE)*W(WNW,WNE))
            CHNGE2(4,INE)=CHNGE2(4,INE)+0.50*((U4NW-U4NE)*W(WNW,WNE))
C
          ELSE
C
C         ---BOUNDARY ON NORTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
            CHNGE2(1,INE)=CHNGE2(1,INE)+0.50*((U1N -U1NE)*W(WN ,WNE))
            CHNGE2(2,INE)=CHNGE2(2,INE)+0.50*((U2N -U2NE)*W(WN ,WNE))
            CHNGE2(3,INE)=CHNGE2(3,INE)+0.50*((U3N -U3NE)*W(WN ,WNE))
            CHNGE2(4,INE)=CHNGE2(4,INE)+0.50*((U4N -U4NE)*W(WN ,WNE))
C
          ENDIF
C
        ELSEIF(IAND(IAUX,HL0004).EQ.HL0004) THEN
C         ---INSIDE CORNER BOUNDARY
            CHNGE2(1,INE)=CHNGE2(1,INE)+0.50*((U1SE-U1NE)*W(WSE,WNE)
     &                                    +(U1NW-U1NE)*W(WNW,WNE))
            CHNGE2(2,INE)=CHNGE2(2,INE)+0.50*((U2SE-U2NE)*W(WSE,WNE)
     &                                    +(U2NW-U2NE)*W(WNW,WNE))
            CHNGE2(3,INE)=CHNGE2(3,INE)+0.50*((U3SE-U3NE)*W(WSE,WNE)
     &                                    +(U3NW-U3NE)*W(WNW,WNE))
            CHNGE2(4,INE)=CHNGE2(4,INE)+0.50*((U4SE-U4NE)*W(WSE,WNE)
     &                                    +(U4NW-U4NE)*W(WNW,WNE))
C
        ELSE
          CALL UTEROR(+3,REAL(ICELL),REAL(IAUX))
        ENDIF
C
C     NORTH SMOOTHING
C
        IF(IN.NE.0) THEN
          CHNGE2(1,IN)=CHNGE2(1,IN)+0.1250*(    (U1NW-U1N )*W(WNW,WN )
     &                                     +    (U1W -U1N )*W(WW ,WN )
     &                                    +2.0*(U1C -U1N )*W(WC ,WN )
     &                                     +    (U1E -U1N )*W(WE ,WN )
     &                                     +    (U1NE-U1N )*W(WNE,WN ))
          CHNGE2(2,IN)=CHNGE2(2,IN)+0.1250*(    (U2NW-U2N )*W(WNW,WN )
     &                                     +    (U2W -U2N )*W(WW ,WN )
     &                                    +2.0*(U2C -U2N )*W(WC ,WN )
     &                                     +    (U2E -U2N )*W(WE ,WN )
     &                                     +    (U2NE-U2N )*W(WNE,WN ))
          CHNGE2(3,IN)=CHNGE2(3,IN)+0.1250*(    (U3NW-U3N )*W(WNW,WN )
     &                                     +    (U3W -U3N )*W(WW ,WN )
     &                                    +2.0*(U3C -U3N )*W(WC ,WN )
     &                                     +    (U3E -U3N )*W(WE ,WN )
     &                                     +    (U3NE-U3N )*W(WNE,WN ))
          CHNGE2(4,IN)=CHNGE2(4,IN)+0.1250*(    (U4NW-U4N )*W(WNW,WN )
     &                                     +    (U4W -U4N )*W(WW ,WN )
     &                                    +2.0*(U4C -U4N )*W(WC ,WN )
     &                                     +    (U4E -U4N )*W(WE ,WN )
     &                                     +    (U4NE-U4N )*W(WNE,WN ))
        ENDIF
```

```
C
C     NORTHWEST SMOOTHING
C
        IF(IAND(IAUX,HL0008).EQ.HL0000) THEN
           IF(IAND(IAUX,HL0080).EQ.HL0000) THEN
C
C          ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
             CHNGE2(1,INW)=CHNGE2(1,INW)+0.1250*((U1SW-U1NW)*W(WSW,WNW)
     &                                         +(U1SE-U1NW)*W(WSE,WNW)
     &                                         +(U1NE-U1NW)*W(WNE,WNW))
             CHNGE2(2,INW)=CHNGE2(2,INW)+0.1250*((U2SW-U2NW)*W(WSW,WNW)
     &                                         +(U2SE-U2NW)*W(WSE,WNW)
     &                                         +(U2NE-U2NW)*W(WNE,WNW))
             CHNGE2(3,INW)=CHNGE2(3,INW)+0.1250*((U3SW-U3NW)*W(WSW,WNW)
     &                                         +(U3SE-U3NW)*W(WSE,WNW)
     &                                         +(U3NE-U3NW)*W(WNE,WNW))
             CHNGE2(4,INW)=CHNGE2(4,INW)+0.1250*((U4SW-U4NW)*W(WSW,WNW)
     &                                         +(U4SE-U4NW)*W(WSE,WNW)
     &                                         +(U4NE-U4NW)*W(WNE,WNW))
C
           ELSE
C
C          ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
             CHNGE2(1,INW)=CHNGE2(1,INW)+0.1250*((U1W -U1NW)*W(WW ,WNW)
     &                                         +(U1C -U1NW)*W(WC ,WNW)
     &                                         +(U1N -U1NW)*W(WN ,WNW))
             CHNGE2(2,INW)=CHNGE2(2,INW)+0.1250*((U2W -U2NW)*W(WW ,WNW)
     &                                         +(U2C -U2NW)*W(WC ,WNW)
     &                                         +(U2N -U2NW)*W(WN ,WNW))
             CHNGE2(3,INW)=CHNGE2(3,INW)+0.1250*((U3W -U3NW)*W(WW ,WNW)
     &                                         +(U3C -U3NW)*W(WC ,WNW)
     &                                         +(U3N -U3NW)*W(WN ,WNW))
             CHNGE2(4,INW)=CHNGE2(4,INW)+0.1250*((U4W -U4NW)*W(WW ,WNW)
     &                                         +(U4C -U4NW)*W(WC ,WNW)
     &                                         +(U4N -U4NW)*W(WN ,WNW))
C
           ENDIF
        ELSEIF(IAND(IAUX,HL000D).EQ.HL000D) THEN
C
C          ---CORNER BOUNDARY CELL (NORTHWEST CORNER OF DOMAIN)
             CONTINUE
C
        ELSEIF(IAND(IAUX,HL000C).EQ.HL000C) THEN
           IF(IAND(IAUX,HL0080).EQ.HL0000) THEN
C
C          ---BOUNDARY ON NORTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
             CHNGE2(1,INW)=CHNGE2(1,INW)+0.50*((U1NE-U1NW)*W(WNE,WNW))
             CHNGE2(2,INW)=CHNGE2(2,INW)+0.50*((U2NE-U2NW)*W(WNE,WNW))
             CHNGE2(3,INW)=CHNGE2(3,INW)+0.50*((U3NE-U3NW)*W(WNE,WNW))
             CHNGE2(4,INW)=CHNGE2(4,INW)+0.50*((U4NE-U4NW)*W(WNE,WNW))
C
           ELSE
C
```

```
C          ---BOUNDARY ON NORTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
             CHNGE2(1,INW)=CHNGE2(1,INW)+0.50*((U1N -U1NW)*W(WN ,WNW))
             CHNGE2(2,INW)=CHNGE2(2,INW)+0.50*((U2N -U2NW)*W(WN ,WNW))
             CHNGE2(3,INW)=CHNGE2(3,INW)+0.50*((U3N -U3NW)*W(WN ,WNW))
             CHNGE2(4,INW)=CHNGE2(4,INW)+0.50*((U4N -U4NW)*W(WN ,WNW))
C
          ENDIF
        ELSEIF(IAND(IAUX,HL0009).EQ.HL0009) THEN
          IF(IAND(IAUX,HL0080).EQ.HL0000) THEN
C
C          ---BOUNDARY ON WESTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
             CHNGE2(1,INW)=CHNGE2(1,INW)+0.50*((U1SW-U1NW)*W(WSW,WNW))
             CHNGE2(2,INW)=CHNGE2(2,INW)+0.50*((U2SW-U2NW)*W(WSW,WNW))
             CHNGE2(3,INW)=CHNGE2(3,INW)+0.50*((U3SW-U3NW)*W(WSW,WNW))
             CHNGE2(4,INW)=CHNGE2(4,INW)+0.50*((U4SW-U4NW)*W(WSW,WNW))
C
          ELSE
C
C          ---BOUNDARY ON WESTERN EDGE, JUST OUTSIDE EMBEDDED REGION
             CHNGE2(1,INW)=CHNGE2(1,INW)+0.50*((U1W -U1NW)*W(WW ,WNW))
             CHNGE2(2,INW)=CHNGE2(2,INW)+0.50*((U2W -U2NW)*W(WW ,WNW))
             CHNGE2(3,INW)=CHNGE2(3,INW)+0.50*((U3W -U3NW)*W(WW ,WNW))
             CHNGE2(4,INW)=CHNGE2(4,INW)+0.50*((U4W -U4NW)*W(WW ,WNW))
C
          ENDIF
C
        ELSEIF(IAND(IAUX,HL0008).EQ.HL0008) THEN
C          ---INSIDE CORNER BOUNDARY
             CHNGE2(1,INW)=CHNGE2(1,INW)+0.50*((U1SW-U1NW)*W(WSW,WNW)
     &                                     +(U1NE-U1NW)*W(WNE,WNW))
             CHNGE2(2,INW)=CHNGE2(2,INW)+0.50*((U2SW-U2NW)*W(WSW,WNW)
     &                                     +(U2NE-U2NW)*W(WNE,WNW))
             CHNGE2(3,INW)=CHNGE2(3,INW)+0.50*((U3SW-U3NW)*W(WSW,WNW)
     &                                     +(U3NE-U3NW)*W(WNE,WNW))
             CHNGE2(4,INW)=CHNGE2(4,INW)+0.50*((U4SW-U4NW)*W(WSW,WNW)
     &                                     +(U4NE-U4NW)*W(WNE,WNW))
C
        ELSE
          CALL UTEROR(+4,REAL(ICELL),REAL(IAUX))
        ENDIF
C
C     WEST SMOOTHING
C
        IF(IW.NE.0) THEN
          CHNGE2(1,IW)=CHNGE2(1,IW)+0.1250*(    (U1SW-U1W )*W(WSW,WW )
     &                                     +    (U1S -U1W )*W(WS ,WW )
     &                                     +2.0*(U1C -U1W )*W(WC ,WW )
     &                                     +    (U1N -U1W )*W(WN ,WW )
     &                                     +    (U1NW-U1W )*W(WNW,WW ))
          CHNGE2(2,IW)=CHNGE2(2,IW)+0.1250*(    (U2SW-U2W )*W(WSW,WW )
     &                                     +    (U2S -U2W )*W(WS ,WW )
     &                                     +2.0*(U2C -U2W )*W(WC ,WW )
     &                                     +    (U2N -U2W )*W(WN ,WW )
```

```
     &                                      +    (U2NW-U2W )*W(WNW,WW ))
          CHNGE2(3,IW)=CHNGE2(3,IW)+0.1250*(    (U3SW-U3W )*W(WSW,WW )
     &                                      +    (U3S -U3W )*W(WS ,WW )
     &                                     +2.0*(U3C -U3W )*W(WC ,WW )
     &                                      +    (U3N -U3W )*W(WN ,WW )
     &                                      +    (U3NW-U3W )*W(WNW,WW ))
          CHNGE2(4,IW)=CHNGE2(4,IW)+0.1250*(    (U4SW-U4W )*W(WSW,WW )
     &                                      +    (U4S -U4W )*W(WS ,WW )
     &                                     +2.0*(U4C -U4W )*W(WC ,WW )
     &                                      +    (U4N -U4W )*W(WN ,WW )
     &                                      +    (U4NW-U4W )*W(WNW,WW ))
        ENDIF
C
C     GO BACK FOR NEXT CELL
C
10      CONTINUE
C
        RETURN
        END
```

E2SMTH

```
        SUBROUTINE E2SMTH
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE APPLIES POST-SMOOTHING WITH THE SMOOTHING
C        COEFFICIENT DISTRIBUTION GIVEN IN DPENG2(5,I)
C
C**********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
C**********************************************************************
C
C     STEP THROUGH ALL CELLS WHICH ARE NOT NEAR A BOUNDARY AND ARE
C        JUST OUTSIDE AN EMBEDDED DOMAIN TO ACCUMULATE SMOOTHING
C
        CALL E2SMT1
C
C     STEP THROUGH ALL CELLS WHICH ARE NEAR A BOUNDARY OR ARE JUST
C        OUTSIDE AN EMBEDDED DOMAIN TO ACCUMULATE SMOOTHING
C
        CALL E2SMT2
C
        RETURN
        END
```

E2STAT

```
      SUBROUTINE E2STAT(IUNIT
     &
     &                         )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE WRITES CONTROL AND INDEX VECTOR STATISTICS TO
C       UNIT IUNIT
C
C*****************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
C*****************************************************************
C
      CALL UTHEAD('EULER CONTROL AND INDEX VECTORS')
C
C     FIND FIRST AND LAST LEVEL WHICH HAVE CELLS
C
      DO 10 IMGL=-MLVLG2,MLVLG2
      IF(ILVLG2(1,IMGL).LE.ILVLG2(6,IMGL)) THEN
         IBEG=IMGL
         GOTO 20
      ENDIF
10    CONTINUE
C
      CALL UTEROR(+1,REAL(MLVLG2),0.0)
C
20    DO 30 IMGL=MLVLG2,-MLVLG2,-1
      IF(ILVLG2(1,IMGL).LE.ILVLG2(6,IMGL)) THEN
         IEND=IMGL
         GOTO 40
      ENDIF
30    CONTINUE
C
      CALL UTEROR(+2,REAL(MLVLG2),0.0)
C
40    WRITE(IUNIT,45) NNODG2,
     &                NCELG2,
     &                NBNDG2,
     &                NBODG2
45    FORMAT(/' SUMMARY:'/
     &        10X,'NODE  ',I6/
     &        10X,'CELL  ',I6/
     &        10X,'BOUND ',I6/
     &        10X,'BODY  ',I6)
C
```

```
C*****FLUX BALANCE, DISTRIBUTION, AND SMOOTHING
C
      WRITE(IUNIT,50)
50    FORMAT(/' FLUX BALANCE, DISTRIBUTION AND SMOOTHING:'          /
     &       10X,' MGLVL  ---STANDARD---  -NON-STANDARD-  '        ,
     &           '----COARSE----   TOTAL  -------INTERFACES------ '/
     &       10X,'         CELLS  FRACT    CELLS  FRACT   '        ,
     &           ' CELLS  FRACT    CELLS  SOUTH  EAST NORTH  WEST ')
C
      DO 70 IMGL=IBEG,IEND
C
      NFINE =ILVLG2(2,IMGL)-ILVLG2(1,IMGL)+1
      NOUTSD=ILVLG2(4,IMGL)-ILVLG2(3,IMGL)+1
      NCOARS=ILVLG2(6,IMGL)-ILVLG2(5,IMGL)+1
      NTOTAL=ILVLG2(6,IMGL)-ILVLG2(1,IMGL)+1
C
      IF(NTOTAL.GT.0) THEN
         FFINE =REAL(NFINE )/REAL(NTOTAL)
         FOUTSD=REAL(NOUTSD)/REAL(NTOTAL)
         FCOARS=REAL(NCOARS)/REAL(NTOTAL)
      ELSE
         FFINE =0.0
         FOUTSD=0.0
         FCOARS=0.0
      ENDIF
C
C     DETERMINE FREQUENCY OF FLUX MODIFICATIONS FOR CALLS JUST OUTSIDE
C       EMBEDDED DOMAINS
C
      LS=0
      LE=0
      LN=0
      LW=0
C
      DO 55 ICELL=ILVLG2(3,IMGL),ILVLG2(4,IMGL)
      IF(ICELG2(3,ICELL).NE.0) LS=LS+1
      IF(ICELG2(5,ICELL).NE.0) LE=LE+1
      IF(ICELG2(7,ICELL).NE.0) LN=LN+1
      IF(ICELG2(9,ICELL).NE.0) LW=LW+1
55    CONTINUE
C
      WRITE(IUNIT,60) IMGL,NFINE ,FFINE ,NOUTSD,FOUTSD,
     &                NCOARS,FCOARS,NTOTAL,LS,LE,LN,LW
60    FORMAT(10X,I5,3(2X,I6,2X,F6.3),2X,I6,1X,4I6)
C
70    CONTINUE
C
C*****INTERPOLATION
C
      WRITE(IUNIT,80)
80    FORMAT(/' INTERPOLATION TRIPLETS:'                            /
     &       10X,' MGLVL  ----SIDES-----  ---CENTERS----  '.
     &           '----EDGES-----   TOTAL  '                        /
```

```
     &          10X,'         TRPLT  FRACT    TRPLT  FRACT   ',
     &              ' TRPLT  FRACT  TRIPLETS '                )
C
        DO 100 IMGL=IBEG,IEND
C
        NSIDE =LITPE2(2,IMGL)-LITPE2(1,IMGL)+1
        NCENTR=LITPE2(4,IMGL)-LITPE2(3,IMGL)+1
        NEDGE =LITPE2(6,IMGL)-LITPE2(5,IMGL)+1
        NTOTAL=LITPE2(6,IMGL)-LITPE2(1,IMGL)+1
C
        IF(NTOTAL.GT.0) THEN
           FSIDE =REAL(NSIDE )/REAL(NTOTAL)
           FCENTR=REAL(NCENTR)/REAL(NTOTAL)
           FEDGE =REAL(NEDGE )/REAL(NTOTAL)
        ELSE
           FSIDE =0.0
           FCENTR=0.0
           FEDGE =0.0
        ENDIF
C
        WRITE(IUNIT,90) IMGL,NSIDE ,FSIDE ,NCENTR,FCENTR,
     &                                     NEDGE ,FEDGE ,NTOTAL
90      FORMAT(10X,I5,3(2X,I6,2X,F6.3),2X,I6)
C
100     CONTINUE
C
C*****UPDATE AND ZERO
C
        WRITE(IUNIT,110)
110     FORMAT(/' CONTROL VECTORS:'                            /
     &          10X,' MGLVL  ----UPDATE----  -----ZERO-----  '/
     &          10X,'         HITS   FRACT    HITS   FRACT   ')
C
        DO 140 IMGL=IBEG,IEND
        IMASK=2**(IMGL+MLVLG2)
C
        NHITU=0
        NHITZ=0
C
        DO 120 INODE=1,NNODG2
        IF(IAND(CUPDE2(INODE),IMASK).NE.0) NHITU=NHITU+1
        IF(IAND(CZROE2(INODE),IMASK).NE.0) NHITZ=NHITZ+1
120     CONTINUE
C
        FHITU =REAL(NHITU )/REAL(NNODG2)
        FHITZ =REAL(NHITZ )/REAL(NNODG2)
C
        WRITE(IUNIT,130) IMGL,NHITU,FHITU,NHITZ,FHITZ
130     FORMAT(10X,I5,2(2X,I6,2X,F6.3))
C
140     CONTINUE
C
        RETURN
```

```
      END
```

E2TIME

```
      SUBROUTINE E2TIME(IUNIT
     &
     &                        )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CONTROLS INITIALIZATION AND OUTPUTING OF
C       TIMER INFORMATION
C
C     IF IUNIT =0 INITIALIZE TIMERS
C              >0 WRITE TIMES TO UNIT IUNIT
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
C*********************************************************************
C
      IF(IUNIT.EQ.0) THEN
C
C*****INITIALIZE TIMES
C
         TM01E2=0.00
         TM02E2=0.00
         TM03E2=0.00
         TM04E2=0.00
         TM05E2=0.00
         TM06E2=0.00
         TM07E2=0.00
         TM08E2=0.00
         TM09E2=0.00
         TM10E2=0.00
C
      ELSEIF(IUNIT.GT.0) THEN
C
C*****OUTPUT TIMES
C
         TOTAL=TM01E2+TM02E2+TM03E2+TM04E2+TM05E2
     &        +TM06E2+TM07E2+TM08E2+TM09E2+TM10E2
C
         PCT01=100.0*TM01E2/TOTAL
         PCT02=100.0*TM02E2/TOTAL
         PCT03=100.0*TM03E2/TOTAL
         PCT04=100.0*TM04E2/TOTAL
```

```
            PCT05=100.0*TM05E2/TOTAL
            PCT06=100.0*TM06E2/TOTAL
            PCT07=100.0*TM07E2/TOTAL
            PCT08=100.0*TM08E2/TOTAL
            PCT09=100.0*TM09E2/TOTAL
            PCT10=100.0*TM10E2/TOTAL
C
            WRITE(IUNIT,10) TM01E2,TM02E2,TM03E2,TM04E2,TM05E2,
     &                      TM06E2,TM07E2,TM08E2,TM09E2,TM10E2,TOTAL,
     &                      PCT01 ,PCT02 ,PCT03 ,PCT04 ,PCT05 ,
     &                      PCT06 ,PCT07 ,PCT08 ,PCT09 ,PCT10
10          FORMAT(' TIMES FOR EULR2D:'                                    /
     &       '         E2CONV    E2WGHT    E2SMTH    E2TRAN    E2ZERO  ',
     &             '  E2DIST    E2BCON    E2INTP    E2UPDT    E2BFIX  ',
     &             '   TOTAL  '                                           /
     &             ' TIME ',11F10.3                                       /
     &             ' PCT  ',10F10.3                                       )
C
          ENDIF
C
          RETURN
          END
```

E2TRAN

```
          SUBROUTINE E2TRAN(IMGL
     &
     &                      )
C
          INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE TRANSPORTS CHANGES FROM THE FINE GRID TO THE
C        CENTERS OF COARSER GRIDS USING THE SUM OF SIMPLE INJECTION
C        AND INWARD DISTRIBUTION
C
C*********************************************************************
C
          INCLUDE '[.GRID2D]G2COMN.INC'
C
          INCLUDE '[.EULR2D]E2COMN.INC'
C
C*********************************************************************
C
C     LOOP THROUGH EACH COARSE CELL ON THIS LEVEL
C
          DO 20 ICELL=ILVLG2(5,IMGL),ILVLG2(6,IMGL)
C
          ICENT=ICELG2(1,ICELL)
          ISW  =ICELG2(2,ICELL)
          ISE  =ICELG2(4,ICELL)
```

```
        INE  =ICELG2(6,ICELL)
        INW  =ICELG2(8,ICELL)
C
C     GEOMETRY OF ALL CELL CORNERS
C
        XSW=GEOMG2(1,ISW)
        YSW=GEOMG2(2,ISW)
        XSE=GEOMG2(1,ISE)
        YSE=GEOMG2(2,ISE)
        XNE=GEOMG2(1,INE)
        YNE=GEOMG2(2,INE)
        XNW=GEOMG2(1,INW)
        YNW=GEOMG2(2,INW)
C
C     COMPUTE PROJECTIONS OF CELL FACES
C
        DXEW=0.5*(XNE+XSE-XNW-XSW)
        DYEW=0.5*(YNE+YSE-YNW-YSW)
        DXNS=0.5*(XNW+XNE-XSW-XSE)
        DYNS=0.5*(YNW+YNE-YSW-YSE)
C
        DEW= SQRT(DXEW*DXEW+DYEW*DYEW)
        DNS= SQRT(DXNS*DXNS+DYNS*DYNS)
C
C     SAVE THE CELL CENTER VALUES
C
        U1C=DPENG2(1,ICENT)
        U2C=DPENG2(2,ICENT)
        U3C=DPENG2(3,ICENT)
        U4C=DPENG2(4,ICENT)
C
C     COMPUTE COMBINATIONS OF THE DEPENDENT VARIABLES AT THE CELL CENTER
C
        U21C=U2C/U1C
        U31C=U3C/U1C
        U41C=U4C/U1C
C
C     COMPUTE THE MAXIMUM STABLE TIME STEP FOR THIS CELL
C
        RHO  =U1C
        PRES =GM03E2*(U4C-0.5*(U2C*U21C+U3C*U31C))
        SOUND= SQRT(GM00E2*PRES/RHO       )
        UCNS = ABS(U21C*DYNS-U31C*DXNS       )+SOUND*DNS
        UCEW = ABS(U21C*DYEW-U31C*DXEW       )+SOUND*DEW
        DTVOL=CFLE2/MAX(UCNS,UCEW)
C
        DXEW=DXEW*DTVOL
        DYEW=DYEW*DTVOL
        DXNS=DXNS*DTVOL
        DYNS=DYNS*DTVOL
C
C     SET UP THE JACOBIAN OF F-MATRIX
C
```

```
        DFDU11= 0.00
        DFDU12= 1.00
        DFDU13= 0.00
        DFDU14 =0.00
C
        DFDU21= GM02E2*U21C*U21C+GM04E2*U31C*U31C
        DFDU22=-GM06E2*U21C
        DFDU23=-GM03E2*U31C
        DFDU24= GM03E2
C
        DFDU31=-U21C*U31C
        DFDU32= U31C
        DFDU33= U21C
        DFDU34= 0.00
C
        DFDU41=-GM00E2*U21C*U41C+GM03E2*U21C*(U21C*U21C+U31C*U31C)
        DFDU42= GM00E2*U41C     -GM06E2*U21C*U21C-GM04E2*U31C*U31C
        DFDU43=-GM03E2*U21C*U31C
        DFDU44= GM00E2*U21C
C
C     SET UP THE JACOBIAN OF G-MATRIX
C
        DGDU11= 0.00
        DGDU12= 0.00
        DGDU13= 1.00
        DGDU14 =0.00
C
        DGDU21=-U21C*U31C
        DGDU22= U31C
        DGDU23= U21C
        DGDU24= 0.00
C
        DGDU31= GM02E2*U31C*U31C+GM04E2*U21C*U21C
        DGDU32=-GM03E2*U21C
        DGDU33=-GM06E2*U31C
        DGDU34= GM03E2
C
        DGDU41=-GM00E2*U31C*U41C+GM03E2*U31C*(U21C*U21C+U31C*U31C)
        DGDU42=-GM03E2*U21C*U31C
        DGDU43= GM00E2*U41C     -GM06E2*U31C*U31C-GM04E2*U21C*U21C
        DGDU44= GM00E2*U31C
C
C     SAVE CORNER CHANGES AND COMPUTE CHANGES IN F AND G
C
C     ...SOUTHWEST
C
        DU1SW=CHNGE2(1,ISW)
        DU2SW=CHNGE2(2,ISW)
        DU3SW=CHNGE2(3,ISW)
        DU4SW=CHNGE2(4,ISW)
C
        DF1SW=DFDU11*DU1SW+DFDU12*DU2SW+DFDU13*DU3SW+DFDU14*DU4SW
        DF2SW=DFDU21*DU1SW+DFDU22*DU2SW+DFDU23*DU3SW+DFDU24*DU4SW
```

```
      DF3SW=DFDU31*DU1SW+DFDU32*DU2SW+DFDU33*DU3SW+DFDU34*DU4SW
      DF4SW=DFDU41*DU1SW+DFDU42*DU2SW+DFDU43*DU3SW+DFDU44*DU4SW
C
      DG1SW=DGDU11*DU1SW+DGDU12*DU2SW+DGDU13*DU3SW+DGDU14*DU4SW
      DG2SW=DGDU21*DU1SW+DGDU22*DU2SW+DGDU23*DU3SW+DGDU24*DU4SW
      DG3SW=DGDU31*DU1SW+DGDU32*DU2SW+DGDU33*DU3SW+DGDU34*DU4SW
      DG4SW=DGDU41*DU1SW+DGDU42*DU2SW+DGDU43*DU3SW+DGDU44*DU4SW
C
      DDF1SW= DF1SW*DYNS-DG1SW*DXNS
      DDF2SW= DF2SW*DYNS-DG2SW*DXNS
      DDF3SW= DF3SW*DYNS-DG3SW*DXNS
      DDF4SW= DF4SW*DYNS-DG4SW*DXNS
C
      DDG1SW=-DF1SW*DYEW+DG1SW*DXEW
      DDG2SW=-DF2SW*DYEW+DG2SW*DXEW
      DDG3SW=-DF3SW*DYEW+DG3SW*DXEW
      DDG4SW=-DF4SW*DYEW+DG4SW*DXEW
C
      CHNG1=      (DU1SW+DDF1SW+DDG1SW)
      CHNG2=      (DU2SW+DDF2SW+DDG2SW)
      CHNG3=      (DU3SW+DDF3SW+DDG3SW)
      CHNG4=      (DU4SW+DDF4SW+DDG4SW)
C
C     ...SOUTHEAST
C
      DU1SE=CHNGE2(1,ISE)
      DU2SE=CHNGE2(2,ISE)
      DU3SE=CHNGE2(3,ISE)
      DU4SE=CHNGE2(4,ISE)
C
      DF1SE=DFDU11*DU1SE+DFDU12*DU2SE+DFDU13*DU3SE+DFDU14*DU4SE
      DF2SE=DFDU21*DU1SE+DFDU22*DU2SE+DFDU23*DU3SE+DFDU24*DU4SE
      DF3SE=DFDU31*DU1SE+DFDU32*DU2SE+DFDU33*DU3SE+DFDU34*DU4SE
      DF4SE=DFDU41*DU1SE+DFDU42*DU2SE+DFDU43*DU3SE+DFDU44*DU4SE
C
      DG1SE=DGDU11*DU1SE+DGDU12*DU2SE+DGDU13*DU3SE+DGDU14*DU4SE
      DG2SE=DGDU21*DU1SE+DGDU22*DU2SE+DGDU23*DU3SE+DGDU24*DU4SE
      DG3SE=DGDU31*DU1SE+DGDU32*DU2SE+DGDU33*DU3SE+DGDU34*DU4SE
      DG4SE=DGDU41*DU1SE+DGDU42*DU2SE+DGDU43*DU3SE+DGDU44*DU4SE
C
      DDF1SE= DF1SE*DYNS-DG1SE*DXNS
      DDF2SE= DF2SE*DYNS-DG2SE*DXNS
      DDF3SE= DF3SE*DYNS-DG3SE*DXNS
      DDF4SE= DF4SE*DYNS-DG4SE*DXNS
C
      DDG1SE=-DF1SE*DYEW+DG1SE*DXEW
      DDG2SE=-DF2SE*DYEW+DG2SE*DXEW
      DDG3SE=-DF3SE*DYEW+DG3SE*DXEW
      DDG4SE=-DF4SE*DYEW+DG4SE*DXEW
C
      CHNG1=CHNG1+(DU1SE-DDF1SE+DDG1SE)
      CHNG2=CHNG2+(DU2SE-DDF2SE+DDG2SE)
      CHNG3=CHNG3+(DU3SE-DDF3SE+DDG3SE)
```

```
      CHNG4=CHNG4+(DU4SE-DDF4SE+DDG4SE)
C
C     ...NORTHEAST
C
      DU1NE=CHNGE2(1,INE)
      DU2NE=CHNGE2(2,INE)
      DU3NE=CHNGE2(3,INE)
      DU4NE=CHNGE2(4,INE)
C
      DF1NE=DFDU11*DU1NE+DFDU12*DU2NE+DFDU13*DU3NE+DFDU14*DU4NE
      DF2NE=DFDU21*DU1NE+DFDU22*DU2NE+DFDU23*DU3NE+DFDU24*DU4NE
      DF3NE=DFDU31*DU1NE+DFDU32*DU2NE+DFDU33*DU3NE+DFDU34*DU4NE
      DF4NE=DFDU41*DU1NE+DFDU42*DU2NE+DFDU43*DU3NE+DFDU44*DU4NE
C
      DG1NE=DGDU11*DU1NE+DGDU12*DU2NE+DGDU13*DU3NE+DGDU14*DU4NE
      DG2NE=DGDU21*DU1NE+DGDU22*DU2NE+DGDU23*DU3NE+DGDU24*DU4NE
      DG3NE=DGDU31*DU1NE+DGDU32*DU2NE+DGDU33*DU3NE+DGDU34*DU4NE
      DG4NE=DGDU41*DU1NE+DGDU42*DU2NE+DGDU43*DU3NE+DGDU44*DU4NE
C
      DDF1NE= DF1NE*DYNS-DG1NE*DXNS
      DDF2NE= DF2NE*DYNS-DG2NE*DXNS
      DDF3NE= DF3NE*DYNS-DG3NE*DXNS
      DDF4NE= DF4NE*DYNS-DG4NE*DXNS
C
      DDG1NE=-DF1NE*DYEW+DG1NE*DXEW
      DDG2NE=-DF2NE*DYEW+DG2NE*DXEW
      DDG3NE=-DF3NE*DYEW+DG3NE*DXEW
      DDG4NE=-DF4NE*DYEW+DG4NE*DXEW
C
      CHNG1=CHNG1+(DU1NE-DDF1NE-DDG1NE)
      CHNG2=CHNG2+(DU2NE-DDF2NE-DDG2NE)
      CHNG3=CHNG3+(DU3NE-DDF3NE-DDG3NE)
      CHNG4=CHNG4+(DU4NE-DDF4NE-DDG4NE)
C
C     ...NORTHWEST
C
      DU1NW=CHNGE2(1,INW)
      DU2NW=CHNGE2(2,INW)
      DU3NW=CHNGE2(3,INW)
      DU4NW=CHNGE2(4,INW)
C
      DF1NW=DFDU11*DU1NW+DFDU12*DU2NW+DFDU13*DU3NW+DFDU14*DU4NW
      DF2NW=DFDU21*DU1NW+DFDU22*DU2NW+DFDU23*DU3NW+DFDU24*DU4NW
      DF3NW=DFDU31*DU1NW+DFDU32*DU2NW+DFDU33*DU3NW+DFDU34*DU4NW
      DF4NW=DFDU41*DU1NW+DFDU42*DU2NW+DFDU43*DU3NW+DFDU44*DU4NW
C
      DG1NW=DGDU11*DU1NW+DGDU12*DU2NW+DGDU13*DU3NW+DGDU14*DU4NW
      DG2NW=DGDU21*DU1NW+DGDU22*DU2NW+DGDU23*DU3NW+DGDU24*DU4NW
      DG3NW=DGDU31*DU1NW+DGDU32*DU2NW+DGDU33*DU3NW+DGDU34*DU4NW
      DG4NW=DGDU41*DU1NW+DGDU42*DU2NW+DGDU43*DU3NW+DGDU44*DU4NW
C
      DDF1NW= DF1NW*DYNS-DG1NW*DXNS
      DDF2NW= DF2NW*DYNS-DG2NW*DXNS
```

```
        DDF3NW= DF3NW*DYNS-DG3NW*DXNS
        DDF4NW= DF4NW*DYNS-DG4NW*DXNS
C
        DDG1NW=-DF1NW*DYEW+DG1NW*DXEW
        DDG2NW=-DF2NW*DYEW+DG2NW*DXEW
        DDG3NW=-DF3NW*DYEW+DG3NW*DXEW
        DDG4NW=-DF4NW*DYEW+DG4NW*DXEW
C
        CHNG1=CHNG1+(DU1NW+DDF1NW-DDG1NW)
        CHNG2=CHNG2+(DU2NW+DDF2NW-DDG3NW)
        CHNG3=CHNG3+(DU3NW+DDF3NW-DDG3NW)
        CHNG4=CHNG4+(DU4NW+DDF4NW-DDG4NW)
C
C     DISTRIBUTE CHANGES
C
        CHNGE2(1,ICENT)=CHNGE2(1,ICENT)+0.25*CHNG1
        CHNGE2(2,ICENT)=CHNGE2(2,ICENT)+0.25*CHNG2
        CHNGE2(3,ICENT)=CHNGE2(3,ICENT)+0.25*CHNG3
        CHNGE2(4,ICENT)=CHNGE2(4,ICENT)+0.25*CHNG4
C
C     GO BACK FOR NEXT CELL ON THIS LEVEL
C
20      CONTINUE
C
        RETURN
        END
```

E2UPDT

```
        SUBROUTINE E2UPDT(IMGL
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE UPDATES THE DEPENDENT VARIABLES AT EACH NODE
C        ASSOCIATED WITH A CELL ON THIS AND ALL FINER CELLS
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
        IMASK=2**(IMGL+NVLG2)
        DO 10 INODE=1,NNODG2
```

```
         IF(IAND(CUPDE2(INODE),IMASK).NE.0) THEN
            DPENG2(1,INODE)=DPENG2(1,INODE)+CHNGE2(1,INODE)
            DPENG2(2,INODE)=DPENG2(2,INODE)+CHNGE2(2,INODE)
            DPENG2(3,INODE)=DPENG2(3,INODE)+CHNGE2(3,INODE)
            DPENG2(4,INODE)=DPENG2(4,INODE)+CHNGE2(4,INODE)
         ENDIF
10       CONTINUE
C
         RETURN
         END
```

E2VRFY

```
         SUBROUTINE E2VRFY(IBEG,IEND,ISTEP,IUNIT
     &
     &                                          )
C
         INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE COMPUTES THE NET MASS, MOMENTUM, AND ENERGY
C        BALANCES THROUGH LINES STARTING AT INODE=IBEG,IEND,ISTEP
C
C*********************************************************************
C
         INCLUDE '[.GRID2D]G2COMN.INC'
C
         INCLUDE '[.EULR2D]E2COMN.INC'
C
         PARAMETER (MNODE=500)
         DIMENSION ANODE(MNODE)
C
C*********************************************************************
C
         DMASSO=1.00
         DXMOMO=1.00
         DYMOMO=1.00
         DENGYO=1.00
C
         WRITE(IUNIT,10)
10       FORMAT(' GLOBAL FLUX BALANCES AND CIRCULATION:'  /
     &          ' ISTART  NNODE   RADIUS        DMASS    ',
     &           '    DXMOM        DYMOM        DENGY    ',
     &           '   2*CIRC        DXPRJ        DYPRJ    ')
C
         NLINE=(IEND-IBEG)/ISTEP+1
         DO 40 ILINE=1,NLINE
C
         ISTART=IBEG+(ILINE-1)*ISTEP
C
C     GET THE NODES ALONG THIS LINE
```

```
C
        CALL G2LINE(0,ISTART,ANODE,NNODE)
        ANODE(NNODE+1)=ANODE(1)
C
C     COMPUTE THE FREE-STREAM STAGNATION ENTHALPY FOR SCALING
C
        HRF=GM00E2*(U4RFE2/U1RFE2)-GM04E2*(U2RFE2*U2RFE2+U3RFE2*U3RFE2)
     &                                    /(U1RFE2*U1RFE2)
C
C     INITIALIZE THE NET FLUXES
C
        SARC =0.00
        RADUS=0.00
        DXPRJ=0.00
        DYPRJ=0.00
        DMASS=0.00
        DXMOM=0.00
        DYMOM=0.00
        DENGY=0.00
C
        CIRC =0.00
C
C     LOOP THROUGH ALL NODE PAIRS ON THIS LINE
C
        IF(ISTART.GE.0) THEN
           LNODE=NNODE
        ELSE
           LNODE=NNODE-1
        ENDIF
C
        DO 20 KNODE=1,LNODE
        INODE=NINT(ANODE(KNODE  ))
        JNODE=NINT(ANODE(KNODE+1))
C
C     CALCULATE THE PROJECTIONS OF THE FACE BETWEEN THESE TWO NODES
C
        DX=GEOMG2(1,JNODE)-GEOMG2(1,INODE)
        DY=GEOMG2(2,JNODE)-GEOMG2(2,INODE)
C
C     CALCULATE THE FLUX QUANTITIES AT BOTH NODES
C
        U1I=DPENG2(1,INODE)
        U2I=DPENG2(2,INODE)
        U3I=DPENG2(3,INODE)
        U4I=DPENG2(4,INODE)
C
        U21I=U2I/U1I
        U31I=U3I/U1I
        PI  =GM03E2*(U4I-0.5*(U2I*U21I+U3I*U31I))
        HI  =GM00E2*(U4I/U1I)-GM04E2*(U21I*U21I+U31I*U31I)
C
        F1I=U2I
        F2I=U2I*U21I
```

```
      F3I=U2I*U31I
      F4I=U2I*HI/HRF
C
      G1I=U3I
      G2I=U3I*U21I
      G3I=U3I*U31I
      G4I=U3I*HI/HRF
C
      U1J=DPENG2(1,JNODE)
      U2J=DPENG2(2,JNODE)
      U3J=DPENG2(3,JNODE)
      U4J=DPENG2(4,JNODE)
C
      U21J=U2J/U1J
      U31J=U3J/U1J
      PJ  =GM03E2*(U4J-0.5*(U2J*U21J+U3J*U31J))
      HJ  =GM00E2*(U4J/U1J)-GM04E2*(U21J*U21J+U31J*U31J)
C
      F1J=U2J
      F2J=U2J*U21J
      F3J=U2J*U31J
      F4J=U2J*HJ/HRF
C
      G1J=U3J
      G2J=U3J*U21J
      G3J=U3J*U31J
      G4J=U3J*HJ/HRF
C
      DXPRJ=DXPRJ+                                          DX
      DYPRJ=DYPRJ+                        DY
      DMASS=DMASS+((F1J   )+(F1I   ))*DY-((G1J   )+(G1I   ))*DX
      DXMOM=DXMOM+((F2J+PJ)+(F2I+PI))*DY-((G2J   )+(G2I   ))*DX
      DYMOM=DYMOM+((F3J   )+(F3I   ))*DY-((G3J+PJ)+(G3I+PI))*DX
      DENGY=DENGY+((F4J   )+(F4I   ))*DY-((G4J   )+(G4I   ))*DX
C
      CIRC =CIRC +((U21I+U21J)*DX+(U31I+U31J)*DY)
C
      DSARC=SQRT(DX*DX+DY*DY)
      SARC =SARC +DSARC
C
      IF(ISTART.GE.0) THEN
         RADUS=RADUS
     &        +DSARC*0.50*(SQRT(GEOMG2(1,INODE)**2+GEOMG2(2,INODE)**2)
     &                    +SQRT(GEOMG2(1,JNODE)**2+GEOMG2(2,JNODE)**2))
      ELSE
         RADUS=RADUS+DSARC*0.50*(GEOMG2(1,INODE)+GEOMG2(1,JNODE))
      ENDIF
C
20    CONTINUE
C
      RADUS=RADUS/SARC
C
      IF(ISTART.LE.0 .AND. ILINE.EQ.1) THEN
```

```
            DMASSO=DMASS+1.0E-10
            DXMOMO=DXMOM+1.0E-10
            DYMOMO=DYMOM+1.0E-10
            DENGYO=DENGY+1.0E-10
          ENDIF
C
C       WRITE OUT THE NET FLUXES THROUGH THIS LINE (LOOP)
C
          WRITE(IUNIT,30) ISTART,NNODE,RADUS,DMASS/DMASSO,DXMOM/DXMOMO,
     &                                       DYMOM/DYMOMO,DENGY/DENGYO,
     &                                       CIRC,DXPRJ,DYPRJ
30        FORMAT(2I7,8G13.6)
C
C       NEXT LINE (OR LOOP)
C
40        CONTINUE
C
          RETURN
          END
```

E2WGH1

```
          SUBROUTINE E2WGH1
C
          INCLUDE '[.UTILITY]PROLOG.INC'
C
C       THIS SUBROUTINE CALCULATES THE NEW SMOOTHING COEFFICIENT
C          DISTRIBUTION FOR CELLS WHICH ARE NOT NEAR BOUNDARIES AND
C          NOT JUST OUTSIDE EMBEDDED DOMAINS
C
C*********************************************************************
C
          INCLUDE '[.GRID2D]G2COMN.INC'
C
          INCLUDE '[.EULR2D]E2COMN.INC'
C
C*********************************************************************
C
C       STEP THROUGH EACH FINE CELL TO ACCUMULATE SECOND DIFFERENCES
C          OF THE WEIGHT FUNCTION AND DENSITY (FORCING FUNCTION)
C
          DO 20 IMGL=0,MLVLG2
C
          DO 10 ICELL=ILVLG2(1,IMGL),ILVLG2(2,IMGL)
C
C       SET UP NODE POINTERS FOR THIS CELL
C
          ISW  =ICELG2(2,ICELL)
          ISE  =ICELG2(4,ICELL)
          INE  =ICELG2(6,ICELL)
```

```
        INW  =ICELG2(8,ICELL)
C
C     SAVE SMOOTHING WEIGHTING FACTOR AND FORCING FUNCTION
C       AT ALL CELL CORNERS
C
        U4SW=DPENG2(4,ISW)
        WTSW=DPENG2(5,ISW)
        U4SE=DPENG2(4,ISE)
        WTSE=DPENG2(5,ISE)
        U4NE=DPENG2(4,INE)
        WTNE=DPENG2(5,INE)
        U4NW=DPENG2(4,INW)
        WTNW=DPENG2(5,INW)
C
C     ACCUMULATE 4*DX*DX*(UXX+UYY) IN CHNGE2(1)
C                24*UAVG           IN CHNGE2(2)
C                4*DX*DX*(WXX+WYY) IN CHNGE2(3)
C
C     SOUTHWEST CORNER
C
        CHNGE2(1,ISW)=CHNGE2(1,ISW)+(U4SE+U4NE+U4NW-3.0*U4SW)
        CHNGE2(2,ISW)=CHNGE2(2,ISW)+(U4SE+U4NE+U4NW+3.0*U4SW)
        CHNGE2(3,ISW)=CHNGE2(3,ISW)+(WTSE+WTNE+WTNW-3.0*WTSW)
C
C     SOUTHEAST CORNER
C
        CHNGE2(1,ISE)=CHNGE2(1,ISE)+(U4NE+U4NW+U4SW-3.0*U4SE)
        CHNGE2(2,ISE)=CHNGE2(2,ISE)+(U4NE+U4NW+U4SW+3.0*U4SE)
        CHNGE2(3,ISE)=CHNGE2(3,ISE)+(WTNE+WTNW+WTSW-3.0*WTSE)
C
C     NORTHEAST CORNER
C
        CHNGE2(1,INE)=CHNGE2(1,INE)+(U4NW+U4SW+U4SE-3.0*U4NE)
        CHNGE2(2,INE)=CHNGE2(2,INE)+(U4NW+U4SW+U4SE+3.0*U4NE)
        CHNGE2(3,INE)=CHNGE2(3,INE)+(WTNW+WTSW+WTSE-3.0*WTNE)
C
C     NORTHWEST CORNER
C
        CHNGE2(1,INW)=CHNGE2(1,INW)+(U4SW+U4SE+U4NE-3.0*U4NW)
        CHNGE2(2,INW)=CHNGE2(2,INW)+(U4SW+U4SE+U4NE+3.0*U4NW)
        CHNGE2(3,INW)=CHNGE2(3,INW)+(WTSW+WTSE+WTNE-3.0*WTNW)
C
C     GO BACK FOR NEXT CELL
C
10      CONTINUE
C
20      CONTINUE
C
        RETURN
        END
```

E2WGH2

```
      SUBROUTINE E2WGH2
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CALCULATES THE NEW SMOOTHING COEFFICIENT
C       DISTRIBUTION FOR ALL CELLS WHICH ARE EITHER NEAR A BOUNDARY
C       OR JUST OUTSIDE AN EMBEDDED DOMAIN
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.EULR2D]E2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
C     STEP THROUGH EACH FINE CELL TO ACCUMULATE SECOND DIFFERENCES
C       OF THE WEIGHT FUNCTION AND DENSITY (FORCING FUNCTION)
C
      DO 10 IMGL=0,MLVLG2
      DO 10 ICELL=ILVLG2(3,IMGL),ILVLG2(4,IMGL)
C
C     SET UP NODE POINTERS FOR THIS CELL
C
      ICENT=ICELG2(1,ICELL)
      ISW  =ICELG2(2,ICELL)
      IS   =ICELG2(3,ICELL)
      ISE  =ICELG2(4,ICELL)
      IE   =ICELG2(5,ICELL)
      INE  =ICELG2(6,ICELL)
      IN   =ICELG2(7,ICELL)
      INW  =ICELG2(8,ICELL)
      IW   =ICELG2(9,ICELL)
C
      IAUX =ICELG2(10,ICELL)
C
C     SAVE SMOOTHING WEIGHTING FACTOR AND FORCING FUNCTION
C       AT ALL CELL CORNERS
C
      U4SW=DPENG2(4,ISW)
      WTSW=DPENG2(5,ISW)
C
      U4SE=DPENG2(4,ISE)
      WTSE=DPENG2(5,ISE)
C
      U4NE=DPENG2(4,INE)
      WTNE=DPENG2(5,INE)
```

```
C
        U4NW=DPENG2(4,INW)
        WTNW=DPENG2(5,INW)
C
        IF(IS.EQ.0) THEN
           U4S=0.5*(U4SW+U4SE)
           WTS=0.5*(WTSW+WTSE)
        ELSE
           U4S=DPENG2(4,IS)
           WTS=DPENG2(5,IS)
        ENDIF
C
        IF(IE.EQ.0) THEN
           U4E=0.5*(U4SE+U4NE)
           WTE=0.5*(WTSE+WTNE)
        ELSE
           U4E=DPENG2(4,IE)
           WTE=DPENG2(5,IE)
        ENDIF
C
        IF(IN.EQ.0) THEN
           U4N=0.5*(U4NE+U4NW)
           WTN=0.5*(WTNE+WTNW)
        ELSE
           U4N=DPENG2(4,IN)
           WTN=DPENG2(5,IN)
        ENDIF
C
        IF(IW.EQ.0) THEN
           U4W=0.5*(U4SW+U4NW)
           WTW=0.5*(WTSW+WTNW)
        ELSE
           U4W=DPENG2(4,IW)
           WTW=DPENG2(5,IW)
        ENDIF
C
C     SET UP DEPENDENT VARIABLES AT THE CELL CENTER
C
        IF(ICENT.EQ.0) THEN
           U4C=0.25*(U4SW+U4SE+U4NE+U4NW)
           WTC=0.25*(WTSW+WTSE+WTNE+WTNW)
        ELSE
           U4C=DPENG2(4,ICENT)
           WTC=DPENG2(5,ICENT)
        ENDIF
C
C     ACCUMULATE 4*DX*DX*(UXX+UYY) IN CHNGE2(1)
C                24*UAVG           IN CHNGE2(2)
C                4*DX*DX*(WXX+WYY) IN CHNGE2(3)
C
C     SOUTHWEST CORNER
C
        IF(IAND(IAUX,HL0001).EQ.HL0000) THEN
```

```
            IF(IAND(IAUX,HL0010).EQ.HL0000) THEN
C
C           ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
               CHNGE2(1,ISW)=CHNGE2(1,ISW)+(U4SE+U4NE+U4NW-3.0*U4SW)
               CHNGE2(2,ISW)=CHNGE2(2,ISW)+(U4SE+U4NE+U4NW+3.0*U4SW)
               CHNGE2(3,ISW)=CHNGE2(3,ISW)+(WTSE+WTNE+WTNW-3.0*WTSW)
C
            ELSE
C
C           ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,ISW)=CHNGE2(1,ISW)+(U4S +U4C +U4W -3.0*U4SW)
               CHNGE2(2,ISW)=CHNGE2(2,ISW)+(U4S +U4C +U4W +3.0*U4SW)
               CHNGE2(3,ISW)=CHNGE2(3,ISW)+(WTS +WTC +WTW -3.0*WTSW)
C
            ENDIF
         ELSEIF(IAND(IAUX,HL000B).EQ.HL000B) THEN
C
C           ---CORNER BOUNDARY CELL (SOUTHWEST CORNER OF DOMAIN)
               CHNGE2(1,ISW)=0.0
               CHNGE2(2,ISW)=1.0
               CHNGE2(3,ISW)=0.0
C
         ELSEIF(IAND(IAUX,HL0009).EQ.HL0009) THEN
            IF(IAND(IAUX,HL0010).EQ.HL0000) THEN
C
C           ---BOUNDARY ON WESTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
               CHNGE2(1,ISW)=CHNGE2(1,ISW)+4.0*(U4NW-U4SW)
               CHNGE2(2,ISW)=CHNGE2(2,ISW)+6.0*(U4NW+U4SW)
               CHNGE2(3,ISW)=CHNGE2(3,ISW)+4.0*(WTNW-WTSW)
C
            ELSE
C
C           ---BOUNDARY ON WESTERN EDGE, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,ISW)=CHNGE2(1,ISW)+4.0*(U4W -U4SW)
               CHNGE2(2,ISW)=CHNGE2(2,ISW)+6.0*(U4W +U4SW)
               CHNGE2(3,ISW)=CHNGE2(3,ISW)+4.0*(WTW -WTSW)
C
            ENDIF
         ELSEIF(IAND(IAUX,HL0003).EQ.HL0003) THEN
            IF(IAND(IAUX,HL0010).EQ.HL0000) THEN
C
C           ---BOUNDARY ON SOUTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
               CHNGE2(1,ISW)=CHNGE2(1,ISW)+4.0*(U4SE-U4SW)
               CHNGE2(2,ISW)=CHNGE2(2,ISW)+6.0*(U4SE+U4SW)
               CHNGE2(3,ISW)=CHNGE2(3,ISW)+4.0*(WTSE-WTSW)
C
            ELSE
C
C           ---BOUNDARY ON SOUTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,ISW)=CHNGE2(1,ISW)+4.0*(U4S -U4SW)
               CHNGE2(2,ISW)=CHNGE2(2,ISW)+6.0*(U4S +U4SW)
               CHNGE2(3,ISW)=CHNGE2(3,ISW)+4.0*(WTS -WTSW)
C
```

```
            ENDIF
C
          ELSEIF(IAND(IAUX,HL0001).EQ.HL0001) THEN
            CONTINUE
C
          ELSE
            CALL UTEROR(+1,REAL(ICELL),REAL(IAUX))
          ENDIF
C
C     SOUTH SIDE
C
          IF(IS.NE.0) THEN
            CHNGE2(1,IS )=CHNGE2(1,IS )+(U4SW+U4E +2.0*U4C
     &                                   +U4W +U4SW-6.0*U4S)
            CHNGE2(2,IS )=CHNGE2(2,IS )+(U4SW+U4E +2.0*U4C
     &                                   +U4W +U4SW+6.0*U4S)
            CHNGE2(3,IS )=CHNGE2(3,IS )+(WTSW+WTE +2.0*WTC
     &                                   +WTW +WTSW-6.0*WTS)
          ENDIF
C
C     SOUTHEAST CORNER
C
          IF(IAND(IAUX,HL0002).EQ.HL0000) THEN
            IF(IAND(IAUX,HL0020).EQ.HL0000) THEN
C
C             ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
                CHNGE2(1,ISE)=CHNGE2(1,ISE)+(U4NE+U4NW+U4SW-3.0*U4SE)
                CHNGE2(2,ISE)=CHNGE2(2,ISE)+(U4NE+U4NW+U4SW+3.0*U4SE)
                CHNGE2(3,ISE)=CHNGE2(3,ISE)+(WTNE+WTNW+WTSW-3.0*WTSE)
C
            ELSE
C
C             ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
                CHNGE2(1,ISE)=CHNGE2(1,ISE)+(U4E +U4C +U4S -3.0*U4SE)
                CHNGE2(2,ISE)=CHNGE2(2,ISE)+(U4E +U4C +U4S +3.0*U4SE)
                CHNGE2(3,ISE)=CHNGE2(3,ISE)+(WTE +WTC +WTS -3.0*WTSE)
C
            ENDIF
          ELSEIF(IAND(IAUX,HL0007).EQ.HL0007) THEN
C
C             ---CORNER BOUNDARY CELL (SOUTHEAST CORNER OF DOMAIN)
                CHNGE2(1,ISE)=0.0
                CHNGE2(2,ISE)=1.0
                CHNGE2(3,ISE)=0.0
C
          ELSEIF(IAND(IAUX,HL0003).EQ.HL0003) THEN
            IF(IAND(IAUX,HL0020).EQ.HL0000) THEN
C
C             ---BOUNDARY ON SOUTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
                CHNGE2(1,ISE)=CHNGE2(1,ISE)+4.0*(U4SW-U4SE)
                CHNGE2(2,ISE)=CHNGE2(2,ISE)+6.0*(U4SW+U4SE)
                CHNGE2(3,ISE)=CHNGE2(3,ISE)+4.0*(WTSW-WTSE)
C
```

```
            ELSE
C
C           ---BOUNDARY ON SOUTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,ISE)=CHNGE2(1,ISE)+4.0*(U4S -U4SE)
               CHNGE2(2,ISE)=CHNGE2(2,ISE)+6.0*(U4S +U4SE)
               CHNGE2(3,ISE)=CHNGE2(3,ISE)+4.0*(WTS -WTSE)
C
            ENDIF
         ELSEIF(IAND(IAUX,HL0006).EQ.HL0006) THEN
            IF(IAND(IAUX,HL0020).EQ.HL0000) THEN
C
C           ---BOUNDARY ON EASTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
               CHNGE2(1,ISE)=CHNGE2(1,ISE)+4.0*(U4NE-U4SE)
               CHNGE2(2,ISE)=CHNGE2(2,ISE)+6.0*(U4NE+U4SE)
               CHNGE2(3,ISE)=CHNGE2(3,ISE)+4.0*(WTNE-WTSE)
C
            ELSE
C
C           ---BOUNDARY ON EASTERN EDGE, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,ISE)=CHNGE2(1,ISE)+4.0*(U4E -U4SE)
               CHNGE2(2,ISE)=CHNGE2(2,ISE)+6.0*(U4E +U4SE)
               CHNGE2(3,ISE)=CHNGE2(3,ISE)+4.0*(WTE -WTSE)
C
            ENDIF
C
         ELSEIF(IAND(IAUX,HL0002).EQ.HL0002) THEN
            CONTINUE
C
         ELSE
            CALL UTEROR(+2,REAL(ICELL),REAL(IAUX))
         ENDIF
C
C     EAST SIDE
C
         IF(IE.NE.0) THEN
            CHNGE2(1,IE )=CHNGE2(1,IE )+(U4NE+U4N +2.0*U4C
     &                                   +U4S +U4SE-6.0*U4E)
            CHNGE2(2,IE )=CHNGE2(2,IE )+(U4NE+U4N +2.0*U4C
     &                                   +U4S +U4SE+6.0*U4E)
            CHNGE2(3,IE )=CHNGE2(3,IE )+(WTNE+WTN +2.0*WTC
     &                                   +WTS +WTSE-6.0*WTE)
         ENDIF
C
C     NORTHEAST CORNER
C
         IF(IAND(IAUX,HL0004).EQ.HL0000) THEN
            IF(IAND(IAUX,HL0040).EQ.HL0000) THEN
C
C           ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
               CHNGE2(1,INE)=CHNGE2(1,INE)+(U4NW+U4SW+U4SE-3.0*U4NE)
               CHNGE2(2,INE)=CHNGE2(2,INE)+(U4NW+U4SW+U4SE+3.0*U4NE)
               CHNGE2(3,INE)=CHNGE2(3,INE)+(WTNW+WTSW+WTSE-3.0*WTNE)
C
```

```
            ELSE
C
C           ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,INE)=CHNGE2(1,INE)+(U4N +U4C +U4E -3.0*U4NE)
               CHNGE2(2,INE)=CHNGE2(2,INE)+(U4N +U4C +U4E +3.0*U4NE)
               CHNGE2(3,INE)=CHNGE2(3,INE)+(WTN +WTC +WTE -3.0*WTNE)
C
            ENDIF
         ELSEIF(IAND(IAUX,HL000E).EQ.HL000E) THEN
C
C           ---CORNER BOUNDARY CELL (NORTHEAST CORNER OF DOMAIN)
               CHNGE2(1,INE)=0.0
               CHNGE2(2,INE)=1.0
               CHNGE2(3,INE)=0.0
C
         ELSEIF(IAND(IAUX,HL0006).EQ.HL0006) THEN
            IF(IAND(IAUX,HL0040).EQ.HL0000) THEN
C
C           ---BOUNDARY ON EASTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
               CHNGE2(1,INE)=CHNGE2(1,INE)+4.0*(U4SE-U4NE)
               CHNGE2(2,INE)=CHNGE2(2,INE)+6.0*(U4SE+U4NE)
               CHNGE2(3,INE)=CHNGE2(3,INE)+4.0*(WTSE-WTNE)
C
            ELSE
C
C           ---BOUNDARY ON EASTERN EDGE, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,INE)=CHNGE2(1,INE)+4.0*(U4E -U4NE)
               CHNGE2(2,INE)=CHNGE2(2,INE)+6.0*(U4E +U4NE)
               CHNGE2(3,INE)=CHNGE2(3,INE)+4.0*(WTE -WTNE)
C
            ENDIF
         ELSEIF(IAND(IAUX,HL000C).EQ.HL000C) THEN
            IF(IAND(IAUX,HL0040).EQ.HL0000) THEN
C
C           ---BOUNDARY ON NORTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
               CHNGE2(1,INE)=CHNGE2(1,INE)+4.0*(U4NW-U4NE)
               CHNGE2(2,INE)=CHNGE2(2,INE)+6.0*(U4NW+U4NE)
               CHNGE2(3,INE)=CHNGE2(3,INE)+4.0*(WTNW-WTNE)
C
            ELSE
C
C           ---BOUNDARY ON NORTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
               CHNGE2(1,INE)=CHNGE2(1,INE)+4.0*(U4N -U4NE)
               CHNGE2(2,INE)=CHNGE2(2,INE)+6.0*(U4N +U4NE)
               CHNGE2(3,INE)=CHNGE2(3,INE)+4.0*(WTN -WTNE)
C
            ENDIF
C
         ELSEIF(IAND(IAUX,HL0004).EQ.HL0004) THEN
            CONTINUE
C
         ELSE
            CALL UTEROR(+3,REAL(ICELL),REAL(IAUX))
```

```
        ENDIF
C
C     NORTH SIDE
C
        IF(IN.NE.0) THEN
           CHNGE2(1,IN )=CHNGE2(1,IN )+(U4NW+U4W +2.0*U4C
     &                                  +U4E +U4NE-6.0*U4N)
           CHNGE2(2,IN )=CHNGE2(2,IN )+(U4NW+U4W +2.0*U4C
     &                                  +U4E +U4NE+6.0*U4N)
           CHNGE2(3,IN )=CHNGE2(3,IN )+(WTNW+WTW +2.0*WTC
     &                                  +WTE +WTNE-6.0*WTN)
        ENDIF
C
C     NORTHWEST CORNER
C
        IF(IAND(IAUX,HL0008).EQ.HL0000) THEN
           IF(IAND(IAUX,HL0080).EQ.HL0000) THEN
C
C          ---INTERIOR CELL, NOT AT EDGE OF EMBEDDED REGION
              CHNGE2(1,INW)=CHNGE2(1,INW)+(U4SW+U4SE+U4NE-3.0*U4NW)
              CHNGE2(2,INW)=CHNGE2(2,INW)+(U4SW+U4SE+U4NE+3.0*U4NW)
              CHNGE2(3,INW)=CHNGE2(3,INW)+(WTSW+WTSE+WTNE-3.0*WTNW)
C
           ELSE
C
C          ---INTERIOR CELL, JUST OUTSIDE EMBEDDED REGION
              CHNGE2(1,INW)=CHNGE2(1,INW)+(U4W +U4C +U4N -3.0*U4NW)
              CHNGE2(2,INW)=CHNGE2(2,INW)+(U4W +U4C +U4N +3.0*U4NW)
              CHNGE2(3,INW)=CHNGE2(3,INW)+(WTW +WTC +WTN -3.0*WTNW)
C
           ENDIF
        ELSEIF(IAND(IAUX,HL000D).EQ.HL000D) THEN
C
C          ---CORNER BOUNDARY CELL (NORTHWEST CORNER OF DOMAIN)
              CHNGE2(1,INW)=0.0
              CHNGE2(2,INW)=1.0
              CHNGE2(3,INW)=0.0
C
        ELSEIF(IAND(IAUX,HL000C).EQ.HL000C) THEN
           IF(IAND(IAUX,HL0080).EQ.HL0000) THEN
C
C          ---BOUNDARY ON NORTHERN EDGE, NOT AT EDGE OF EMBEDDED REGION
              CHNGE2(1,INW)=CHNGE2(1,INW)+4.0*(U4NE-U4NW)
              CHNGE2(2,INW)=CHNGE2(2,INW)+6.0*(U4NE+U4NW)
              CHNGE2(3,INW)=CHNGE2(3,INW)+4.0*(WTNE-WTNW)
C
           ELSE
C
C          ---BOUNDARY ON NORTHERN EDGE, JUST OUTSIDE EMBEDDED REGION
              CHNGE2(1,INW)=CHNGE2(1,INW)+4.0*(U4N -U4NW)
              CHNGE2(2,INW)=CHNGE2(2,INW)+6.0*(U4N +U4NW)
              CHNGE2(3,INW)=CHNGE2(3,INW)+4.0*(WTN -WTNW)
C
```

```
            ENDIF
          ELSEIF(IAND(IAUX,HL0009).EQ.HL0009) THEN
            IF(IAND(IAUX,HL0080).EQ.HL0000) THEN
C
C           ---BOUNDARY ON WESTERN EDGE, NOT AT EDGE OF EMBEDDED REGION
              CHNGE2(1,INW)=CHNGE2(1,INW)+4.0*(U4SW-U4NW)
              CHNGE2(2,INW)=CHNGE2(2,INW)+6.0*(U4SW+U4NW)
              CHNGE2(3,INW)=CHNGE2(3,INW)+4.0*(WTSW-WTNW)
C
            ELSE
C
C           ---BOUNDARY ON WESTERN EDGE, JUST OUTSIDE EMBEDDED REGION
              CHNGE2(1,INW)=CHNGE2(1,INW)+4.0*(U4W -U4NW)
              CHNGE2(2,INW)=CHNGE2(2,INW)+6.0*(U4W +U4NW)
              CHNGE2(3,INW)=CHNGE2(3,INW)+4.0*(WTW -WTNW)
C
            ENDIF
C
          ELSEIF(IAND(IAUX,HL0008).EQ.HL0008) THEN
            CONTINUE
C
          ELSE
            CALL UTEROR(+4,REAL(ICELL),REAL(IAUX))
          ENDIF
C
C     WEST SIDE
C
          IF(IW.NE.0) THEN
            CHNGE2(1,IW )=CHNGE2(1,IW )+(U4SW+U4S +2.0*U4C
     &                                  +U4N +U4NW-6.0*U4W)
            CHNGE2(2,IW )=CHNGE2(2,IW )+(U4SW+U4S +2.0*U4C
     &                                  +U4N +U4NW+6.0*U4W)
            CHNGE2(3,IW )=CHNGE2(3,IW )+(WTSW+WTS +2.0*WTC
     &                                  +WTN +WTNW-6.0*WTW)
          ENDIF
C
C     GO BACK FOR NEXT CELL
C
10        CONTINUE
C
          RETURN
          END
```

E2WGHT

```
          SUBROUTINE E2WGHT
C
          INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CALCULATES THE NEW SMOOTHING COEFFICIENT
```

```
C        DISTRIBUTION
C
C*************************************************************************
C
         INCLUDE '[.GRID2D]G2COMM.INC'
C
         INCLUDE '[.EULR2D]E2COMM.INC'
C
C*************************************************************************
C
         ALFA=AMUE2(1)
         BETA=AMUE2(2)
         GAMA=AMUE2(3)
         DLTA=AMUE2(4)
C
C     ZERO OUT THE CHANGES BEFORE COMPUTING SMOOTHING WEIGHT FACTOR
C
         DO 10 INODE=1,NNODG2
         CHNGE2(1,INODE)=0.00
         CHNGE2(2,INODE)=0.000001
         CHNGE2(3,INODE)=0.00
10       CONTINUE
C
C     STEP THROUGH EACH FINE CELL TO ACCUMULATE SECOND DIFFERENCES
C        OF THE WEIGHT FUNCTION AND DENSITY (FORCING FUNCTION)
C
         CALL E2WGH1
C
C     STEP THROUGH EACH FINE CELL TO ACCUMULATE SECOND DIFFERENCES
C        OF THE WEIGHT FUNCTION AND DENSITY (FORCING FUNCTION)
C
         CALL E2WGH2
C
C     COMPUTE NEW WEIGHTING FUNCTION AND INITIALIZE FOR SMOOTHING
C
         DO 20 INODE=1,NNODG2
C
C     COMPUTE THE FORCING FUNCTION (BASED UPON NORMALIZED SECOND
C        DIFFERENCE OF DENSITY)
C
         FORCE=GAMA*(1.0+DLTA*ABS(CHNGE2(1,INODE))/CHNGE2(2,INODE))
C
C     INTEGRATE THE WEIGHT FUNCTION EQUATION
C
         DPENG2(5,INODE)=DPENG2(5,INODE)
     &                  +ALFA*(FORCE-DPENG2(5,INODE)+BETA*CHNGE2(3,INODE))
C
C     ZERO OUT THE CHANGES (WORK ARRAY)
C
         CHNGE2(1,INODE)=0.00
         CHNGE2(2,INODE)=0.00
         CHNGE2(3,INODE)=0.00
         CHNGE2(4,INODE)=0.00
```

```
C
20      CONTINUE
C
        RETURN
        END
```

E2WIND

```
        SUBROUTINE E2WIND(
     &                    IPASS,
     &                    ICONV)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CHECKS FOR CONVERGENCE AND DIVERGENCE OF THE
C        TIME MARCHING ITERATIONS. IT USES BOTH RESIDUAL INFORMATION
C        AND CONVERGENCE OF THE LIFT AND DRAG COEFFICIENTS.  IPASS
C        SHOULD BE SET TO 0 FOR THE FIRST ITERATION ON A GIVEN GRID
C        AND WILL SUBSEQUENTLY BE MAINTAINED BY THIS ROUTINE
C
C        ON RETURN, ICONV =-1 DIVERGENCE
C                         = 0 TAKE ANOTHER ITERATION
C                         =+1 CONVERGENCE
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
        DIMENSION CLWIND(100),CDWIND(100)
C
C*************************************************************************
C
        NWIND=MIN(NWNDE2,100)
C
        GOTO (100,200,300,400), (IPASS+1)
C
C*****FIRST ITERATION FOR THIS GRID
C
100     DUMIN=DU2ME2
C
        IPASS=1
        ICONV=0
        RETURN
C
C*****PRE QUASI-CONVERGENCE
C
200     DUMIN=MIN(DU2ME2,DUMIN)
C
```

```
C     CHECK FOR DIVERGENCE
C
        IF(DU2ME2.GT.10.0*DUMIN) THEN
           ICONV=-1
           IPASS=0
           RETURN
        ENDIF
C
C     CHECK FOR QUASI-CONVERGENCE
C
        IF(DU2ME2.LT.TOL1E2) THEN
           IWIND=1
           CLWIND(IWIND)=CLFTE2
           CDWIND(IWIND)=CDRGE2
           ICLMIN=0
           ICDMIN=0
           ICONV=0
           IPASS=2
        ELSE
           ICONV=0
           IPASS=1
        ENDIF
C
        RETURN
C
C*****FILL WINDOWS
C
300     IF(DU2ME2.GT.TOL1E2) GOTO 200
C
        IWIND=IWIND+1
        CLWIND(IWIND)=CLFTE2
        CDWIND(IWIND)=CDRGE2
C
        IF(IWIND.LT.NWIND) THEN
           ICONV=0
           IPASS=2
        ELSE
           ICONV=0
           IPASS=3
        ENDIF
C
        RETURN
C
C*****ADD TO WINDOWS AND CHECK CONVERGENCE
C
400     IF(DU2ME2.GT.TOL1E2) GOTO 200
C
C     DETERMINE SLOT IN WINDOW TABLE TO USE
C
        IWIND=IWIND+1
        IF(IWIND.GT.NWIND) IWIND=1
C
C     ADD THE NEW LIFT AND DRAG TO THE LIST
```

```
C
        CLWIND(IWIND)=CLFTE2
        CDWIND(IWIND)=CDRGE2
C
C     IF EXTREMA HAVE NOT BEEN PREVIOUSLY CALCULATED, OR
C       CLFTE2 WHICH WAS DROPPED OUT OF WINDOW IS CURRENTLY SMALLEST OR
C       LARGEST, THEN RECOMPUTE MINIMUMS AND MAXIMUMS
C
        IF(ICLMIN.EQ.0      .OR.
     &     ICLMIN.EQ.IWIND .OR. ICLMAX.EQ.IWIND) THEN
C
           CLMIN =CLWIND(1)
           ICLMIN=1
           CLMAX =CLWIND(1)
           ICLMAX=1
C
           DO 410 JWIND=2,NWIND
           IF(CLWIND(JWIND).LT.CLMIN) THEN
              CLMIN =CLWIND(JWIND)
              ICLMIN=JWIND
           ENDIF
           IF(CLWIND(JWIND).GT.CLMAX) THEN
              CLMAX =CLWIND(JWIND)
              ICLMAX=JWIND
           ENDIF
410        CONTINUE
C
C     SINCE THE VALUE WHICH WAS DROPPED OUT WAS NOT AN EXTREMUM, ONLY
C        ADJUST BASED UPON LAST ADDED VALUE
C
        ELSE
           IF(CLWIND(IWIND).LT.CLMIN) THEN
              CLMIN =CLWIND(IWIND)
              ICLMIN=IWIND
           ENDIF
           IF(CLWIND(IWIND).GT.CLMAX) THEN
              CLMAX =CLWIND(IWIND)
              ICLMAX=IWIND
           ENDIF
C
        ENDIF
C
C     IF EXTREMA HAVE NOT BEEN PREVIOUSLY CALCULATED, OR
C       CDRGE2 WHICH WAS DROPPED OUT OF WINDOW IS CURRENTLY SMALLEST OR
C       LARGEST, THEN RECOMPUTE MINIMUMS AND MAXIMUMS
C
        IF(ICDMIN.EQ.0      .OR.
     &     ICDMIN.EQ.IWIND .OR. ICDMAX.EQ.IWIND) THEN
C
           CDMIN =CDWIND(1)
           ICDMIN=1
           CDMAX =CDWIND(1)
           ICDMAX=1
```

```
C
          DO 420 JWIND=2,NWIND
          IF(CDWIND(JWIND).LT.CDMIN) THEN
             CDMIN =CDWIND(JWIND)
             ICDMIN=JWIND
          ENDIF
          IF(CDWIND(JWIND).GT.CDMAX) THEN
             CDMAX =CDWIND(JWIND)
             ICDMAX=JWIND
          ENDIF
420       CONTINUE
C
C     SINCE THE VALUE WHICH WAS DROPPED OUT WAS NOT AN EXTREMUM, ONLY
C        ADJUST BASED UPON LAST ADDED VALUE
C
        ELSE
          IF(CDWIND(IWIND).LT.CDMIN) THEN
             CDMIN =CDWIND(IWIND)
             ICDMIN=IWIND
          ENDIF
          IF(CDWIND(IWIND).GT.CDMAX) THEN
             CDMAX =CDWIND(IWIND)
             ICDMAX=IWIND
          ENDIF
C
        ENDIF
C
C     CHECK FOR CONVERGENCE
C
        IF((CLMAX-CLMIN).LT.TOL2E2 .AND. (CDMAX-CDMIN).LT.TOL3E2) THEN
          ICONV=+1
          IPASS=0
        ELSE
          ICONV=0
          IPASS=3
        ENDIF
C
C     WRITE A RECORD TO THE CONVERGENCE HISTORY FILE IF CONVERGED
C
        IF(ICONV.EQ.1) WRITE(IHSTE2,500)
500     FORMAT('    0')
C
        RETURN
C
        END
```

E2ZERO

```
        SUBROUTINE E2ZERO(IMGL
     &
```

```
     &                        )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE INITIALIZES THE CHANGES FOR ALL NODES AT
C        THIS LEVEL, EXCEPT AT THE EDGE OF AN EMBEDDED MESH
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.EULR2D]E2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************************
C
        IMASK=2**(IMGL+MLVLG2)
        DO 10 INODE=1,NNODG2
        IF(IAND(CZROE2(INODE),IMASK).NE.0) THEN
           CHNGE2(1,INODE)=0.00
           CHNGE2(2,INODE)=0.00
           CHNGE2(3,INODE)=0.00
           CHNGE2(4,INODE)=0.00
        ENDIF
10      CONTINUE
C
        RETURN
        END
```

B.4 FEAT2D — Two-Dimensional Feature Detector

F2COMN

```
C-----SCALARS
C     ALFAF2       DIVIDE THRESHOLD SET AT SMALLEST VALUE WHERE
C                  DF/DT<ALFAF2
C     BETAF2       LOWER LIMIT ON DIVIDE THRESHOLD
C     GAMAF2       LIMIT ON DIVIDE THRESHOLD SO THAT NO MORE THAN
C                  GAMAF2*NNODG2 NODE ARE ABOVE THE DIVIDE THRESHOLD
C     DLTAF2       FUSION THRESHOLD (INPUT)
C     THRDF2       DIVIDE THRESHOLD
C     THRFF2       FUSION THRESHOLD
C     NFFSF2       NUMBER OF TIMES THE FEATURE FINDER HAS BEEN USED
C     TITLF2       DESCRIPTION OF REFINEMENT PARAMETER
C
```

```
C-----REFINEMENT SINGULARITY ARRAYS
C     XSNGF2(I)  I=1,NSNGF2
C                  GEOM1 LOCATION OF CENTER OF ITH REFINEMENT SINGULARITY
C     YSNGF2(I)  I=1,NSNGF2
C                  GEOM2 LOCATION OF CENTER OF ITH REFINEMENT SINGULARITY
C     RSNGF2(I)  I=1,NSNGF2
C                  RADIUS OF ITH REFINEMENT SINGULARITY
C
C-----CHNGF2(I,INODE)   INODE=1,MNODG2
C            I=1   REFINEMENT PARAMETER
C             =2   SCRATH STORAGE
C
        PARAMETER (MDUMO1=9*MCELG2+1,
     &             MSNGF2=10        )
C
        DIMENSION CHNGF2(2,MNODG2)
C
        EQUIVALENCE (DUMMG2(MDUMO1),CHNGF2(1,1))
C
        COMMON /F2COMN/ ALFAF2,BETAF2,GAMAF2,DLTAF2,
     &                  THRDF2,THRFF2,
     &                  NFFSF2,NSNGF2,
     &                  XSNGF2(MSNGF2),YSNGF2(MSNGF2),RSNGF2(MSNGF2)
C
        CHARACTER*24 TITLF2
C
        COMMON /F2CHAR/ TITLF2
```

F2ADPT

```
        SUBROUTINE F2ADPT(TDIVD,TFUSE,
     &
     &                    NCHANG     )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE MARKS A CELL FOR DIVISION AND/OR FUSION BASED
C        UPON ITS REFINEMENT PARAMETERS AND THE DIVISION AND FUSION
C        THRESHOLDS
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.FEAT2D]F2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
```

```
      NCHANG=0
C
C     COPY THE REFINEMENT PARAMETER INTO THE WORK ARRAY
C
      DO 10 INODE=1,NNODG2
      WORKG2(INODE)=CHNGF2(1,INODE)
10    CONTINUE
C
      TITLG2='REFN PAR'
C
C     MODIFY THE REFINEMENT PARAMETER NEAR REFINEMENT SINGULARITIES
C
      DO 30 ISING=1,NSNGF2
      RSNG2=RSNGF2(ISING)*RSNGF2(ISING)
C
      DO 20 INODE=1,NNODG2
      RAD2=(GEOMG2(1,INODE)-XSNGF2(ISING))**2
     &     +(GEOMG2(2,INODE)-YSNGF2(ISING))**2
      IF(RAD2.LT.RSNG2) WORKG2(INODE)=WORKG2(INODE)+5.0
20    CONTINUE
C
30    CONTINUE
C
C     LOOP THROUGH ALL CELLS ON ALL FINE MULTIPLE-GRID-LEVELS
C
      DO 40 ICELL=ILVLG2(1,0),ILVLG2(6,MLVLG2-1)
      IF(ICELG2(2,ICELL).EQ.0) GOTO 40
C
C     FIND CHANGES AT THE FOUR CORNER NODES
C
      CHNGSW=WORKG2(ICELG2(2,ICELL))
      CHNGSE=WORKG2(ICELG2(4,ICELL))
      CHNGNE=WORKG2(ICELG2(6,ICELL))
      CHNGNW=WORKG2(ICELG2(8,ICELL))
C
C     MARK CELL FOR DIVISION IF ANY CHANGE IS ABOVE THE DIVIDE THRESHOLD
C        (ALSO UNMARK FOR FUSION)
C
      IF(CHNGSW.GE.TDIVD .OR. CHNGSE.GE.TDIVD .OR.
     &   CHNGNE.GE.TDIVD .OR. CHNGNW.GE.TDIVD    ) THEN
        ICELG2(10,ICELL)=IOR (ICELG2(10,ICELL),HU8000)
        ICELG2(10,ICELL)=IAND(ICELG2(10,ICELL),HUBFFF)
        NCHANG=NCHANG+1
C
C     IF CELL WAS PREVIOUSLY MARKED FOR FUSION AND IF ANY CHANGES ARE
C        ABOVE THE FUSION THRESHOLD, UNMARK THE CELL FOR FUSION
C
      ELSEIF(IAND(ICELG2(10,ICELL),HU4000).NE.0 .AND.
     &       (CHNGSW.GE.TFUSE .OR. CHNGSE.GE.TFUSE .OR.
     &        CHNGNE.GE.TFUSE .OR. CHNGNW.GE.TFUSE    )) THEN
        ICELG2(10,ICELL)=IAND(ICELG2(10,ICELL),HUBFFF)
        NCHANG=NCHANG+1
C
```

```
C     MARK CELL FOR FUSION IF CHANGES ARE ALL BELOW THE FUSE THRESHOLD
C        AND CELL IS NOT ALREADY MARKED FOR DIVISION AND THIS IS THE
C        FIRST FEATURE FINDING ON THIS GRID
C
        ELSEIF(NFFSF2                          .EQ.0     .AND.
     &         IAND(ICELG2(10,ICELL),HU8000).EQ.0     .AND.
     &         CHNGSW.LT.TFUSE .AND. CHNGSE.LT.TFUSE .AND.
     &         CHNGNE.LT.TFUSE .AND. CHNGNW.LT.TFUSE     ) THEN
C
          ICELG2(10,ICELL)=IOR(ICELG2(10,ICELL),HU4000)
          NCHANG=NCHANG+1
C
        ENDIF
C
40      CONTINUE
C
C     INCREMENT NUMBER OF FEATURE FINDINGS ON THIS GRID
C
        NFFSF2=NFFSF2+1
C
        RETURN
        END
```

F2CHNG

```
        SUBROUTINE F2CHNG(ITYPE
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE SETS UP THE REFINEMENT PARAMETER BY CALLING:
C        ITYPE=0  F2VALU
C             =1  F2GRAD
C             =2  F2LAPL
C
C*****************************************************************************
C
        IF(ITYPE.EQ.0) THEN
           CALL F2VALU
        ELSEIF(ITYPE.EQ.1) THEN
           CALL F2GRAD
        ELSEIF(ITYPE.EQ.2) THEN
           CALL F2LAPL
        ENDIF
C
        RETURN
        END
```

F2DATA

```
      SUBROUTINE F2DATA(IVAR
     &
     &                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CREATES OUTPUT ARRAYS
C        IF IVAR =  1      DATA$ = CHNGF2(1)
C               =  2            CHNGF2(2)
C               =  3            CHNGF2(1) (CLIPPED)
C
C*******************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.FEAT2D]F2COMN.INC'
C
C*******************************************************************************
C
C     BRANCH OUT BASED ON INPUT PARAMETER
C
      IF(IVAR.EQ. 1) GOTO  100
      IF(IVAR.EQ. 2) GOTO  200
      IF(IVAR.EQ. 3) GOTO  300
                     RETURN
C
C*****CHANGE 1
C
100   DO 110 I=1,NNODG2
      WORKG2(I)=CHNGF2(1,I)
110   CONTINUE
C
      TITLG2=TITLF2(1:1)//TITLF2(13:19)
C
      RETURN
C
C*****CHANGE 2
C
200   DO 210 I=1,NNODG2
      WORKG2(I)=CHNGF2(2,I)
210   CONTINUE
C
      TITLG2='F2-SCRCH'
C
      RETURN
C
C*****CHANGE 1 (CLIPPED)
C
300   CALL UTINPF('CLIP',CLIP)
```

```
C
      DO 310 I=1,NNODG2
      WORKG2(I)=MIN(CHNGF2(1,I),CLIP)
310   CONTINUE
C
      TITLG2=TITLF2(1:1)//TITLF2(13:19)
C
      RETURN
C
      END
```

F2GRAD

```
      SUBROUTINE F2GRAD
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CALCULATES THE FIRST DIFFERENCE OF WORKG2 AT
C       EACH NODE AND THEN NORMALIZES THEM BY THE AVERAGE
C
C*****************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.FEAT2D]F2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************************
C
C     ZERO OUT THE CHANGE AT EVERY NODE
C
      DO 10 INODE=1,NNODG2
      CHNGF2(1,INODE)=0.0
      CHNGF2(2,INODE)=0.0
10    CONTINUE
C
C     STEP THROUGH EACH FINE CELL TO ACCUMULATE AVERAGE DIFFERENCE
C
      DO 20 ICELL=1,NCELG2
C
      IC   =ICELG2(1,ICELL)
      IF(IC.NE.0) GOTO 20
C
C     SET UP NODE POINTERS FOR THIS CELL
C
      ISW  =ICELG2(2,ICELL)
      IS   =ICELG2(3,ICELL)
      ISE  =ICELG2(4,ICELL)
      IE   =ICELG2(5,ICELL)
```

```
      INE  =ICELG2(6,ICELL)
      IN   =ICELG2(7,ICELL)
      INW  =ICELG2(8,ICELL)
      IW   =ICELG2(9,ICELL)
C
      IAUX =ICELG2(10,ICELL)
C
C     SAVE DEPENDENT VARIABLES AT ALL CELL CORNERS
C
      USW=WORKG2(ISW)
      USE=WORKG2(ISE)
      UNE=WORKG2(INE)
      UNW=WORKG2(INW)
C
C     SAVE DEPENDENT VARIABLES AT CELL EDGES
C
                  US=0.5*(USW+USE)
      IF(IS.NE.0) US=WORKG2(IS)
                  UE=0.5*(USE+UNE)
      IF(IE.NE.0) UE=WORKG2(IE)
                  UN=0.5*(UNE+UNW)
      IF(IN.NE.0) UN=WORKG2(IN)
                  UW=0.5*(UNW+USW)
      IF(IW.NE.0) UW=WORKG2(IW)
C
                  UC=0.25*(USW+USE+UNE+UNW)
      IF(IC.NE.0) UC=WORKG2(IC)
C
C     COMPUTE CONTRIBUTION TO FIRST DIFFERENCE AT EACH CORNER...
C        ...SOUTHWEST
C
                                       FACTSW=1.00
      IF(IAND(IAUX,HL0001).NE.0     ) FACTSW=2.00
      IF(IAND(IAUX,HL000B).EQ.HL000B) FACTSW=4.00
C
      IF(IAND(IAUX,HL1000).EQ.0) THEN
         CHNGF2(1,ISW)=CHNGF2(1,ISW)+FACTSW*(US -USW)
         CHNGF2(2,ISW)=CHNGF2(2,ISW)+FACTSW*(UW -USW)
      ELSE
         CHNGF2(1,ISW)=CHNGF2(1,ISW)+FACTSW*(USE-USW)
         CHNGF2(2,ISW)=CHNGF2(2,ISW)+FACTSW*(UNW-USW)
      ENDIF
C
C        ...SOUTHEAST
C
                                       FACTSE=1.00
      IF(IAND(IAUX,HL0002).NE.0     ) FACTSE=2.00
      IF(IAND(IAUX,HL0007).EQ.HL0007) FACTSE=4.00
C
      IF(IAND(IAUX,HL2000).EQ.0) THEN
         CHNGF2(1,ISE)=CHNGF2(1,ISE)+FACTSE*(USE-US )
         CHNGF2(2,ISE)=CHNGF2(2,ISE)+FACTSE*(UE -USE)
      ELSE
```

```
            CHNGF2(1,ISE)=CHNGF2(1,ISE)+FACTSE*(USE-USW)
            CHNGF2(2,ISE)=CHNGF2(2,ISE)+FACTSE*(UNE-USE)
         ENDIF
C
C         ...NORTHEAST
C
                                          FACTNE=1.00
         IF(IAND(IAUX,HL0004).NE.0     ) FACTNE=2.00
         IF(IAND(IAUX,HL000E).EQ.HL000E) FACTNE=4.00
C
         IF(IAND(IAUX,HL4000).EQ.0) THEN
            CHNGF2(1,INE)=CHNGF2(1,INE)+FACTNE*(UNE-UN )
            CHNGF2(2,INE)=CHNGF2(2,INE)+FACTNE*(UNE-UE )
         ELSE
            CHNGF2(1,INE)=CHNGF2(1,INE)+FACTNE*(UNE-UNW)
            CHNGF2(2,INE)=CHNGF2(2,INE)+FACTNE*(UNE-USE)
         ENDIF
C
C         ...NORTHWEST
C
                                          FACTNW=1.00
         IF(IAND(IAUX,HL0008).NE.0     ) FACTNW=2.00
         IF(IAND(IAUX,HL000D).EQ.HL000D) FACTNW=4.00
C
         IF(IAND(IAUX,HL8000).EQ.0) THEN
            CHNGF2(1,INW)=CHNGF2(1,INW)+FACTNW*(UN -UNW)
            CHNGF2(2,INW)=CHNGF2(2,INW)+FACTNW*(UNW-UW )
         ELSE
            CHNGF2(1,INW)=CHNGF2(1,INW)+FACTNW*(UNE-UNW)
            CHNGF2(2,INW)=CHNGF2(2,INW)+FACTNW*(UNW-USW)
         ENDIF
C
C     ADD CONTRIBUTIONS IF SIDES NODES EXIST
C
         IF(IS.NE.0) THEN
            CHNGF2(1,IS)=CHNGF2(1,IS)+    (USE-USW)
            CHNGF2(2,IS)=CHNGF2(2,IS)+2.0*(UC -US )
         ENDIF
C
         IF(IE.NE.0) THEN
            CHNGF2(1,IE)=CHNGF2(1,IE)+2.0*(UE -UC )
            CHNGF2(2,IE)=CHNGF2(2,IE)+    (UNE-USE)
         ENDIF
C
         IF(IN.NE.0) THEN
            CHNGF2(1,IN)=CHNGF2(1,IN)+    (UNE-UNW)
            CHNGF2(2,IN)=CHNGF2(2,IN)+2.0*(UN -UC )
         ENDIF
C
         IF(IW.NE.0) THEN
            CHNGF2(1,IW)=CHNGF2(1,IW)+2.0*(UC -UW )
            CHNGF2(2,IW)=CHNGF2(2,IW)+    (UNW-USW)
         ENDIF
```

```
C
20      CONTINUE
C
C     COMBINE X AND Y COMPONENTS AND COMPUTE THE AVREAGE
C        CHANGE FOR ALL THE NODES
C
        AVG=0.0
C
        DO 30 INODE=1,NNODG2
        CHNGF2(1,INODE)=SQRT(CHNGF2(1,INODE)**2+CHNGF2(2,INODE)**2)
        AVG=AVG+CHNGF2(1,INODE)
30      CONTINUE
C
        AVG=AVG/NNODG2
C
C     NORMALIZE THE CHANGES
C
        DO 50 INODE=1,NNODG2
        CHNGF2(1,INODE)=CHNGF2(1,INODE)/AVG
50      CONTINUE
C
        TITLF2='1ST DIFF OF '//TITLG2
C
        RETURN
        END
```

F2INIT

```
        SUBROUTINE F2INIT(IOPT
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS THE CONSTANTS FOR THE BURGER'S EQUATION
C        SOLVER.
C
C        IOPT =0 READ FROM JCARDS AND PRINT
C             =1 READ FROM JTERMI AND PRINT
C             =2 INITIALIZE NFFSF2
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.FEAT2D]F2COMN.INC'
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*****************************************************************************
```

```
C
      IF(IOPT.EQ.1) GOTO 20
      IF(IOPT.EQ.2) GOTO 70
C
C     READ INPUTS FROM INPUT FILE
C
      READ(JCARDS,10) ALFAF2,BETAF2,GAMAF2,DLTAF2,NSNGF2
10    FORMAT(4F10.0,I5)
      IF(NSNGF2.GT.0)
     &   READ(JCARDS,15) (XSNGF2(I),YSNGF2(I),RSNGF2(I),I=1,NSNGF2)
15    FORMAT(3F10.0)
      GOTO 40
C
C     CHANGE INPUTS FROM TERMINAL
C
20    WRITE(JTERMO,30) ALFAF2,
     &                 BETAF2,
     &                 GAMAF2,
     &                 DLTAF2
30    FORMAT(' 1. ALFAF2 =',F10.5/
     &       ' 2. BETAF2 =',F10.5/
     &       ' 3. GAMAF2 =',F10.5/
     &       ' 4. DLTAF2 =',F10.5/)
C
      CALL UTINPI('VARIABLE NUMBER',IVAR)
C
      IF(IVAR.EQ.0) GOTO 40
C
      IF(IVAR.EQ.1) CALL UTINPF('ALFAF2',ALFAF2)
      IF(IVAR.EQ.2) CALL UTINPF('BETAF2',BETAF2)
      IF(IVAR.EQ.3) CALL UTINPF('GAMAF2',GAMAF2)
      IF(IVAR.EQ.4) CALL UTINPF('DLTAF2',DLTAF2)
C
      GOTO 20
C
C     PRINT OUT PARAMETERS
C
40    WRITE(JPRINT,50) ALFAF2,
     &                 BETAF2,
     &                 GAMAF2,
     &                 DLTAF2,
     &                 NSNGF2
50    FORMAT(' 2-D FEATURE FINDER INITIALIZED'/
     &       10X,' ALFAF2 = ',F10.5/
     &       10X,' BETAF2 = ',F10.5/
     &       10X,' GAMAF2 = ',F10.5/
     &       10X,' DLTAF2 = ',F10.5/
     &       10X,' NSNGF2 = ',I5   )
      IF(NSNGF2.GT.0)
     &   WRITE(JPRINT,60) (I,XSNGF2(I),YSNGF2(I),RSNGF2(I),I=1,NSNGF2)
60    FORMAT(I5,'  XSNGF2=',G15.7,'  YSNGF2=',G15.7,'  RSNGF2=',G15.7)
C
C     INITIALIZE THE NUMBER OF PREVIOUS FEATURE FINDER APPLICATIONS
```

```
C
70      NFFSF2=0
C
        RETURN
        END
```

F2LAPL

```
        SUBROUTINE F2LAPL
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CALCULATES THE SECOND DIFFERENCE OF WORKG2 AT
C        EACH NODE AND THEN NORMALIZES THEM BY THE AVERAGE
C
C*****************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.FEAT2D]F2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************
C
C     ZERO OUT THE CHANGE AT EVERY NODE
C
        DO 10 INODE=1,NNODG2
        CHNGF2(1,INODE)=0.0
        CHNGF2(2,INODE)=0.0
10      CONTINUE
C
C     STEP THROUGH EACH FINE CELL TO ACCUMULATE AVERAGE DIFFERENCE
C
        DO 20 ICELL=1,NCELG2
C
        IC   =ICELG2(1,ICELL)
        IF(IC.NE.0) GOTO 20
C
C     SET UP NODE POINTERS FOR THIS CELL
C
        ISW  =ICELG2(2,ICELL)
        IS   =ICELG2(3,ICELL)
        ISE  =ICELG2(4,ICELL)
        IE   =ICELG2(5,ICELL)
        INE  =ICELG2(6,ICELL)
        IN   =ICELG2(7,ICELL)
        INW  =ICELG2(8,ICELL)
        IW   =ICELG2(9,ICELL)
C
```

```
        IAUX =ICELG2(10,ICELL)
C
C     SAVE DEPENDENT VARIABLES AT ALL CELL CORNERS
C
        USW=WORKG2(ISW)
        USE=WORKG2(ISE)
        UNE=WORKG2(INE)
        UNW=WORKG2(INW)
C
C     SAVE DEPENDENT VARIABLES AT CELL EDGES
C
                    US=0.5*(USW+USE)
        IF(IS.NE.0) US=WORKG2(IS)
                    UE=0.5*(USE+UNE)
        IF(IE.NE.0) UE=WORKG2(IE)
                    UN=0.5*(UNE+UNW)
        IF(IN.NE.0) UN=WORKG2(IN)
                    UW=0.5*(UNW+USW)
        IF(IW.NE.0) UW=WORKG2(IW)
C
                    UC=0.25*(USW+USE+UNE+UNW)
        IF(IC.NE.0) UC=WORKG2(IC)
C
C     COMPUTE CONTRIBUTION TO FIRST DIFFERENCE AT EACH CORNER...
C        ...SOUTHWEST
C
                                  FACTSW=1.00
        IF(IAND(IAUX,HL0001).NE.0) FACTSW=0.00
C
        IF(IAND(IAUX,HL1000).EQ.0) THEN
           CHNGF2(1,ISW)=CHNGF2(1,ISW)+FACTSW*(US -USW)
           CHNGF2(2,ISW)=CHNGF2(2,ISW)+FACTSW*(UW -USW)
        ELSE
           CHNGF2(1,ISW)=CHNGF2(1,ISW)+FACTSW*(USE-USW)
           CHNGF2(2,ISW)=CHNGF2(2,ISW)+FACTSW*(UNW-USW)
        ENDIF
C
C        ...SOUTHEAST
C
                                  FACTSE=1.00
        IF(IAND(IAUX,HL0002).NE.0) FACTSE=0.00
C
        IF(IAND(IAUX,HL2000).EQ.0) THEN
           CHNGF2(1,ISE)=CHNGF2(1,ISE)+FACTSE*(US -USE)
           CHNGF2(2,ISE)=CHNGF2(2,ISE)+FACTSE*(UE -USE)
        ELSE
           CHNGF2(1,ISE)=CHNGF2(1,ISE)+FACTSE*(USW-USE)
           CHNGF2(2,ISE)=CHNGF2(2,ISE)+FACTSE*(UNE-USE)
        ENDIF
C
C        ...NORTHEAST
C
                                  FACTNE=1.00
```

```
        IF(IAND(IAUX,HL0004).NE.0) FACTNE=0.00
C
        IF(IAND(IAUX,HL4000).EQ.0) THEN
           CHNGF2(1,INE)=CHNGF2(1,INE)+FACTNE*(UN -UNE)
           CHNGF2(2,INE)=CHNGF2(2,INE)+FACTNE*(UE -UNE)
        ELSE
           CHNGF2(1,INE)=CHNGF2(1,INE)+FACTNE*(UNW-UNE)
           CHNGF2(2,INE)=CHNGF2(2,INE)+FACTNE*(USE-UNE)
        ENDIF
C
C        ...NORTHWEST
C
                                    FACTNW=1.00
        IF(IAND(IAUX,HL0008).NE.0) FACTNW=0.00
C
        IF(IAND(IAUX,HL8000).EQ.0) THEN
           CHNGF2(1,INW)=CHNGF2(1,INW)+FACTNW*(UN -UNW)
           CHNGF2(2,INW)=CHNGF2(2,INW)+FACTNW*(UW -UNW)
        ELSE
           CHNGF2(1,INW)=CHNGF2(1,INW)+FACTNW*(UNE-UNW)
           CHNGF2(2,INW)=CHNGF2(2,INW)+FACTNW*(USW-UNW)
        ENDIF
C
C     ADD CONTRIBUTIONS IF SIDES NODES EXIST
C
        IF(IS.NE.0) THEN
           CHNGF2(1,IS)=CHNGF2(1,IS)+    (USE-2.0*US+USW)
           CHNGF2(2,IS)=CHNGF2(2,IS)+2.0*(UC        -US )
        ENDIF
C
        IF(IE.NE.0) THEN
           CHNGF2(1,IE)=CHNGF2(1,IE)+2.0*(UC        -UE )
           CHNGF2(2,IE)=CHNGF2(2,IE)+    (UNE-2.0*UE+USE)
        ENDIF
C
        IF(IN.NE.0) THEN
           CHNGF2(1,IN)=CHNGF2(1,IN)+    (UNE-2.0*UN+UNW)
           CHNGF2(2,IN)=CHNGF2(2,IN)+2.0*(UC        -UN )
        ENDIF
C
        IF(IW.NE.0) THEN
           CHNGF2(1,IW)=CHNGF2(1,IW)+2.0*(UC        -UW )
           CHNGF2(2,IW)=CHNGF2(2,IW)+    (UNW-2.0*UW+USW)
        ENDIF
C
20      CONTINUE
C
C     COMBINE X AND Y COMPONENTS AND COMPUTE THE AVREAGE
C        CHANGE FOR ALL THE NODES
C
        AVG=0.0
C
        DO 30 INODE=1,NNODG2
```

```
        CHNGF2(1,INODE)=ABS(CHNGF2(1,INODE)+CHNGF2(2,INODE))
        AVG=AVG+CHNGF2(1,INODE)
30      CONTINUE
C
        AVG=AVG/NNODG2
C
C     NORMALIZE THE CHANGES
C
        DO 50 INODE=1,NNODG2
        CHNGF2(1,INODE)=CHNGF2(1,INODE)/AVG
50      CONTINUE
C
        TITLF2='2ND DIFF OF '//TITLG2
C
        RETURN
        END
```

F2PLOT

```
        SUBROUTINE F2PLOT(IOPT,IOPTN
     &
     &                                )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PLOTS:
C        IF IOPT =1    NODES WHICH ARE ABOVE THRDF2 ARE PLOTTED AS
C                      CROSSES, THOSE BELOW THRFF2 AS CIRCLES, AND
C                      OTHERS SIMPLY AS POINTS
C                =2    THRESHOLD FINDER
C                =3    "CHANGE" VS. NODE NUMBER
C                =4    HISTOGRAMS OF THE REFINEMENT PARAMETER AND ITS
C                      CUMMULATIVE DISTRIBUTION ARE PLOTTED
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.FEAT2D]F2COMN.INC'
C
        INCLUDE '[.UTILITY]UTCOMN.INC'
C
        EXTERNAL F2PLTN,F2PLTT,F2PLTH
C
        DIMENSION XPLOT(1000),YPLOT(1000),ILINPL(4),ISYMPL(4),NPTSPL(4)
C
        CHARACTER*80 PLTITL
C
C*********************************************************************
C
```

```
        IF(IOPT.EQ.1) GOTO 100
        IF(IOPT.EQ.2) GOTO 200
        IF(IOPT.EQ.3) GOTO 300
        IF(IOPT.EQ.4) GOTO 400
                      RETURN
C
C*****PLOT OF THRESHOLDED NODES
C
C     FIND MINIMUM AND MAXIMUM COORDINATES FOR SCALING
C
100     XPLOT(1)=GEOMG2(1,1)
        XPLOT(2)=GEOMG2(1,1)
        YPLOT(1)=GEOMG2(2,1)
        YPLOT(2)=GEOMG2(2,1)
C
        DO 110 INODE=2,NNODG2
        IF(GEOMG2(1,INODE).LT.XPLOT(1)) XPLOT(1)=GEOMG2(1,INODE)
        IF(GEOMG2(1,INODE).GT.XPLOT(2)) XPLOT(2)=GEOMG2(1,INODE)
        IF(GEOMG2(2,INODE).LT.YPLOT(1)) YPLOT(1)=GEOMG2(2,INODE)
        IF(GEOMG2(2,INODE).GT.YPLOT(2)) YPLOT(2)=GEOMG2(2,INODE)
110     CONTINUE
C
C     SET UP TITLE
C
        WRITE(PLTITL,120) TITLF2,THRDF2,THRFF2,NNODG2
120     FORMAT('  GEOM1   GEOM2 ',A,'D=',F6.3,'F=',F6.3,5X,'NODES='I6)
C
C     PLOT
C
        INDGR=23+IOPTN
        CALL GRAPHC(F2PLTN,INDGR,PLTITL,XPLOT,YPLOT,2,GRDUMY)
C
        RETURN
C
C*****PLOT OF THRESHOLD FINDER
C
C     GET NUMBER OF POINTS, ETC.
C
200     NPTS=NINT(CHNGF2(2,1))
        IX  =1
        IY  =5+NPTS
C
C     SET UP PLOT ARRAYS
C
        DO 210 I=1,NPTS+4
        XPLOT(I)=CHNGF2(2,I+IX)
        YPLOT(I)=CHNGF2(2,I+IY)
210     CONTINUE
C
C     SET UP TITLE
C
        WRITE(PLTITL,220) TITLF2,THRDF2,THRFF2,NNODG2
220     FORMAT(' THRESH  FRACTN ',A,'D=',F6.3,'F=',F6.3,5X,'NODES='I6)
```

```
C
C     PLOT
C
        INDGR=21+IOPTN
        CALL GRAPHC(F2PLTT,INDGR,PLTITL,XPLOT,YPLOT,NPTS,GRDUMY)
C
        RETURN
C
C*****PLOT CHANGE VS. NODE NUMBER
C
300     CALL UTBASE(+1,NNODG2+5,INODE)
        CALL UTBASE(+1,NNODG2+5,ICHNG)
C
C     SET UP DIVIDE AND FUSE THRESHOLDS (AND A DUMMY LINE)
C
        VARUT(1+INODE)=0.0
        VARUT(2+INODE)=1.0
        VARUT(1+ICHNG)=THRFF2
        VARUT(2+ICHNG)=THRFF2
C
        VARUT(3+INODE)=0.0
        VARUT(4+INODE)=1.0
        VARUT(3+ICHNG)=THRDF2
        VARUT(4+ICHNG)=THRDF2
C
        VARUT(5+INODE)=1.0
        VARUT(5+ICHNG)=THRDF2
C
        ISEED=12345
C
        DO 310 I=1,NNODG2
        VARUT(5+I+INODE)=RAN(ISEED)
        VARUT(5+I+ICHNG)=CHNGF2(1,I)
310     CONTINUE
C
C     SET UP PLOT INDICATORS
C
        ILINPL(1)=1
        ILINPL(2)=1
        ILINPL(3)=1
        ILINPL(4)=0
        ISYMPL(1)=0
        ISYMPL(2)=0
        ISYMPL(3)=0
        ISYMPL(4)=4
        NPTSPL(1)=2
        NPTSPL(2)=2
        NPTSPL(3)=1
        NPTSPL(4)=NNODG2
C
C     SET UP THE TITLE AND PLOT
C
        WRITE(PLTITL,320) TITLF2,NNODG2
```

```
320     FORMAT('        REFN PAR',A,21X,'NODES='I6)
C
        INDGR=21+IOPTN
        CALL GRLINE(ILINPL,ISYMPL,4,PLTITL,INDGR,VARUT(1+INODE),
     &                                    VARUT(1+ICHNG),NPTSPL)
C
C     RELEASE BASED VARIABLES
C
        CALL UTBASE(-1,IDUM,INODE)
        CALL UTBASE(-1,IDUM,ICHNG)
C
        RETURN
C
C*****PLOT OF REFINEMENT PARAMETER HISTOGRAMS
C
C     SET UP BINS FOR THE HISTOGRAMS
C
400     NHIST=100
        DXHST=5.0/NHIST
C
        DO 410 I=1,NHIST
        XPLOT(I)=(I-1)*DXHST
        YPLOT(I)=0.0
410     CONTINUE
C
C     PUT THE REFINEMENT PARAMETERS INTO THE APPROPRIATE BINS
C
        DO 420 INODE=1,NNODG2
        JBIN=MAX(MIN(INT(CHNGF2(1,INODE)/DXHST),NHIST),1)
        YPLOT(JBIN)=YPLOT(JBIN)+1.
420     CONTINUE
C
C     SET UP TITLE
C
        WRITE(PLTITL,430) TITLF2,NNODG2
430     FORMAT('REFN PARNUM HITS',A,5X,'NODES='I6)
C
C     PLOT
C
        INDGR=21+IOPTN
        CALL GRAPHC(F2PLTH,INDGR,PLTITL,XPLOT,YPLOT,NHIST,GRDUMY)
C
C     COMPUTE THE CUMULATIVE FUNCTION
C
        DO 440 I=NHIST-1,1,-1
        YPLOT(I)=YPLOT(I)+YPLOT(I+1)
440     CONTINUE
C
C     SET UP TITLE
C
        WRITE(PLTITL,450) TITLF2,NNODG2
450     FORMAT('REFN PARCUM HITS',A,5X,'NODES='I6)
C
```

```
C     PLOT
C
        INDGR=21+IOPTN
        CALL GRAPHC(F2PLTH,INDGR,PLTITL,XPLOT,YPLOT,NHIST,GRDUMY)
C
        RETURN
C
        END
```

F2PLTH

```
        SUBROUTINE F2PLTH(XPLOT,YPLOT,NPTS,GRDUMY
     &
     &                                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PLOTS THE HISTOGRAM AND CUMULATIVE DISTRIBUTION
C       FUNCTIONS OF THE REFINEMENT PARAMETERS
C
        DIMENSION XPLOT(*),YPLOT(*),GRDUMY(*)
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.FEAT2D]F2COMN.INC'
C
C*************************************************************************
C
C     GET THE WIDTH OF THE WINDOW
C
        XWID=XPLOT(2)-XPLOT(1)
C
C     DRAW EACH BAR OF THE HISTOGRAM
C
        DO 10 IPTS=1,NPTS
C
        CALL GRMOVE(XPLOT(IPTS)     ,0.0        ,0)
        CALL GRDRAW(XPLOT(IPTS)     ,YPLOT(IPTS),0)
        CALL GRDRAW(XPLOT(IPTS)+XWID,YPLOT(IPTS),0)
        CALL GRDRAW(XPLOT(IPTS)+XWID,0.0        ,0)
C
10      CONTINUE
C
        RETURN
        END
```

F2PLTN

```
      SUBROUTINE F2PLTN(XDUM$,YDUM$,NDUM,GRDUMY
     &
     &                                          )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PLOTS THE THRESHOLDED NODES
C
      DIMENSION XDUM$(*),YDUM$(*)
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.FEAT2D]F2COMN.INC'
C
C*********************************************************************
C
C     LOOP THROUGH EACH NODE
C
      DO 10 INODE=1,NNODG2
C
C     DRAW A CROSS IF ABOVE THRDF2, A CIRCLE IF BELOW THRFF2, OR
C       OTHERWISE A POINT
C
                                    ISYM=0
      IF(CHNGF2(1,INODE).GT.THRDF2) ISYM=4
      IF(CHNGF2(1,INODE).LT.THRFF2) ISYM=2
C
      CALL GRMOVE(GEOMG2(1,INODE),GEOMG2(2,INODE),   0)
      CALL GRDRAW(GEOMG2(1,INODE),GEOMG2(2,INODE),ISYM)
C
10    CONTINUE
C
      RETURN
      END
```

F2PLTT

```
      SUBROUTINE F2PLTT(THRSH$,FRACT$,NPTS,GRDUMY
     &
     &                                          )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PLOTS THE THRESHOLD FINDER AS WELL AS THE
C       FOUR COMPUTED THRESHOLDS
```

```
C
      DIMENSION THRSH$(*),FRACT$(*)
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.FEAT2D]F2COMN.INC'
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
      DELX=(XMAXGR-XMINGR)/100.
      DELY=(YMAXGR-YMINGR)/100.
C
C     PLOT THE CURVE
C
      CALL GRMOVE(THRSH$(1),FRACT$(1),0)
C
      DO 10 I=2,NPTS
      CALL GRDRAW(THRSH$(I),FRACT$(I),0)
10    CONTINUE
C
C     PLOT THE FOUR THRESHOLDS
C
C.....ALFA
C
      CALL GRMOVE(THRSH$(NPTS+1),FRACT$(NPTS+1),1)
C
C.....BETA
C
      CALL GRMOVE(THRSH$(NPTS+2),YMINGR,0)
      CALL GRDRAW(THRSH$(NPTS+2),YMAXGR,0)
C
C.....GAMA
C
      CALL GRMOVE(XMINGR,FRACT$(NPTS+3),0)
      CALL GRDRAW(XMAXGR,FRACT$(NPTS+3),0)
C
C     STIPPLE THE RESTRICTED AREA
C
      DO 20 IX=0,100
      DO 20 IY=0,100
C
      XX=XMINGR+IX*DELX
      YY=YMINGR+IY*DELY
C
      IF(XX.GT.THRSH$(NPTS+2) .AND. YY.LT.FRACT$(NPTS+3)) GOTO 20
C
      CALL GRMOVE(XX,YY,0)
      CALL GRDRAW(XX,YY,0)
C
```

```
20      CONTINUE
C
        RETURN
        END
```

F2PRNT

```
        SUBROUTINE F2PRNT
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PRINTS OUT THE VARIABLES IN /F2COMN/
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.FEAT2D]F2COMN.INC'
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
        CALL UTHEAD('FEATURE PROCESSING VARIABLES')
C
        WRITE(JPRINT,10) ALFAF2,BETAF2,GAMAF2,DLTAF2
10      FORMAT(' INPUT PARAMETERS:'/
     &              10X,'ALFAF2=',G15.7,10X,'BETAF2=',G15.7,
     &              10X,'GAMAF2=',G15.7,10X,'DLTAF2=',G15.7)
C
        WRITE(JPRINT,20) THRDF2,THRFF2
20      FORMAT(' COMPUTED THRESHOLDS:'/
     &              10X,'DIVIDE=',G15.7,10X,'FUSE=',G15.7)
C
        DO 60 IBEG=1,NNODG2,10
        IEND=MIN(IBEG+9,NNODG2)
C
        WRITE(JPRINT,30) (          I,I=IBEG,IEND)
30      FORMAT(/'         I= ',10I10)
        WRITE(JPRINT,40) (CHNGF2(1,I),I=IBEG,IEND)
40      FORMAT( ' CHNGF2(1)= ',10F10.5)
        WRITE(JPRINT,50) (CHNGF2(2,I),I=IBEG,IEND)
50      FORMAT( ' CHNGF2(2)= ',10F10.5)
C
60      CONTINUE
C
        RETURN
        END
```

F2THRS

```
      SUBROUTINE F2THRS(
     &
     &                    TDIVD,TFUSE)
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PICKS THE DIVIDE THRESHOLD (THRDF2) AND
C        FUSE THRESHOLD (THRFF2) SUCH THAT:
C        -   THRDF2 = MAX (THRSH1,THRSH2,THRSH3)
C        -   THRFF2 =             THRSH4
C
C*************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.FEAT2D]F2COMN.INC'
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
      PARAMETER (NBIN=201, NSMTH=100)
C
      DIMENSION THRESH(NBIN),FRACTN(NBIN)
C
C*************************************************************************
C
      DTHRSH=5.0/(NBIN-1)
C
C     INITIALIZE THRESH AND FRACTN
C
      DO 10 IBIN=1,NBIN
      THRESH(IBIN)=(IBIN-1)*DTHRSH
      FRACTN(IBIN)=0.0
10    CONTINUE
C
C     ACCUMULATE NUMBER OF POINTS IN EACH THRESHOLD BIN
C
      DO 20 INODE=1,NNODG2
      IBIN=1+INT(CHNGF2(1,INODE)/DTHRSH)
      IF(IBIN.LT.   1) IBIN=   1
      IF(IBIN.GT.NBIN) IBIN=NBIN
      FRACTN(IBIN)=FRACTN(IBIN)+1.0
20    CONTINUE
C
C     CALCULATE ACTUAL FRACTION OF POINTS WITH CHANGE ABOVE THRESH
C
      SUM=0.0
C
      DO 30 IBIN=NBIN,1,-1
      SUM=SUM+FRACTN(IBIN)
```

```
        FRACTN(IBIN)=SUM/NNODG2
30      CONTINUE
C
C     SMOOTH THE FRACTN(THRESH) CURVE TO MAKE DIFFERENCING ACCURATE
C
        CALL UTSMTH(NSMTH,NBIN,THRESH,FRACTN)
C
C     FIND THRSH1...LAST POINT WHERE SLOPE OF FRACTN VS. THRESH IS
C        LESS THAN ALFAF2
C
        THRSH1=0.00
C
        DO 40 IBIN=NBIN-1,2,-1
        DFDT=(FRACTN(IBIN+1)-FRACTN(IBIN-1))
     &      /(THRESH(IBIN+1)-THRESH(IBIN-1))
        IF(DFDT.LT.ALFAF2) THEN
           THRSH1=THRESH(IBIN)
           GOTO 50
        ENDIF
40      CONTINUE
C
C     FIND THRSH2...POINT WHERE THRESH EQUALS BETAF2
C
50      THRSH2=BETAF2
C
C     FIND THRSH3...POINT WHERE FRACTN EQUALS GAMAF2
C
        THRSH3=0.00
C
        DO 60 IBIN=NBIN,1,-1
        IF(FRACTN(IBIN).GT.GAMAF2) THEN
           THRSH3=THRESH(IBIN)
           GOTO 70
        ENDIF
60      CONTINUE
C
C     FIND THRSH4...POINT WHERE THRESH EQUALS DLTAF2
C
70      THRSH4=DLTAF2
C
C     DETERMINE DIVIDE AND FUSE THRESHOLDS AND THE FRACTION
C        OF POINTS AT EACH
C
        THRDF2=MAX(THRSH1,THRSH2,THRSH3)
        THRFF2=           THRSH4
C
        CALL UTINTP(THRESH,FRACTN,NBIN,THRDF2,-1,FRACD,DUM)
        CALL UTINTP(THRESH,FRACTN,NBIN,THRFF2,-1,FRACC,DUM)
C
C     PRINT RESULTS
C
        WRITE(JPRINT,80) TITLF2,
     &                   THRSH1,ALFAF2,
```

```
     &                    THRSH2,
     &                    THRSH3,GAMAF2,
     &                    THRSH4,
     &                    THRDF2,FRACD,
     &                    THRFF2,FRACC
80      FORMAT(' REFINEMENT BASED UPON ',A//
     &         ' THRSH1 = ',F7.3,
     &                    '  (POINT WHERE  DF/DT EXCEEDS ',F7.3,')'/
     &         ' THRSH2 = ',F7.3,
     &                    '  (GIVEN VALUE)'/
     &         ' THRSH3 = ',F7.3,
     &                    '  (POINT WHERE FRACTN EXCEEDS ',F7.3,')'/
     &         ' THRSH4 = ',F7.3,
     &                    '  (GIVEN VALUE)'//
     &         '  DIVIDE  THRESHOLD = ',F7.3,' WITH FRACTN = ',F7.3/
     &         '  FUSE    THRESHOLD = ',F7.3,' WITH FRACTN = ',F7.3)
C
C     INTERPOLATE THE GET FRACTIONAL VALUES AT THRESHOLDS
C
        CALL UTINTP(THRESH,FRACTN,NBIN,THRSH1,-1,FRCTN1,DUM)
        CALL UTINTP(THRESH,FRACTN,NBIN,THRSH2,-1,FRCTN2,DUM)
        CALL UTINTP(THRESH,FRACTN,NBIN,THRSH3,-1,FRCTN3,DUM)
        CALL UTINTP(THRESH,FRACTN,NBIN,THRSH4,-1,FRCTN4,DUM)
C
C     PUT THRESHOLD PLOT INFO INTO CHNGF2(2)
C
        CHNGF2(2,1)=NBIN
C
        IX=1
        IY=5+NBIN
C
        DO 90 IBIN=1,NBIN
        CHNGF2(2,IBIN+IX)=THRESH(IBIN)
        CHNGF2(2,IBIN+IY)=FRACTN(IBIN)
90      CONTINUE
C
        CHNGF2(2,NBIN  +2)=THRSH1
        CHNGF2(2,NBIN*2+6)=FRCTN1
C
        CHNGF2(2,NBIN  +3)=THRSH2
        CHNGF2(2,NBIN*2+7)=FRCTN2
C
        CHNGF2(2,NBIN  +4)=THRSH3
        CHNGF2(2,NBIN*2+8)=FRCTN3
C
        CHNGF2(2,NBIN  +5)=THRSH4
        CHNGF2(2,NBIN*2+9)=FRCTN4
C
C     RETURN THE DIVIDE AND FUSE THRESHOLDS
C
        TDIVD=THRDF2
        TFUSE=THRFF2
C
```

```
        RETURN
        END
```

F2VALU

```
        SUBROUTINE F2VALU
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE USES WORKG2 AS CHNGF2 AT EACH NODE AND THEN
C       NORMALIZES THEM BY THE AVERAGE
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.FEAT2D]F2COMN.INC'
C
C*********************************************************************
C
C     STORE WORKG2 AT EVERY NODE
C
        DO 10 INODE=1,NNODG2
        CHNGF2(1,INODE)=ABS(WORKG2(INODE))
10      CONTINUE
C
C     COMPUTE THE AVERAGE CHANGE FOR ALL THE NODES
C
        AVG=0.0
C
        DO 20 INODE=1,NNODG2
        AVG=AVG+CHNGF2(1,INODE)
20      CONTINUE
C
        AVG=AVG/NNODG2
C
C     NORMALIZE THE CHANGES
C
        DO 40 INODE=1,NNODG2
        CHNGF2(1,INODE)=CHNGF2(1,INODE)/AVG
40      CONTINUE
C
        TITLF2='OTH DIFF OF '//TITLG2
C
        RETURN
        END
```

B.5 GRID2D — Two-Dimensional Grid and Data Structure Manager

G2COMN

```
C-----SCALARS
C     KDEFG2        GRID LEVEL AT WHICH BODY IS DEFINED
C     INBRG2        =1 IF NEIGHBORS ARE IN MEMORY, OTHERWISE =0
C
C-----INDEPENDENT VARIABLE ARRAY
C     GEOMG2(I,INODE)   INODE=1,NNODG2
C           I=1   1ST INDEPENDENT VARIABLE
C            =2   2ND INDEPENDENT VARIABLE
C
C-----DEPENDENT VARIABLE ARRAY
C     DPENG2(I,INODE)   INODE=1,NNODG2
C           I=1   1ST DEPENDENT VARIABLE
C            =2   2ND DEPENDENT VARIABLE
C            =3   3RD DEPENDENT VARIABLE
C            =4   4TH DEPENDENT VARIABLE
C            =5   5TH DEPENDENT VARIABLE
C
C-----CELL ARRAY (POINTS TO NODES)
C     ICELG2(I,ICELL)   ICELL=1,NCELG2
C           I=1   ICENT
C            =2   ISW           INW------IN-------INE
C            =3   IS             |                 |
C            =4   ISE            |                 |
C            =5   IE            IW      ICENT      IE
C            =6   INE            |                 |
C            =7   IN             |                 |
C            =8   INW           ISW------IS-------ISE
C            =9   IW
C            =10  IAUX=Z'X X X X X X X X'
C                     |       | | | |
C                     |       | | | -> BOUNDARY
C                     |       | | ---> JUST OUTSIDE EMBEDDED REGION
C                     |       | -----> JUST INSIDE (COARSE)
C                     |       -------> JUST INSIDE (FINE)
C                     |
C                     ---------------> BIT 8 MARKED FOR DIVISION
C                                      BIT 4 MARKED FOR FUSION
C                                      BIT 2 (NOT USED)
C                                      BIT 1 (NOT USED)
C
C          . .   . .   . .   . .   . +   . +   . +   . +
C        0=. .  1=+ .  2=. +  3=+ +  4=. .  5=+ .  6=. +  7=+ +
C
C          + .   + .   + .   + .   + +   + +   + +   + +
C        8=. .  9=+ .  A=. +  B=+ +  C=. .  D=+ .  E=. +  F=+ +
```

```
C
C          NOTE: (.)=NO,  (+)=YES
C
C-----NEIGHBOR ARRAY
C     NBORG2(I,ICELL)   ICELL=1,NCELG2
C            I=1   LFSW     IF =0 THEN NOT DIVIDED OR COARSE GLOBAL
C                                                  MULTIPLE-GRID LEVEL
C             =2   LSW                      *           *
C             =3   LS                LNW    *     LN    *    LNE
C             =4   LSE                      *      .    *
C             =5   LE           ************************************
C             =6   LNE                      *      .    *
C             =7   LN                LW   ..*.....*..  LE
C             =8   LNW                     *LFSW  .     *
C             =9   LW           ************************************
C                                           *      .     *
C          NOTE: A NEGATIVE ENTRY    LSW    *     LS     *    LSE
C                IMPLIES A NEIGH-           *            *
C                BOR ACROSS A SLIT
C
C-----BOUNDARY ARRAY (INTEGER)
C     IBNDG2(I,IBOUND)   IBOUND=1,NBNDG2
C            I=1   INODE
C             =2   ILEFT (CELL)
C             =3   IRITE (CELL)  (0 IF CORNER)
C             =4   IEDGE  0=ERROR;   1=SW-IN;    2=SE-IN;    3=SOUTH;
C                         4=NE-IN;   5=LE-SLIT;  6=EAST;     7=SE-OUT;
C                         8=NW-IN;   9=WEST;    10=TE-SLIT; 11=SW-OUT;
C                        12=NORTH;  13=NW-OUT;  14=NE-OUT;  15=ERROR
C             =5   ITYPE
C             =6   ILEVEL          |                          |
C                                  |                          |
C                       ILEFT      |    IRITE         ILEFT   | (IRITE=0)
C                                  |                          |
C                       -------INODE-------         -------INODE
C
C
C                  13    12    14                      12
C              12   +-----------+   12            6  +=======
C          13 +-------+           +--------+ 14           3
C             +       8         4         +
C           9 +                           +  6
C             +       1         2         +
C          11 +-------+           +--------+ 7          12
C               3    +-----------+    3             =======+  10
C                   11     3     7                      3
C
C-----BOUNDARY ARRAY(REAL)
C     BONDG2(I,IBOUND)   IBOUND=1,NBNDG2
C            I=1   DGEOM1/DN FOR SEGMENT TO LEFT    (COUNTER-CLOCKWISE
C             =2   DGEOM2/DN FOR SEGMENT TO LEFT     ROTATION AROUND
C             =3   DGEOM1/DN FOR SEGMENT TO RIGHT    DOMAIN SW->SE->NE...)
C             =4   DGEOM2/DN FOR SEGMENT TO RIGHT
```

```
C             =5   DPEN1 (INITIAL)
C             =6   DPEN2 (INITIAL)
C             =7   DPEN3 (INITIAL)
C             =8   DPEN4 (INITIAL)
C             =9   DPEN5 (INITIAL)
C
C-----BODY DEFINITION ARRAY
C     BODYG2(I,IBODY)    IBODY=1,NBODG2
C           I= 1  GEOM1 ON BOUNDARY
C            = 2  GEOM2 ON BOUNDARY
C            = 3  DPEN1
C            = 4  DPEN2
C            = 5  DPEN3
C            = 6  DPEN4
C            = 7  DPEN5
C            = 8  DGEOM1/DN FOR SEGMENT TO LEFT    (COUNTER-CLOCKWISE
C            = 9  DGEOM2/DN FOR SEGMENT TO LEFT     RITATION AROUND
C            =10  DGEOM1/DN FOR SEGMENT TO RIGHT    DOMAIN SW->SE->NE...)
C            =11  DGEOM2/DN FOR SEGMENT TO RIGHT
C
C-----MULTIPLE-GRID LEVEL ARRAY (POINTS TO CELLS)
C     ILVLG2(I,ILEVEL)   ILEVEL=-5,5
C           I=1   ICBEG FINE CELLS NOT JUST OUTSIDE NOT AT BOUNDARY
C            =2   ICEND FINE CELLS NOT HUST OUTSIDE NOT AT BOUNDARY
C            =3   ICBEG ALL OTHER FINE CELLS
C            =4   ICEND ALL OTHER FINE CELLS
C            =5   ICBEG COARSE CELLS
C            =6   ICEND COARSE CELLS
C
C-----WORKING(SCRATCH) ARRAY
C     WORKG2(INODE)      INODE=1,NNODG2
C                 SCRATCH VARIABLE
C     TITLGR      NAME OF VARIABLE CURRENTLY IN WORKG2
C
C
C-----DUMMY ARRAY TO WHICH MULTIPLE EQUIVALENCES ARE MADE
C     DUMMG2(MDUM00)
C                 NOTE: TO WORK PROPERLY, ENSURE THAT:
C                    7*MNODG2.LE.5*MCELG2
C
      PARAMETER (MNODG2=35000,
     &           MCELG2=50000,
     &           MBNDG2=1000,
     &           MBODG2=2000,
     &           MLVLG2=10,
     &           MDUM00=2*MNODG2+9*MCELG2)
C
      COMMON /G2COMN/ NNODG2,NCELG2,NBNDG2,NBODG2,KDEFG2,INBRG2,
     &                GEOMG2( 2,MNODG2),DPENG2( 5,MNODG2),
     &                ICELG2(10,MCELG2),
     &                IBNDG2( 6,MBNDG2),BONDG2( 9,MBNDG2),
     &                BODYG2(11,MBODG2),
     &                ILVLG2( 6,-MLVLG2:MLVLG2+1),
```

```
     &                  WORKG2( 2*MNODG2),
     &                  DUMMG2(   MDUM00)
C
        DIMENSION NBORG2(9,MCELG2)
        EQUIVALENCE(DUMMG2(1),NBORG2(1,1))
C
        CHARACTER*8 TITLG2
C
        COMMON /G2CHAR/ TITLG2
```

G2ADPT

```
        SUBROUTINE G2ADPT(
     &
     &                    NCHANG)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE FUSES OR DIVIDES CELLS DEPENDING ON MARKINGS
C        LEFT IN ICELG2(10,ICELL) BY F2ADPT
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
        NCHANG=0
C
C     LOOP THROUGH ALL CELLS ON ALL FINE MULTIPLE-GRID-LEVELS
C
        DO 10 ICELL=ILVLG2(6,MLVLG2-1),ILVLG2(1,0),-1
        IF(ICELG2(2,ICELL).EQ.0) GOTO 10
C
C     DIVIDE CELL IF SO MARKED (TOP BIT SET)
C
        IF(IAND(ICELG2(10,ICELL),HU8000).NE.0) THEN
           ICELG2(10,ICELL)=IAND(ICELG2(10,ICELL),HU7FFF)
           CALL G2DIVD(ICELL,ICHANG)
           NCHANG=NCHANG+ICHANG
C
C     FUSE CELL IF SO MARKED (NEXT TO TOP BIT SET)
C
        ELSEIF(IAND(ICELG2(10,ICELL),HU4000).NE.0) THEN
           ICELG2(10,ICELL)=IAND(ICELG2(10,ICELL),HUBFFF)
           CALL G2FUSE(ICELL,ICHANG)
           NCHANG=NCHANG+ICHANG
C
```

```
        ENDIF
C
10      CONTINUE
C
C     ENSURE THAT THERE ARE NO VOIDS IN THIS NEW GRID
C
        CALL G2VOID(ICHANG)
        NCHANG=NCHANG+ICHANG
C
C     GARBAGE COLLECT
C
        CALL G2GRBG
C
        RETURN
        END
```

G2BOND

```
        SUBROUTINE G2BOND(IBEG,IEND,ITYPE
     &
     &                                  )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE FILLS IN SPLINE INFORMATION FOR THE STRING OF
C        BODY POINTS, STARTING AT IBEG AND GOING TO IEND.
C
C     IF ITYPE=0 PERIODIC END CONDITIONS
C             =1 VANISHING SECOND DERIVATIVE AT ENDS
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
C*********************************************************************
C
        NPTS=IEND-IBEG+1
C
C     SET UP THE X-COORDINATES FOR SPLINING
C
        DO 10 I=1,NPTS
        WOREG2(I)=BODYG2(1,I+IBEG-1)
10      CONTINUE
C
C     CALCULATE THE X-SPLINE COEFFICIENTS
C
        NN=NPTS+ITYPE-1
        CALL UTSPLN(WOREG2(1),NN,ITYPE,WORRG2(NPTS+1))
C
C     STORE THE X-SPLINE INFORMATION
```

```
C
        DO 20 I=2,NPTS-1
        BODYG2( 8,I+IBEG-1)=WORKG2(NPTS+I)
        BODYG2(10,I+IBEG-1)=WORKG2(NPTS+I)
20      CONTINUE
C
        IF(ITYPE.EQ.0) THEN
           BODYG2( 8,IBEG)=WORKG2(NPTS+1)
           BODYG2(10,IBEG)=WORKG2(NPTS+1)
           BODYG2( 8,IEND)=WORKG2(NPTS+1)
           BODYG2(10,IEND)=WORKG2(NPTS+1)
        ELSE
           BODYG2(10,IBEG)=WORKG2(NPTS+   1)
           BODYG2( 8,IEND)=WORKG2(NPTS+NPTS)
        ENDIF
C
C     SET UP THE Y-COORDINATES FOR SPLINING
C
        DO 30 I=1,NPTS
        WORKG2(I)=BODYG2(2,I+IBEG-1)
30      CONTINUE
C
C     CALCULATE THE Y-SPLINE COEFFICIENTS
C
        NN=NPTS+ITYPE-1
        CALL UTSPLN(WORKG2(1),NN,ITYPE,WORKG2(NPTS+1))
C
C     STORE THE Y-SPLINE INFORMATION
C
        DO 40 I=2,NPTS-1
        BODYG2( 9,I+IBEG-1)=WORKG2(NPTS+I)
        BODYG2(11,I+IBEG-1)=WORKG2(NPTS+I)
40      CONTINUE
C
        IF(ITYPE.EQ.0) THEN
           BODYG2( 9,IBEG)=WORKG2(NPTS+1)
           BODYG2(11,IBEG)=WORKG2(NPTS+1)
           BODYG2( 9,IEND)=WORKG2(NPTS+1)
           BODYG2(11,IEND)=WORKG2(NPTS+1)
        ELSE
           BODYG2(11,IBEG)=WORKG2(NPTS+   1)
           BODYG2( 9,IEND)=WORKG2(NPTS+NPTS)
        ENDIF
C
        RETURN
        END
```

G2CHEK

```
        SUBROUTINE G2CHEK(IUNIT,
```

```
     &
     &                    IER  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CHECKS THE CURRENT GRID, FINDING EACH CELLS AREA
C       AND DETERMINING IF THE CELL IS CONVEX.  OUTPUT IS WRITTEN TO
C       UNIT IUNIT
C
C*****************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
C*****************************************************************
C
      IER=0
C
      AREAMN=10000.0
      AREAMX=    0.0
C
C     LOOP THROUGH ALL FINE CELLS
C
      DO 70  ICELL=1,NCELG2
      IF(ICELG2(1,ICELL).NE.0) GOTO 70
      JER=0
C
      ISW=ICELG2(2,ICELL)
      ISE=ICELG2(4,ICELL)
      INE=ICELG2(6,ICELL)
      INW=ICELG2(8,ICELL)
C
C     COMPUTE THE CELL AREA
C
      AREA=0.50*((GEOMG2(1,INE)-GEOMG2(1,ISW))
     &          *(GEOMG2(2,INW)-GEOMG2(2,ISE))
     &          -(GEOMG2(1,INW)-GEOMG2(1,ISE))
     &          *(GEOMG2(2,INE)-GEOMG2(2,ISW)))
C
      AREAMN=MIN(AREAMN,AREA)
      AREAMX=MAX(AREAMX,AREA)
C
      IF(AREA.LE.0.00) THEN
        WRITE(IUNIT,10) AREA
10      FORMAT(' CELL AREA IS NEGATIVE, AREA=',G15.7)
        JER=JER+1
      ENDIF
C
C     DETERMINE IF THE CELL IS CONVEX
C
      CHECK1=(GEOMG2(1,INE)-GEOMG2(1,ISW))
     &       *(GEOMG2(2,INW)-GEOMG2(2,ISW))
     &       -(GEOMG2(1,INW)-GEOMG2(1,ISW))
     &       *(GEOMG2(2,INE)-GEOMG2(2,ISW))
```

```
      IF(CHECK1.LE.0) THEN
         WRITE(IUNIT,20) CHECK1
20       FORMAT(' NODE NW IS BELOW (SW-NE) DIAGONAL, CHECK1=',G15.7)
         JER=JER+1
      ENDIF
C
      CHECK2=(GEOMG2(1,INE)-GEOMG2(1,ISW))
     &       *(GEOMG2(2,ISE)-GEOMG2(2,ISW))
     &       -(GEOMG2(1,ISE)-GEOMG2(1,ISW))
     &       *(GEOMG2(2,INE)-GEOMG2(2,ISW))
      IF(CHECK2.GE.0) THEN
         WRITE(IUNIT,30) CHECK2
30       FORMAT(' NODE SE IS ABOVE (SW-NE) DIAGONAL, CHECK2=',G15.7)
         JER=JER+1
      ENDIF
C
      CHECK3=(GEOMG2(1,INW)-GEOMG2(1,ISE))
     &       *(GEOMG2(2,ISW)-GEOMG2(2,ISE))
     &       -(GEOMG2(1,ISW)-GEOMG2(1,ISE))
     &       *(GEOMG2(2,INW)-GEOMG2(2,ISE))
      IF(CHECK3.LE.0) THEN
         WRITE(IUNIT,40) CHECK3
40       FORMAT(' NODE SW IS BELOW (SE-NW) DIAGONAL, CHECK3=',G15.7)
         JER=JER+1
      ENDIF
C
      CHECK4=(GEOMG2(1,INW)-GEOMG2(1,ISE))
     &       *(GEOMG2(2,INE)-GEOMG2(2,ISE))
     &       -(GEOMG2(1,INE)-GEOMG2(1,ISE))
     &       *(GEOMG2(2,INW)-GEOMG2(2,ISE))
      IF(CHECK4.GE.0) THEN
         WRITE(IUNIT,50) CHECK4
50       FORMAT(' NODE NE IS ABOVE (SE-NW) DIAGONAL, CHECK4=',G15.7)
         JER=JER+1
      ENDIF
C
C     WRITE OUT ...   INFORMATION IF AN ERROR WAS ENCOUNTERED
C
      IF(JER.NE.0) THEN
         WRITE(IUNIT,60) ICELL,
     &                   ISW,GEOMG2(1,ISW),GEOMG2(2,ISW),
     &                   ISE,GEOMG2(1,ISE),GEOMG2(2,ISE),
     &                   INE,GEOMG2(1,INE),GEOMG2(2,INE),
     &                   INW,GEOMG2(1,INW),GEOMG2(2,INW)
60       FORMAT(' ERROR IN CELL',I5   /
     &          '     SW=>',I5,2G15.7 /
     &          '     SE=>',I5,2G15.7 /
     &          '     NE=>',I5,2G15.7 /
     &          '     NW=>',I5,2G15.7 )
         IER=IER+JER
      ENDIF
C
70    CONTINUE
```

```
C
C     WRITE OUT MINIMUM AND MAXIMUM AREAS
C
        WRITE(IUNIT,80) IER,AREAMN,AREAMX
80      FORMAT(' NUMBER OR GRID ERRORS=',I5/
     &         ' MINIMUM CELL AREA=',G15.7 /
     &         ' MAXIMUM CELL AREA=',G15.7 )
C
        RETURN
        END
```

G2DATA

```
        SUBROUTINE G2DATA(IUNIT
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE COMPUTES VARIOUS NORMS OF WORKG2 AND WRITES THE
C       RESULTS ON IUNIT
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
C*****************************************************************************
C
        ANMAX =0.00
        ANMAXM=0.00
        ANMIN =1.00E+20
        ANMINM=1.00E+20
        ANAVG =0.00
        ANAVGM=0.00
        ANRMS =0.00
C
        DO 10 INODE=1,NNODG2
        ANMAX =MAX(ANMAX ,    WORKG2(INODE) )
        ANMAXM=MAX(ANMAXM,ABS(WORKG2(INODE)))
        ANMIN =MIN(ANMIN ,    WORKG2(INODE) )
        ANMINM=MIN(ANMINM,ABS(WORKG2(INODE)))
        ANAVG =ANAVG +    WORKG2(INODE)
        ANAVGM=ANAVGM+ABS(WORKG2(INODE))
        ANRMS =ANRMS +WORKG2(INODE)*WORKG2(INODE)
10      CONTINUE
C
        ANAVG =     ANAVG /NNODG2
        ANAVGM=     ANAVGM/NNODG2
        ANRMS =SQRT(ANRMS /NNODG2)
C
```

```
      WRITE(IUNIT,20) TITLG2,
     &                ANMAX,ANMAXM,
     &                ANMIN,ANMINM,
     &                ANAVG,ANAVGM,
     &                ANRMS
20    FORMAT(' STATISTICS FOR ',A      /
     &       '          MAXIMUM:',G13.7,10X,'MAX-MAG:',G13.7/
     &       '          MIMIMUM:',G13.7,10X,'MIN-MAG:',G13.7/
     &       '          AVERAGE:',G13.7,10X,'AVG-MAG:',G13.7/
     &       '          RMS    :',G13.7                     )
C
      RETURN
      END
```

G2DIVD

```
      SUBROUTINE G2DIVD(LCELL,
     &
     &                  ICHNG)
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE DIVIDES CELL 'LCELL' INTO FOUR SMALLER CELLS
C       AND PERFORMS ALL NECESSARY POINTER SYSTEM REALIGNMENTS
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
      ICHNG=0
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
      IF(INBRG2.EQ.0) THEN
        CALL UTEROR(+1,REAL(INBRG2),0.0)
      ENDIF
C
C     CHECK FOR OVERFLOW IN NODE ARRAYS
C
      IF(NNODG2+5.GT.MNODG2) THEN
        CALL UTEROR(+2,REAL(NNODG2),REAL(MNODG2))
      ENDIF
C
C     CHECK FOR OVERFLOW IN BOUNDARY CONDITION ARRAY
C
      IF(NBNDG2+2.GT.MBNDG2) THEN
```

```
          CALL UTEROR(+3,REAL(NBNDG2),REAL(MBNDG2))
        ENDIF
C
C     SEARCH ALL PERTINENT LEVELS FOR LCELL
C
        DO 40 ILEVEL=0,MLVLG2-1
        KLEVEL=ILEVEL
        IF(LCELL.GE.ILVLG2(1,KLEVEL) .AND.
     &     LCELL.LE.ILVLG2(6,KLEVEL)      ) GOTO 50
40      CONTINUE
C
C     ERROR EXIT 4 (CELL NOT FOUND ON A LEVEL WHICH IS VALID
C        FOR DIVISION)
C
        CALL UTEROR(+4,REAL(LCELL),0.0)
C
C     PICK UP POINTERS TO NODES
C
50      KCENT=ICELG2( 1,LCELL)
        KSW  =ICELG2( 2,LCELL)
        KS   =ICELG2( 3,LCELL)
        KSE  =ICELG2( 4,LCELL)
        KE   =ICELG2( 5,LCELL)
        KNE  =ICELG2( 6,LCELL)
        KN   =ICELG2( 7,LCELL)
        KNW  =ICELG2( 8,LCELL)
        KW   =ICELG2( 9,LCELL)
        KAUX =ICELG2(10,LCELL)
C
C     HAS CELL BEEN PREVIOUSLY DIVIDED OR FUSED OR IS IT JUST
C        INSIDE AN EMBEDDED MESH
C
        IF(KCENT.NE.0 .OR. KSW.EQ.0 .OR. IAND(KAUX,HLF000).NE.0) RETURN
C
C     EXPAND THE CELL TABLE IF THERE IS NOT ENOUGH ROOM FOR THE
C        NEW CELLS
C
        IF((ILVLG2(6,KLEVEL+1)+4).GE.ILVLG2(1,KLEVEL+2))
     &                                  CALL G2XPND((KLEVEL+1),500)
C
C     FIND COARSE CELLS WHICH BOUND DIVIDED CELL (LL INDICATES
C        NEIGHBOR ACROSS A SLIT)
C
C                    *            *
C            LNW     *     LN     *    LNE
C                    *            *
C        ************************************
C                    *            *
C            LW      *   LCELL    *    LE
C                    *            *
C        ************************************
C                    *            *
C            LSW     *     LS     *    LSE
```

```
C                    *            *
C
        LSW=    NBORG2(2,LCELL)
        LLS=ABS(NBORG2(3,LCELL))
        LSE=    NBORG2(4,LCELL)
        LLE=ABS(NBORG2(5,LCELL))
        LNE=    NBORG2(6,LCELL)
        LLN=ABS(NBORG2(7,LCELL))
        LNW=    NBORG2(8,LCELL)
        LLW=ABS(NBORG2(9,LCELL))
C
        LS=MAX(NBORG2(3,LCELL),0)
        LE=MAX(NBORG2(5,LCELL),0)
        LN=MAX(NBORG2(7,LCELL),0)
        LW=MAX(NBORG2(9,LCELL),0)
C
C     FIND FINE CELLS WHICH BOUND DIVIDED CELL
C
C                    *            *
C                    *            *
C            LNWF    * LNFW  LNFE *   LNEF
C        ************************************
C            LWFN    * LFNW  LFNE *   LEFN
C                    *            *
C            LWFS    * LFSW  LFSE *   LEFS
C        ************************************
C            LSWF    * LSFW  LSFE *   LSEF
C                    *            *
C                    *            *
C
        LSWF =0
        LLSFW=0
        LLSFE=0
        LSEF =0
        LLEFS=0
        LLEFN=0
        LNEF =0
        LLNFE=0
        LLNFW=0
        LNWF =0
        LLWFN=0
        LLWFS=0
C
        IF(LSW.NE.0 .AND. NBORG2(1,LSW).NE.0)
     &                                  LSWF =NBORG2(6,NBORG2(1,LSW))
        IF(LLS.NE.0 .AND. NBORG2(1,LLS).NE.0)
     &                                  LLSFW=NBORG2(7,NBORG2(1,LLS))
        IF(LLS.NE.0 .AND. NBORG2(1,LLS).NE.0)
     &                                  LLSFE=NBORG2(6,NBORG2(1,LLS))
        IF(LSE.NE.0 .AND. NBORG2(1,LSE).NE.0)
     &                                  LSEF =NBORG2(7,NBORG2(1,LSE))
        IF(LLE.NE.0 .AND. NBORG2(1,LLE).NE.0)
     &                                  LLEFS=          NBORG2(1,LLE)
```

```
      IF(LLE.NE.0 .AND. NBORG2(1,LLE).NE.0)
     &                                LLEFN=NBORG2(7,NBORG2(1,LLE))
      IF(LNE.NE.0 .AND. NBORG2(1,LNE).NE.0)
     &                                LNEF =          NBORG2(1,LNE)
      IF(LLN.NE.0 .AND. NBORG2(1,LLN).NE.0)
     &                                LLNFE=NBORG2(5,NBORG2(1,LLN))
      IF(LLN.NE.0 .AND. NBORG2(1,LLN).NE.0)
     &                                LLNFW=          NBORG2(1,LLN)
      IF(LNW.NE.0 .AND. NBORG2(1,LNW).NE.0)
     &                                LNWF =NBORG2(5,NBORG2(1,LNW))
      IF(LLW.NE.0 .AND. NBORG2(1,LLW).NE.0)
     &                                LLWFN=NBORG2(6,NBORG2(1,LLW))
      IF(LLW.NE.0 .AND. NBORG2(1,LLW).NE.0)
     &                                LLWFS=NBORG2(5,NBORG2(1,LLW))
C
      IF(LS.EQ.LLS) THEN
         LSFW=LLSFW
         LSFE=LLSFE
      ELSE
         LSFW=0
         LSFE=0
         IF(LLSFW.NE.0 .AND. ICELG2(8,LLSFW).NE.KSW) LLSFW=0
         IF(LLSFE.NE.0 .AND. ICELG2(6,LLSFE).NE.KSE) LLSFE=0
      ENDIF
C
      IF(LE.EQ.LLE) THEN
         LEFS=LLEFS
         LEFN=LLEFN
      ELSE
         LEFS=0
         LEFN=0
         IF(LLEFS.NE.0 .AND. ICELG2(2,LLEFS).NE.KSE) LLEFS=0
         IF(LLEFN.NE.0 .AND. ICELG2(8,LLEFN).NE.KNE) LLEFN=0
      ENDIF
C
      IF(LN.EQ.LLN) THEN
         LNFE=LLNFE
         LNFW=LLNFW
      ELSE
         LNFE=0
         LNFW=0
         IF(LLNFE.NE.0 .AND. ICELG2(4,LLNFE).NE.KNE) LLNFE=0
         IF(LLNFW.NE.0 .AND. ICELG2(2,LLNFW).NE.KNW) LLNFW=0
      ENDIF
C
      IF(LW.EQ.LLW) THEN
         LWFN=LLWFN
         LWFS=LLWFS
      ELSE
         LWFN=0
         LWFS=0
         IF(LLWFN.NE.0 .AND. ICELG2(6,LLWFN).NE.KNW) LLWFN=0
         IF(LLWFS.NE.0 .AND. ICELG2(4,LLWFS).NE.KSW) LLWFS=0
```

```
        ENDIF
C
C     CREATE NODE AT CENTER OF CELL
C
        NNODG2=NNODG2+1
        KCENT=NNODG2
C
        GEOMG2(1,KCENT)=0.25*(GEOMG2(1,KSW)+GEOMG2(1,KSE)
     &                       +GEOMG2(1,KNE)+GEOMG2(1,KNW))
        GEOMG2(2,KCENT)=0.25*(GEOMG2(2,KSW)+GEOMG2(2,KSE)
     &                       +GEOMG2(2,KNE)+GEOMG2(2,KNW))
C
        DPENG2(1,KCENT)=0.25*(DPENG2(1,KSW)+DPENG2(1,KSE)
     &                       +DPENG2(1,KNE)+DPENG2(1,KNW))
        DPENG2(2,KCENT)=0.25*(DPENG2(2,KSW)+DPENG2(2,KSE)
     &                       +DPENG2(2,KNE)+DPENG2(2,KNW))
        DPENG2(3,KCENT)=0.25*(DPENG2(3,KSW)+DPENG2(3,KSE)
     &                       +DPENG2(3,KNE)+DPENG2(3,KNW))
        DPENG2(4,KCENT)=0.25*(DPENG2(4,KSW)+DPENG2(4,KSE)
     &                       +DPENG2(4,KNE)+DPENG2(4,KNW))
        DPENG2(5,KCENT)=0.25*(DPENG2(5,KSW)+DPENG2(5,KSE)
     &                       +DPENG2(5,KNE)+DPENG2(5,KNW))
C
C     CREATE SOUTHERN NODE IF IT DOESN'T EXIST
C
        IF(KS.EQ.0) THEN
          NNODG2=NNODG2+1
          KS=NNODG2
C
          GEOMG2(1,KS)=0.5*(GEOMG2(1,KSW)+GEOMG2(1,KSE))
          GEOMG2(2,KS)=0.5*(GEOMG2(2,KSW)+GEOMG2(2,KSE))
C
          DPENG2(1,KS)=0.5*(DPENG2(1,KSW)+DPENG2(1,KSE))
          DPENG2(2,KS)=0.5*(DPENG2(2,KSW)+DPENG2(2,KSE))
          DPENG2(3,KS)=0.5*(DPENG2(3,KSW)+DPENG2(3,KSE))
          DPENG2(4,KS)=0.5*(DPENG2(4,KSW)+DPENG2(4,KSE))
          DPENG2(5,KS)=0.5*(DPENG2(5,KSW)+DPENG2(5,KSE))
        ENDIF
C
C     CREATE EASTERN NODE IF IT DOESN'T EXIST
C
        IF(KE.EQ.0) THEN
          NNODG2=NNODG2+1
          KE=NNODG2
C
          GEOMG2(1,KE)=0.5*(GEOMG2(1,KSE)+GEOMG2(1,KNE))
          GEOMG2(2,KE)=0.5*(GEOMG2(2,KSE)+GEOMG2(2,KNE))
C
          DPENG2(1,KE)=0.5*(DPENG2(1,KSE)+DPENG2(1,KNE))
          DPENG2(2,KE)=0.5*(DPENG2(2,KSE)+DPENG2(2,KNE))
          DPENG2(3,KE)=0.5*(DPENG2(3,KSE)+DPENG2(3,KNE))
          DPENG2(4,KE)=0.5*(DPENG2(4,KSE)+DPENG2(4,KNE))
          DPENG2(5,KE)=0.5*(DPENG2(5,KSE)+DPENG2(5,KNE))
```

```
        ENDIF
C
C     CREATE NORTHERN NODE IF IT DOESN'T EXIST
C
        IF(KN.EQ.0) THEN
           NNODG2=NNODG2+1
           KN=NNODG2
C
           GEOMG2(1,KN)=0.5*(GEOMG2(1,KNE)+GEOMG2(1,KNW))
           GEOMG2(2,KN)=0.5*(GEOMG2(2,KNE)+GEOMG2(2,KNW))
C
           DPENG2(1,KN)=0.5*(DPENG2(1,KNE)+DPENG2(1,KNW))
           DPENG2(2,KN)=0.5*(DPENG2(2,KNE)+DPENG2(2,KNW))
           DPENG2(3,KN)=0.5*(DPENG2(3,KNE)+DPENG2(3,KNW))
           DPENG2(4,KN)=0.5*(DPENG2(4,KNE)+DPENG2(4,KNW))
           DPENG2(5,KN)=0.5*(DPENG2(5,KNE)+DPENG2(5,KNW))
        ENDIF
C
C     CREATE WESTERN NODE IF IT DOESN'T EXIST
C
        IF(KW.EQ.0) THEN
           NNODG2=NNODG2+1
           KW=NNODG2
C
           GEOMG2(1,KW)=0.5*(GEOMG2(1,KNW)+GEOMG2(1,KSW))
           GEOMG2(2,KW)=0.5*(GEOMG2(2,KNW)+GEOMG2(2,KSW))
C
           DPENG2(1,KW)=0.5*(DPENG2(1,KNW)+DPENG2(1,KSW))
           DPENG2(2,KW)=0.5*(DPENG2(2,KNW)+DPENG2(2,KSW))
           DPENG2(3,KW)=0.5*(DPENG2(3,KNW)+DPENG2(3,KSW))
           DPENG2(4,KW)=0.5*(DPENG2(4,KNW)+DPENG2(4,KSW))
           DPENG2(5,KW)=0.5*(DPENG2(5,KNW)+DPENG2(5,KSW))
        ENDIF
C
C     INCREMENT THE TOTAL NUMBER OF CELLS AND THE MULTIPLE-GRID-LEVEL
C        ARRAY
C
        NCELG2=NCELG2+4
C
        ILVLG2(6,KLEVEL+1)=ILVLG2(6,KLEVEL+1)+4
C
C     UPDATE THE DIVIDED CELL
C
        ICELG2(1,LCELL)=KCENT
        ICELG2(3,LCELL)=KS
        ICELG2(5,LCELL)=KE
        ICELG2(7,LCELL)=KN
        ICELG2(9,LCELL)=KW
C
C     CREATE THE NEW CELLS
C
        LFSW=ILVLG2(6,KLEVEL+1)-3
        LFSE=LFSW+1
```

```
      LFNE=LFSE+1
      LFNW=LFNE+1
C
      ICELG2( 1,LFSW)=0
      ICELG2( 2,LFSW)=KSW
      ICELG2( 3,LFSW)=0
      ICELG2( 4,LFSW)=KS
      ICELG2( 5,LFSW)=0
      ICELG2( 6,LFSW)=KCENT
      ICELG2( 7,LFSW)=0
      ICELG2( 8,LFSW)=KW
      ICELG2( 9,LFSW)=0
      ICELG2(10,LFSW)=0
C
      ICELG2( 1,LFSE)=0
      ICELG2( 2,LFSE)=KS
      ICELG2( 3,LFSE)=0
      ICELG2( 4,LFSE)=KSE
      ICELG2( 5,LFSE)=0
      ICELG2( 6,LFSE)=KE
      ICELG2( 7,LFSE)=0
      ICELG2( 8,LFSE)=KCENT
      ICELG2( 9,LFSE)=0
      ICELG2(10,LFSE)=0
C
      ICELG2( 1,LFNE)=0
      ICELG2( 2,LFNE)=KCENT
      ICELG2( 3,LFNE)=0
      ICELG2( 4,LFNE)=KE
      ICELG2( 5,LFNE)=0
      ICELG2( 6,LFNE)=KNE
      ICELG2( 7,LFNE)=0
      ICELG2( 8,LFNE)=KN
      ICELG2( 9,LFNE)=0
      ICELG2(10,LFNE)=0
C
      ICELG2( 1,LFNW)=0
      ICELG2( 2,LFNW)=KW
      ICELG2( 3,LFNW)=0
      ICELG2( 4,LFNW)=KCENT
      ICELG2( 5,LFNW)=0
      ICELG2( 6,LFNW)=KN
      ICELG2( 7,LFNW)=0
      ICELG2( 8,LFNW)=KNW
      ICELG2( 9,LFNW)=0
      ICELG2(10,LFNW)=0
C
C     SET EDGE NODE POINTERS OF ALL NEIGHBORING CELLS
C
      IF(LS.NE.0) ICELG2(7,LS)=KS
      IF(LE.NE 0) ICELG2(9,LE)=KE
      IF(LN.NE.0) ICELG2(3,LN)=KN
      IF(LW.NE.0) ICELG2(5,LW)=KW
```

```
C
C     SKIP NEXT SECTION OF CODE IF LCELL IS NOT A BOUNDARY CELL
C
        IF(IAND(KAUX,HLOOOF).EQ.0) GOTO 320
C
C     SCAN THROUGH ALL BOUNDARY CONDITION POINTERS, LOOKING FOR POINTERS
C       TO THE DIVIDED CELL
C
        ICORN=0
        ILEFT=0
        IRITE=0
C
        DO 180 IBND=1,NBNDG2
        IF(IBNDG2(2,IBND).EQ.LCELL .AND. IBNDG2(3,IBND).EQ.0) ICORN=IBND
        IF(IBNDG2(2,IBND).EQ.LCELL .AND. IBNDG2(3,IBND).NE.0) ILEFT=IBND
        IF(IBNDG2(3,IBND).EQ.LCELL                          ) IRITE=IBND
180     CONTINUE
C
        IBODSW=0
        IBODSE=0
        IBODNE=0
        IBODNW=0
C
        DO 185 IBOD=1,NBODG2
        IF(ABS((GEOMG2(1,KSW)-BODYG2(1,IBOD))**2
     &       +(GEOMG2(2,KSW)-BODYG2(2,IBOD))**2).EQ.0.00) IBODSW=IBOD
        IF(ABS((GEOMG2(1,KSE)-BODYG2(1,IBOD))**2
     &       +(GEOMG2(2,KSE)-BODYG2(2,IBOD))**2).EQ.0.00) IBODSE=IBOD
        IF(ABS((GEOMG2(1,KNE)-BODYG2(1,IBOD))**2
     &       +(GEOMG2(2,KNE)-BODYG2(2,IBOD))**2).EQ.0.00) IBODNE=IBOD
        IF(ABS((GEOMG2(1,KNW)-BODYG2(1,IBOD))**2
     &       +(GEOMG2(2,KNW)-BODYG2(2,IBOD))**2).EQ.0.00) IBODNW=IBOD
185     CONTINUE
C
C     BRANCH OUT DEPENDING ON BOUNDARY CONDITION TYPE
C
        GOTO (310,270,280,200,290,310,220,210,
     &       300,260,310,190,240,250,230,310), (IAND(KAUX,HLOOOF)+1)
        GOTO 310
C
C       ...DIVIDED CELL WAS SOUTHWESTERN CORNER (OUTSIDE)
C
190     IF(ILEFT.EQ.0 .OR. ICORN.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
        IBNDG2(2,ICORN)=LFSW
        IBNDG2(2,ILEFT)=LFSE
        IBNDG2(3,IRITE)=LFNW
C
        NBNDG2=NBNDG2+1
C
        IF((KLEVEL+1).GT.KDEFG2) THEN
           H=0.50**(KLEVEL-KDEFG2)
           GEOMG2(1,KS)=GEOMG2(1,KS)
```

```
     &                +0.125*H*(BONDG2(3,ICORN)-BONDG2(1,ILEFT))
          GEOMG2(2,KS)=GEOMG2(2,KS)
     &                +0.125*H*(BONDG2(4,ICORN)-BONDG2(2,ILEFT))
C
          BONDG2(1,NBNDG2)=(GEOMG2(1,KSE)-GEOMG2(1,KSW))/H
          BONDG2(2,NBNDG2)=(GEOMG2(2,KSE)-GEOMG2(2,KSW))/H
          BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
          BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
          BONDG2(5,NBNDG2)=0.50*(BONDG2(5,ICORN)+BONDG2(5,ILEFT))
          BONDG2(6,NBNDG2)=0.50*(BONDG2(6,ICORN)+BONDG2(6,ILEFT))
          BONDG2(7,NBNDG2)=0.50*(BONDG2(7,ICORN)+BONDG2(7,ILEFT))
          BONDG2(8,NBNDG2)=0.50*(BONDG2(8,ICORN)+BONDG2(8,ILEFT))
          BONDG2(9,NBNDG2)=0.50*(BONDG2(9,ICORN)+BONDG2(9,ILEFT))
        ELSE
          IF(IBODSE.EQ.0) GOTO 310
          INC=2**(KDEFG2-KLEVEL-1)
          IBOD=IBODSE-INC
C
          GEOMG2(1,KS)=BODYG2(1,IBOD)
          GEOMG2(2,KS)=BODYG2(2,IBOD)
C
          BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
          BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
          BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
          BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
          BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
          BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
          BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
          BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
          BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
        ENDIF
C
        IBNDG2(1,NBNDG2)=KS
        IBNDG2(2,NBNDG2)=LFSW
        IBNDG2(3,NBNDG2)=LFSE
        IBNDG2(4,NBNDG2)=3
        IBNDG2(5,NBNDG2)=IBNDG2(5,ILEFT)
        IBNDG2(6,NBNDG2)=KLEVEL+1
C
        NBNDG2=NBNDG2+1
C
        IF((KLEVEL+1).GT.KDEFG2) THEN
          H=0.50**(KLEVEL-KDEFG2)
          GEOMG2(1,KW)=GEOMG2(1,KW)
     &                +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ICORN))
          GEOMG2(2,KW)=GEOMG2(2,KW)
     &                +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ICORN))
C
          BONDG2(1,NBNDG2)=(GEOMG2(1,KSW)-GEOMG2(1,KNW))/H
          BONDG2(2,NBNDG2)=(GEOMG2(2,KSW)-GEOMG2(2,KNW))/H
          BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
```

```
            BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
            BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ICORN))
            BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ICORN))
            BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ICORN))
            BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ICORN))
            BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ICORN))
         ELSE
            IF(IBODNW.EQ.0) GOTO 310
            INC=2**(KDEFG2-KLEVEL-1)
            IBOD=IBODNW+INC
C
            GEOMG2(1,KW)=BODYG2(1,IBOD)
            GEOMG2(2,KW)=BODYG2(2,IBOD)
C
            BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
            BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
            BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
            BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
            BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
            BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
            BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
            BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
            BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
         ENDIF
C
         IBNDG2(1,NBNDG2)=KW
         IBNDG2(2,NBNDG2)=LFNW
         IBNDG2(3,NBNDG2)=LFSW
         IBNDG2(4,NBNDG2)=9
         IBNDG2(5,NBNDG2)=IBNDG2(5,IRITE)
         IBNDG2(6,NBNDG2)=KLEVEL+1
C
         IF(LEFS.NE.0) IBNDG2(6,ILEFT)=KLEVEL+1
                       IBNDG2(6,ICORN)=KLEVEL+1
         IF(LNFW.NE.0) IBNDG2(6,IRITE)=KLEVEL+1
C
         ICELG2(10,LFNW)=IOR(ICELG2(10,LFNW),HL0009)
         ICELG2(10,LFSW)=IOR(ICELG2(10,LFSW),HL000B)
         ICELG2(10,LFSE)=IOR(ICELG2(10,LFSE),HL0003)
C
         GOTO 320
C
C         ...DIVIDED CELL WAS SOUTHERN EDGE
C
200      IF(ILEFT.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
         IBNDG2(2,ILEFT)=LFSE
         IBNDG2(3,IRITE)=LFSW
C
         NBNDG2=NBNDG2+1
C
```

```
      IF((KLEVEL+1).GT.KDEFG2) THEN
         H=0.50**(KLEVEL-KDEFG2)
         GEOMG2(1,KS)=GEOMG2(1,KS)
     &               +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ILEFT))
         GEOMG2(2,KS)=GEOMG2(2,KS)
     &               +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ILEFT))
C
         BONDG2(1,NBNDG2)=(GEOMG2(1,KSE)-GEOMG2(1,KSW))/H
         BONDG2(2,NBNDG2)=(GEOMG2(2,KSE)-GEOMG2(2,KSW))/H
         BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
         BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
         BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ILEFT))
         BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ILEFT))
         BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ILEFT))
         BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ILEFT))
         BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ILEFT))
      ELSE
         INC=2**(KDEFG2-KLEVEL-1)
         IF(IBODSW+2*INC.EQ.IBODSE) THEN
            IBOD=IBODSW+INC
         ELSEIF(((GEOMG2(1,KSE)-BODYG2(1,IBODSW+2*INC))**2
     &          +(GEOMG2(2,KSE)-BODYG2(2,IBODSW+2*INC))**2).EQ.0) THEN
            IBOD=IBODSW+INC
         ELSEIF(((GEOMG2(1,KSW)-BODYG2(1,IBODSE-2*INC))**2
     &          +(GEOMG2(2,KSW)-BODYG2(2,IBODSE-2*INC))**2).EQ.0) THEN
            IBOD=IBODSE-INC
         ELSE
            GOTO 310
         ENDIF
C
         GEOMG2(1,KS)=BODYG2(1,IBOD)
         GEOMG2(2,KS)=BODYG2(2,IBOD)
C
         BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
         BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
         BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
         BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
         BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
         BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
         BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
         BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
         BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
      ENDIF
C
      IBNDG2(1,NBNDG2)=KS
      IBNDG2(2,NBNDG2)=LFSW
      IBNDG2(3,NBNDG2)=LFSE
      IBNDG2(4,NBNDG2)=3
      IBNDG2(5,NBNDG2)=MAX(IBNDG2(5,ILEFT),IBNDG2(5,IRITE))
      IBNDG2(6,NBNDG2)=KLEVEL+1
C
```

```
      IF(LEFS.NE.0) THEN
         IEDGE=IBNDG2(4,ILEFT)
         IF    (IEDGE.EQ. 3) THEN
                                          IBNDG2(6,ILEFT)=KLEVEL+1
         ELSEIF(IEDGE.EQ. 1) THEN
            IF(LSEF.NE.0)                 IBNDG2(6,ILEFT)=KLEVEL+1
         ELSEIF(IEDGE.EQ.10) THEN
            IF(LSEF.NE.0 .AND. LSFE.NE.0) IBNDG2(6,ILEFT)=KLEVEL+1
         ELSE
            GOTO 310
         ENDIF
      ENDIF
      IF(LWFS.NE.0) THEN
         IEDGE=IBNDG2(4,IRITE)
         IF    (IEDGE.EQ. 3) THEN
                                          IBNDG2(6,IRITE)=KLEVEL+1
         ELSEIF(IEDGE.EQ. 2) THEN
            IF(LSWF.NE.0                ) IBNDG2(6,IRITE)=KLEVEL+1
         ELSEIF(IEDGE.EQ. 5) THEN
            IF(LSWF.NE.0 .AND. LSFW.NE.0) IBNDG2(6,IRITE)=KLEVEL+1
         ELSE
            GOTO 310
         ENDIF
      ENDIF
C
      ICELG2(10,LFSW)=IOR(ICELG2(10,LFSW),HL0003)
      ICELG2(10,LFSE)=IOR(ICELG2(10,LFSE),HL0003)
C
      GOTO 320
C
C      ...DIVIDED CELL WAS SOUTHEASTERN CORNER (OUTSIDE)
C
210   IF(ILEFT.EQ.0 .OR. ICORN.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
      IBNDG2(2,ICORN)=LFSE
      IBNDG2(2,ILEFT)=LFNE
      IBNDG2(3,IRITE)=LFSW
C
      NBNDG2=NBNDG2+1
C
      IF((KLEVEL+1).GT.KDEFG2) THEN
         H=0.50**(KLEVEL-KDEFG2)
         GEOMG2(1,KS)=GEOMG2(1,KS)
     &               +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ICORN))
         GEOMG2(2,KS)=GEOMG2(2,KS)
     &               +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ICORN))
C
         BONDG2(1,NBNDG2)=(GEOMG2(1,KSE)-GEOMG2(1,KSW))/H
         BONDG2(2,NBNDG2)=(GEOMG2(2,KSE)-GEOMG2(2,KSW))/H
         BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
         BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
         BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ICORN))
```

```
            BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ICORN))
            BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ICORN))
            BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ICORN))
            BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ICORN))
         ELSE
            IF(IBODSW.EQ.0) GOTO 310
            INC=2**(KDEFG2-KLEVEL-1)
            IBOD=IBODSW+INC
C
            GEOMG2(1,KS)=BODYG2(1,IBOD)
            GEOMG2(2,KS)=BODYG2(2,IBOD)
C
            BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
            BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
            BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
            BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
            BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
            BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
            BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
            BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
            BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
         ENDIF
C
         IBNDG2(1,NBNDG2)=KS
         IBNDG2(2,NBNDG2)=LFSW
         IBNDG2(3,NBNDG2)=LFSE
         IBNDG2(4,NBNDG2)=3
         IBNDG2(5,NBNDG2)=IBNDG2(5,IRITE)
         IBNDG2(6,NBNDG2)=KLEVEL+1
C
         NBNDG2=NBNDG2+1
C
         IF((KLEVEL+1).GT.KDEFG2) THEN
            H=0.50**(KLEVEL-KDEFG2)
            GEOMG2(1,KE)=GEOMG2(1,KE)
     &                  +0.125*H*(BONDG2(3,ICORN)-BONDG2(1,ILEFT))
            GEOMG2(2,KE)=GEOMG2(2,KE)
     &                  +0.125*H*(BONDG2(4,ICORN)-BONDG2(2,ILEFT))
C
            BONDG2(1,NBNDG2)=(GEOMG2(1,KNE)-GEOMG2(1,KSE))/H
            BONDG2(2,NBNDG2)=(GEOMG2(2,KNE)-GEOMG2(2,KSE))/H
            BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
            BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
            BONDG2(5,NBNDG2)=0.50*(BONDG2(5,ICORN)+BONDG2(5,ILEFT))
            BONDG2(6,NBNDG2)=0.50*(BONDG2(6,ICORN)+BONDG2(6,ILEFT))
            BONDG2(7,NBNDG2)=0.50*(BONDG2(7,ICORN)+BONDG2(7,ILEFT))
            BONDG2(8,NBNDG2)=0.50*(BONDG2(8,ICORN)+BONDG2(8,ILEFT))
            BONDG2(9,NBNDG2)=0.50*(BONDG2(9,ICORN)+BONDG2(9,ILEFT))
         ELSE
            IF(IBODNE.EQ.0) GOTO 310
            INC=2**(KDEFG2-KLEVEL-1)
```

```
            IBOD=IBODNE-INC
C
            GEOMG2(1,KE)=BODYG2(1,IBOD)
            GEOMG2(2,KE)=BODYG2(2,IBOD)
C
            BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
            BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
            BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
            BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
            BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
            BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
            BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
            BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
            BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
         ENDIF
C
         IBNDG2(1,NBNDG2)=KE
         IBNDG2(2,NBNDG2)=LFSE
         IBNDG2(3,NBNDG2)=LFNE
         IBNDG2(4,NBNDG2)=6
         IBNDG2(5,NBNDG2)=IBNDG2(5,ILEFT)
         IBNDG2(6,NBNDG2)=KLEVEL+1
C
         IF(LNFE.NE.0) IBNDG2(6,ILEFT)=KLEVEL+1
                       IBNDG2(6,ICORN)=KLEVEL+1
         IF(LWFS.NE.0) IBNDG2(6,IRITE)=KLEVEL+1
C
         ICELG2(10,LFSW)=IOR(ICELG2(10,LFSW),HL0003)
         ICELG2(10,LFSE)=IOR(ICELG2(10,LFSE),HL0007)
         ICELG2(10,LFNE)=IOR(ICELG2(10,LFNE),HL0006)
C
         GOTO 320
C
C         ...DIVIDED CEIL WAS EASTERN EDGE
C
220      IF(ILEFT.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
         IBNDG2(2,ILEFT)=LFNE
         IBNDG2(3,IRITE)=LFSE
C
         NBNDG2=NBNDG2+1
C
         IF((KLEVEL+1).GT.KDEFG2) THEN
            H=0.50**(KLEVEL-KDEFG2)
            GEOMG2(1,KE)=GEOMG2(1,KE)
     &                   +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ILEFT))
            GEOMG2(2,KE)=GEOMG2(2,KE)
     &                   +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ILEFT))
C
            BONDG2(1,NBNDG2)=(GEOMG2(1,KNE)-GEOMG2(1,KSE))/H
            BONDG2(2,NBNDG2)=(GEOMG2(2,KNE)-GEOMG2(2,KSE))/H
            BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
```

```
            BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
            BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ILEFT))
            BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ILEFT))
            BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ILEFT))
            BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ILEFT))
            BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ILEFT))
         ELSE
            INC=2**(KDEFG2-KLEVEL-1)
            IF(IBODSE+2*INC.EQ.IBODNE) THEN
               IBOD=IBODSE+INC
            ELSEIF(((GEOMG2(1,KNE)-BODYG2(1,IBODSE+2*INC))**2
     &           +(GEOMG2(2,KNE)-BODYG2(2,IBODSE+2*INC))**2).EQ.0) THEN
               IBOD=IBODSE+INC
            ELSEIF(((GEOMG2(1,KSE)-BODYG2(1,IBODNE-2*INC))**2
     &           +(GEOMG2(2,KSE)-BODYG2(2,IBODNE-2*INC))**2).EQ.0) THEN
               IBOD=IBODNE-INC
            ELSE
               GOTO 310
            ENDIF
C
            GEOMG2(1,KE)=BODYG2(1,IBOD)
            GEOMG2(2,KE)=BODYG2(2,IBOD)
C
            BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
            BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
            BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
            BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
            BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
            BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
            BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
            BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
            BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
         ENDIF
C
         IBNDG2(1,NBNDG2)=KE
         IBNDG2(2,NBNDG2)=LFSE
         IBNDG2(3,NBNDG2)=LFNE
         IBNDG2(4,NBNDG2)=6
         IBNDG2(5,NBNDG2)=MAX(IBNDG2(5,ILEFT),IBNDG2(5,IRITE))
         IBNDG2(6,NBNDG2)=KLEVEL+1
C
         IF(LNFE.NE.0) THEN
            IEDGE=IBNDG2(4,ILEFT)
            IF    (IEDGE.EQ. 6) THEN
                                          IBNDG2(6,ILEFT)=KLEVEL+1
            ELSEIF(IEDGE.EQ. 2) THEN
               IF(LNEF.NE.0               ) IBNDG2(6,ILEFT)=KLEVEL+1
            ELSE
               GOTO 310
            ENDIF
         ENDIF
```

```
      IF(LSFE.NE.0) THEN
         IEDGE=IBNDG2(4,IRITE)
         IF   (IEDGE.EQ. 6) THEN
                                          IBNDG2(6,IRITE)=KLEVEL+1
         ELSEIF(IEDGE.EQ. 4) THEN
            IF(LSEF.NE.0                ) IBNDG2(6,IRITE)=KLEVEL+1
         ELSE
            GOTO 310
         ENDIF
      ENDIF
C
      ICELG2(10,LFSE)=IOR(ICELG2(10,LFSE),HL0006)
      ICELG2(10,LFNE)=IOR(ICELG2(10,LFNE),HL0006)
C
      GOTO 320
C
C      ...DIVIDED CELL WAS NORTHEASTERN CORNER (OUTSIDE)
C
230   IF(ILEFT.EQ.0 .OR. ICORN.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
      IBNDG2(2,ICORN)=LFNE
      IBNDG2(2,ILEFT)=LFNW
      IBNDG2(3,IRITE)=LFSE
C
      NBNDG2=NBNDG2+1
C
      IF((KLEVEL+1).GT.KDEFG2) THEN
         H=0.50**(KLEVEL-KDEFG2)
         GEOMG2(1,KN)=GEOMG2(1,KN)
     &               +0.125*H*(BONDG2(3,ICORN)-BONDG2(1,ILEFT))
         GEOMG2(2,KN)=GEOMG2(2,KN)
     &               +0.125*H*(BONDG2(4,ICORN)-BONDG2(2,ILEFT))
C
         BONDG2(1,NBNDG2)=(GEOMG2(1,KNW)-GEOMG2(1,KNE))/H
         BONDG2(2,NBNDG2)=(GEOMG2(2,KNW)-GEOMG2(2,KNE))/H
         BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
         BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
         BONDG2(5,NBNDG2)=0.50*(BONDG2(5,ICORN)+BONDG2(5,ILEFT))
         BONDG2(6,NBNDG2)=0.50*(BONDG2(6,ICORN)+BONDG2(6,ILEFT))
         BONDG2(7,NBNDG2)=0.50*(BONDG2(7,ICORN)+BONDG2(7,ILEFT))
         BONDG2(8,NBNDG2)=0.50*(BONDG2(8,ICORN)+BONDG2(8,ILEFT))
         BONDG2(9,NBNDG2)=0.50*(BONDG2(9,ICORN)+BONDG2(9,ILEFT))
      ELSE
         IF(IBODNW.EQ.0) GOTO 310
         INC=2**(KDEFG2-KLEVEL-1)
         IBOD=IBODNW-INC
C
         GEOMG2(1,KN)=BODYG2(1,IBOD)
         GEOMG2(2,KN)=BODYG2(2,IBOD)
C
         BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
         BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
```

```
            BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
            BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
            BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
            BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
            BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
            BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
            BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
         ENDIF
C
         IBNDG2(1,NBNDG2)=KN
         IBNDG2(2,NBNDG2)=LFNE
         IBNDG2(3,NBNDG2)=LFNW
         IBNDG2(4,NBNDG2)=12
         IBNDG2(5,NBNDG2)=IBNDG2(5,ILEFT)
         IBNDG2(6,NBNDG2)=KLEVEL+1
C
         NBNDG2=NBNDG2+1
C
         IF((KLEVEL+1).GT.KDEFG2) THEN
            H=0.50**(KLEVEL-KDEFG2)
            GEOMG2(1,KE)=GEOMG2(1,KE)
     &                  +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ICORN))
            GEOMG2(2,KE)=GEOMG2(2,KE)
     &                  +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ICORN))
C
            BONDG2(1,NBNDG2)=(GEOMG2(1,KNE)-GEOMG2(1,KSE))/H
            BONDG2(2,NBNDG2)=(GEOMG2(2,KNE)-GEOMG2(2,KSE))/H
            BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
            BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
            BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ICORN))
            BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ICORN))
            BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ICORN))
            BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ICORN))
            BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ICORN))
         ELSE
            IF(IBODSE.EQ.0) GOTO 310
            INC=2**(KDEFG2-KLEVEL-1)
            IBOD=IBODSE+INC
C
            GEOMG2(1,KE)=BODYG2(1,IBOD)
            GEOMG2(2,KE)=BODYG2(2,IBOD)
C
            BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
            BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
            BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
            BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
            BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
            BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
            BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
            BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
```

```
            BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
         ENDIF
C
         IBNDG2(1,NBNDG2)=KE
         IBNDG2(2,NBNDG2)=LFSE
         IBNDG2(3,NBNDG2)=LFNE
         IBNDG2(4,NBNDG2)=6
         IBNDG2(5,NBNDG2)=IBNDG2(5,IRITE)
         IBNDG2(6,NBNDG2)=KLEVEL+1
C
         IF(LWFN.NE.0) IBNDG2(6,ILEFT)=KLEVEL+1
                       IBNDG2(6,ICORN)=KLEVEL+1
         IF(LSFE.NE.0) IBNDG2(6,IRITE)=KLEVEL+1
C
         ICELG2(10,LFSE)=IOR(ICELG2(10,LFSE),HL0006)
         ICELG2(10,LFNE)=IOR(ICELG2(10,LFNE),HL000E)
         ICELG2(10,LFNW)=IOR(ICELG2(10,LFNW),HL000C)
C
         GOTO 320
C
C         ...DIVIDED CELL WAS NORTHERN EDGE
C
240      IF(ILEFT.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
         IBNDG2(2,ILEFT)=LFNW
         IBNDG2(3,IRITE)=LFNE
C
         NBNDG2=NBNDG2+1
C
         IF((KLEVEL+1).GT.KDEFG2) THEN
            H=0.50**(KLEVEL-KDEFG2)
            GEOMG2(1,KN)=GEOMG2(1,KN)
     &                   +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ILEFT))
            GEOMG2(2,KN)=GEOMG2(2,KN)
     &                   +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ILEFT))
C
            BONDG2(1,NBNDG2)=(GEOMG2(1,KNW)-GEOMG2(1,KNE))/H
            BONDG2(2,NBNDG2)=(GEOMG2(2,KNW)-GEOMG2(2,KNE))/H
            BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
            BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
            BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ILEFT))
            BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ILEFT))
            BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ILEFT))
            BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ILEFT))
            BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ILEFT))
         ELSE
            INC=2**(KDEFG2-KLEVEL-1)
            IF(IBODNE+2*INC.EQ.IBODNW) THEN
               IBOD=IBODNE+INC
            ELSEIF(((GEOMG2(1,KNW)-BODYG2(1,IBODNE+2*INC))**2
     &           +(GEOMG2(2,KNW)-BODYG2(2,IBODNE+2*INC))**2).EQ.0) THEN
               IBOD=IBODNE+INC
```

```
      ELSEIF(((GEOMG2(1,KNE)-BODYG2(1,IBODNW-2*INC))**2
     &       +(GEOMG2(2,KNE)-BODYG2(2,IBODNW-2*INC))**2).EQ.0) THEN
         IBOD=IBODNW-INC
      ELSE
         GOTO 310
      ENDIF
C
      GEOMG2(1,KN)=BODYG2(1,IBOD)
      GEOMG2(2,KN)=BODYG2(2,IBOD)
C
      BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
      BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
      BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
      BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
      BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
      BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
      BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
      BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
    ENDIF
C
    IBNDG2(1,NBNDG2)=KN
    IBNDG2(2,NBNDG2)=LFNE
    IBNDG2(3,NBNDG2)=LFNW
    IBNDG2(4,NBNDG2)=12
    IBNDG2(5,NBNDG2)=MAX(IBNDG2(5,ILEFT),IBNDG2(5,IRITE))
    IBNDG2(6,NBNDG2)=KLEVEL+1
C
    IF(LWFN.NE.0) THEN
      IEDGE=IBNDG2(4,ILEFT)
      IF    (IEDGE.EQ.12) THEN
                                      IBNDG2(6,ILEFT)=KLEVEL+1
      ELSEIF(IEDGE.EQ. 4) THEN
        IF(LNWF.NE.0                ) IBNDG2(6,ILEFT)=KLEVEL+1
      ELSEIF(IEDGE.EQ. 5) THEN
        IF(LNWF.NE.0 .AND. LNFW.NE.0) IBNDG2(6,ILEFT)=KLEVEL+1
      ELSE
        GOTO 310
      ENDIF
    ENDIF
    IF(LEFN.NE.0) THEN
      IEDGE=IBNDG2(4,IRITE)
      IF    (IEDGE.EQ.12) THEN
                                      IBNDG2(6,IRITE)=KLEVEL+1
      ELSEIF(IEDGE.EQ. 8) THEN
        IF(LNEF.NE.0                ) IBNDG2(6,IRITE)=KLEVEL+1
      ELSEIF(IEDGE.EQ.10) THEN
        IF(LNEF.NE.0 .AND. LNFE.NE.0) IBNDG2(6,IRITE)=KLEVEL+1
      ELSE
        GOTO 310
      ENDIF
    ENDIF
```

```
C
      ICELG2(10,LFNE)=IOR(ICELG2(10,LFNE),HL000C)
      ICELG2(10,LFNW)=IOR(ICELG2(10,LFNW),HL000C)
C
      GOTO 320
C
C       ...DIVIDED CELL WAS NORTHWESTERN CORNER (OUTSIDE)
C
250   IF(ILEFT.EQ.0 .OR. ICORN.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
      IBNDG2(2,ICORN)=LFNW
      IBNDG2(2,ILEFT)=LFSW
      IBNDG2(3,IRITE)=LFNE
C
      NBNDG2=NBNDG2+1
C
      IF((KLEVEL+1).GT.KDEFG2) THEN
         H=0.50**(KLEVEL-KDEFG2)
         GEOMG2(1,KN)=GEOMG2(1,KN)
     &                +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ICORN))
         GEOMG2(2,KN)=GEOMG2(2,KN)
     &                +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ICORN))
C
         BONDG2(1,NBNDG2)=(GEOMG2(1,KNW)-GEOMG2(1,KNE))/H
         BONDG2(2,NBNDG2)=(GEOMG2(2,KNW)-GEOMG2(2,KNE))/H
         BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
         BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
         BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ICORN))
         BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ICORN))
         BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ICORN))
         BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ICORN))
         BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ICORN))
      ELSE
         IF(IBODNE.EQ.0) GOTO 310
         INC=2**(KDEFG2-KLEVEL-1)
         IBOD=IBODNE+INC
C
         GEOMG2(1,KN)=BODYG2(1,IBOD)
         GEOMG2(2,KN)=BODYG2(2,IBOD)
C
         BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
         BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
         BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
         BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
         BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
         BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
         BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
         BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
         BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
      ENDIF
C
```

```
      IBNDG2(1,NBNDG2)=KN
      IBNDG2(2,NBNDG2)=LFNE
      IBNDG2(3,NBNDG2)=LFNW
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=IBNDG2(5,IRITE)
      IBNDG2(6,NBNDG2)=KLEVEL+1
C
      NBNDG2=NBNDG2+1
C
      IF((KLEVEL+1).GT.KDEFG2) THEN
         H=0.50**(KLEVEL-KDEFG2)
         GEOMG2(1,KW)=GEOMG2(1,KW)
     &                +0.125*H*(BONDG2(3,ICORN)-BONDG2(1,ILEFT))
         GEOMG2(2,KW)=GEOMG2(2,KW)
     &                +0.125*H*(BONDG2(4,ICORN)-BONDG2(2,ILEFT))
C
         BONDG2(1,NBNDG2)=(GEOMG2(1,KSW)-GEOMG2(1,KNW))/H
         BONDG2(2,NBNDG2)=(GEOMG2(2,KSW)-GEOMG2(2,KNW))/H
         BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
         BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
         BONDG2(5,NBNDG2)=0.50*(BONDG2(5,ICORN)+BONDG2(5,ILEFT))
         BONDG2(6,NBNDG2)=0.50*(BONDG2(6,ICORN)+BONDG2(6,ILEFT))
         BONDG2(7,NBNDG2)=0.50*(BONDG2(7,ICORN)+BONDG2(7,ILEFT))
         BONDG2(8,NBNDG2)=0.50*(BONDG2(8,ICORN)+BONDG2(8,ILEFT))
         BONDG2(9,NBNDG2)=0.50*(BONDG2(9,ICORN)+BONDG2(9,ILEFT))
      ELSE
         IF(IBODSW.EQ.0) GOTO 310
         INC=2**(KDEFG2-KLEVEL-1)
         IBOD=IBODSW-INC
C
         GEOMG2(1,KW)=BODYG2(1,IBOD)
         GEOMG2(2,KW)=BODYG2(2,IBOD)
C
         BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
         BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
         BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
         BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
         BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
         BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
         BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
         BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
         BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
      ENDIF
C
      IBNDG2(1,NBNDG2)=KW
      IBNDG2(2,NBNDG2)=LFNW
      IBNDG2(3,NBNDG2)=LFSW
      IBNDG2(4,NBNDG2)=9
      IBNDG2(5,NBNDG2)=IBNDG2(5,ILEFT)
      IBNDG2(6,NBNDG2)=KLEVEL+1
C
```

```
      IF(LSFW.NE.0) IBNDG2(6,ILEFT)=KLEVEL+1
                    IBNDG2(6,ICORN)=KLEVEL+1
      IF(LEFN.NE.0) IBNDG2(6,IRITE)=KLEVEL+1
C
      ICELG2(10,LFNE)=IOR(ICELG2(10,LFNE),HL000C)
      ICELG2(10,LFNW)=IOR(ICELG2(10,LFNW),HL000D)
      ICELG2(10,LFSW)=IOR(ICELG2(10,LFSW),HL0009)
C
      GOTO 320
C
C       ...DIVIDED CELL WAS WESTERN EDGE
C
260   IF(ILEFT.EQ.0 .OR. IRITE.EQ.0) GOTO 310
C
      IBNDG2(2,ILEFT)=LFSW
      IBNDG2(3,IRITE)=LFNW
C
      NBNDG2=NBNDG2+1
C
      IF((KLEVEL+1).GT.KDEFG2) THEN
         H=0.50**(KLEVEL-KDEFG2)
         GEOMG2(1,KW)=GEOMG2(1,KW)
     &                +0.125*H*(BONDG2(3,IRITE)-BONDG2(1,ILEFT))
         GEOMG2(2,KW)=GEOMG2(2,KW)
     &                +0.125*H*(BONDG2(4,IRITE)-BONDG2(2,ILEFT))
C
         BONDG2(1,NBNDG2)=(GEOMG2(1,KSW)-GEOMG2(1,KNW))/H
         BONDG2(2,NBNDG2)=(GEOMG2(2,KSW)-GEOMG2(2,KNW))/H
         BONDG2(3,NBNDG2)=BONDG2(1,NBNDG2)
         BONDG2(4,NBNDG2)=BONDG2(2,NBNDG2)
C
         BONDG2(5,NBNDG2)=0.50*(BONDG2(5,IRITE)+BONDG2(5,ILEFT))
         BONDG2(6,NBNDG2)=0.50*(BONDG2(6,IRITE)+BONDG2(6,ILEFT))
         BONDG2(7,NBNDG2)=0.50*(BONDG2(7,IRITE)+BONDG2(7,ILEFT))
         BONDG2(8,NBNDG2)=0.50*(BONDG2(8,IRITE)+BONDG2(8,ILEFT))
         BONDG2(9,NBNDG2)=0.50*(BONDG2(9,IRITE)+BONDG2(9,ILEFT))
      ELSE
         INC=2**(KDEFG2-KLEVEL-1)
         IF(IBCDNW+2*INC.EQ.IBODSW) THEN
            IBOD=IBODNW+INC
         ELSEIF(((GEOMG2(1,KSW)-BODYG2(1,IBODNW+2*INC))**2
     &          +(GEOMG2(2,KSW)-BODYG2(2,IBODNW+2*INC))**2).EQ.0) THEN
            IBOD=IBODNW+INC
         ELSEIF(((GEOMG2(1,KNW)-BODYG2(1,IBODSW-2*INC))**2
     &          +(GEOMG2(2,KNW)-BODYG2(2,IBODSW-2*INC))**2).EQ.0) THEN
            IBOD=IBODSW-INC
         ELSE
            GOTO 310
         ENDIF
C
         GEOMG2(1,KW)=BODYG2(1,IBOD)
         GEOMG2(2,KW)=BODYG2(2,IBOD)
C
```

```
            BONDG2(5,NBNDG2)=BODYG2( 3,IBOD)
            BONDG2(6,NBNDG2)=BODYG2( 4,IBOD)
            BONDG2(7,NBNDG2)=BODYG2( 5,IBOD)
            BONDG2(8,NBNDG2)=BODYG2( 6,IBOD)
            BONDG2(9,NBNDG2)=BODYG2( 7,IBOD)
C
            BONDG2(1,NBNDG2)=BODYG2( 8,IBOD)
            BONDG2(2,NBNDG2)=BODYG2( 9,IBOD)
            BONDG2(3,NBNDG2)=BODYG2(10,IBOD)
            BONDG2(4,NBNDG2)=BODYG2(11,IBOD)
         ENDIF
C
         IBNDG2(1,NBNDG2)=KW
         IBNDG2(2,NBNDG2)=LFNW
         IBNDG2(3,NBNDG2)=LFSW
         IBNDG2(4,NBNDG2)=9
         IBNDG2(5,NBNDG2)=MAX(IBNDG2(5,ILEFT),IBNDG2(5,IRITE))
         IBNDG2(6,NBNDG2)=KLEVEL+1
C
         IF(LSFW.NE.0) THEN
            IEDGE=IBNDG2(4,ILEFT)
            IF    (IEDGE.EQ. 9) THEN
                                        IBNDG2(6,ILEFT)=KLEVEL+1
            ELSEIF(IEDGE.EQ. 8) THEN
               IF(LSWF.NE.0             ) IBNDG2(6,ILEFT)=KLEVEL+1
            ELSE
               GOTO 310
            ENDIF
         ENDIF
         IF(LNFW.NE.0) THEN
            IEDGE=IBNDG2(4,IRITE)
            IF    (IEDGE.EQ. 9) THEN
                                        IBNDG2(6,IRITE)=KLEVEL+1
            ELSEIF(IEDGE.EQ. 1) THEN
               IF(LNWF.NE.0             ) IBNDG2(6,IRITE)=KLEVEL+1
            ELSE
               GOTO 310
            ENDIF
         ENDIF
C
         ICELG2(10,LFNW)=IOR(ICELG2(10,LFNW),HLOOO9)
         ICELG2(10,LFSW)=IOR(ICELG2(10,LFSW),HLOOO9)
C
         GOTO 320
C
C     ...DIVIDED CELL WAS SOUTHWEST CORNER (INSIDE)
C
270      ICORN=0
C
         DO 275 IBND=1,NBNDG2
         IF(IBNDG2(1,IBND).EQ.KSW) ICORN=IBND
275      CONTINUE
C
```

```
      IF(ICORN.EQ.0) GOTO 310
C
      IF(LWFS.NE.0 .AND. LSFW.NE.0) THEN
         IEDGE=IBNDG2(4,ICORN)
         IF    (IEDGE.EQ. 1) THEN
                                          IBNDG2(6,ICORN)=KLEVEL+1
         ELSEIF(IEDGE.EQ.10) THEN
            IF(LSWF.NE.0)                 IBNDG2(6,ICORN)=KLEVEL+1
         ELSE
            GOTO 310
         ENDIF
      ENDIF
C
      ICELG2(10,LFSW)=IOR(ICELG2(10,LFSW),HL0001)
C
      GOTO 320
C
C     ...DIVIDED CELL WAS SOUTHEAST CORNER (INSIDE)
C
280   ICORN=0
C
      DO 285 IBND=1,NBNDG2
      IF(IBNDG2(1,IBND).EQ.KSE) ICORN=IBND
285   CONTINUE
C
      IF(ICORN.EQ.0) GOTO 310
C
      IF(LEFS.NE.0 .AND. LSFE.NE.0) THEN
         IEDGE=IBNDG2(4,ICORN)
         IF    (IEDGE.EQ. 2) THEN
                                          IBNDG2(6,ICORN)=KLEVEL+1
         ELSEIF(IEDGE.EQ. 5) THEN
            IF(LSEF.NE.0)                 IBNDG2(6,ICORN)=KLEVEL+1
         ELSE
            GOTO 310
         ENDIF
      ENDIF
C
      ICELG2(10,LFSE)=IOR(ICELG2(10,LFSE),HL0002)
C
      GOTO 320
C
C     ...DIVIDED CELL WAS NORTHEAST CORNER (INSIDE)
C
290   ICORN=0
C
      DO 295 IBND=1,NBNDG2
      IF(IBNDG2(1,IBND).EQ.KNE) ICORN=IBND
295   CONTINUE
C
      IF(ICORN.EQ.0) GOTO 310
C
      IF(LEFN.NE.0 .AND. LNFE.NE.0) THEN
```

```
            IEDGE=IBNDG2(4,ICORN)
            IF    (IEDGE.EQ. 4) THEN
                                              IBNDG2(6,ICORN)=KLEVEL+1
            ELSEIF(IEDGE.EQ. 6) THEN
               IF(LNEF.NE.0)                  IBNDG2(6,ICORN)=KLEVEL+1
            ELSE
               GOTO 310
            ENDIF
          ENDIF
C
          ICELG2(10,LFNE)=IOR(ICELG2(10,LFNE),HL0004)
C
          GOTO 320
C
C     ...DIVIDED CELL WAS NORTHWEST CORNER (INSIDE)
C
300       ICORN=0
C
          DO 305 IBND=1,NBNDG2
          IF(IBNDG2(1,IBND).EQ.KNW) ICORN=IBND
305       CONTINUE
C
          IF(ICORN.EQ.0) GOTO 310
C
          IF(LWFN.NE.0 .AND. LNFW.NE.0) THEN
             IEDGE=IBNDG2(4,ICORN)
             IF    (IEDGE.EQ. 8) THEN
                                              IBNDG2(6,ICORN)=KLEVEL+1
             ELSEIF(IEDGE.EQ.10) THEN
                IF(LNWF.NE.0)                 IBNDG2(6,ICORN)=KLEVEL+1
             ELSE
                GOTO 310
             ENDIF
          ENDIF
C
          ICELG2(10,LFNW)=IOR(ICELG2(10,LFNW),HL0008)
C
          GOTO 320
C
C     ERROR IN BOUNDARY CELL POINTERS
C
310       CALL UTEROR(+5,REAL(LCELL),REAL(KAUX))
C
C     UPDATE ALL NEIGHBOR POINTERS
C
320       NBORG2(1,LCELL)=LFSW
          NBORG2(1,LFSW )=0
          NBORG2(1,LFSE )=0
          NBORG2(1,LFNE )=0
          NBORG2(1,LFNW )=0
C
          NBORG2(8,LFSW)=LLWFN
          NBORG2(9,LFSW)=LLWFS
```

```
      NBORG2(2,LFSW)=LSWF
      NBORG2(3,LFSW)=LLSFW
      NBORG2(4,LFSW)=LLSFE
C
      NBORG2(2,LFSE)=LLSFW
      NBORG2(3,LFSE)=LLSFE
      NBORG2(4,LFSE)=LSEF
      NBORG2(5,LFSE)=LLEFS
      NBORG2(6,LFSE)=LLEFN
C
      NBORG2(4,LFNE)=LLEFS
      NBORG2(5,LFNE)=LLEFN
      NBORG2(6,LFNE)=LNEF
      NBORG2(7,LFNE)=LLNFE
      NBORG2(8,LFNE)=LLNFW
C
      NBORG2(6,LFNW)=LLNFE
      NBORG2(7,LFNW)=LLNFW
      NBORG2(8,LFNW)=LNWF
      NBORG2(9,LFNW)=LLWFN
      NBORG2(2,LFNW)=LLWFS
C
                     NBORG2(2,LFNE )=LFSW
                     NBORG2(3,LFNW )=LFSW
      IF(LLWFN.NE.0) NBORG2(4,LLWFN)=LFSW
      IF(LLWFS.NE.0) NBORG2(5,LLWFS)=LFSW
      IF(LSWF .NE.0) NBORG2(6,LSWF )=LFSW
      IF(LLSFW.NE.0) NBORG2(7,LLSFW)=LFSW
      IF(LLSFE.NE.0) NBORG2(8,LLSFE)=LFSW
                     NBORG2(9,LFSE )=LFSW
C
      IF(LLEFN.NE.0) NBORG2(2,LLEFN)=LFSE
                     NBORG2(3,LFNE )=LFSE
                     NBORG2(4,LFNW )=LFSE
                     NBORG2(5,LFSW )=LFSE
      IF(LLSFW.NE.0) NBORG2(6,LLSFW)=LFSE
      IF(LLSFE.NE.0) NBORG2(7,LLSFE)=LFSE
      IF(LSEF .NE.0) NBORG2(8,LSEF )=LFSE
      IF(LLEFS.NE.0) NBORG2(9,LLEFS)=LFSE
C
      IF(LNEF .NE.0) NBORG2(2,LNEF )=LFNE
      IF(LLNFE.NE.0) NBORG2(3,LLNFE)=LFNE
      IF(LLNFW.NE.0) NBORG2(4,LLNFW)=LFNE
                     NBORG2(5,LFNW )=LFNE
                     NBORG2(6,LFSW )=LFNE
                     NBORG2(7,LFSE )=LFNE
      IF(LLEFS.NE.0) NBORG2(8,LLEFS)=LFNE
      IF(LLEFN.NE.0) NBORG2(9,LLEFN)=LFNE
C
      IF(LLNFE.NE.0) NBORG2(2,LLNFE)=LFNW
      IF(LLNFW.NE.0) NBORG2(3,LLNFW)=LFNW
      IF(LNWF .NE.0) NBORG2(4,LNWF )=LFNW
      IF(LLWFN.NE.0) NBORG2(5,LLWFN)=LFNW
```

```
      IF(LLWFS.NE.0) NBORG2(6,LLWFS)=LFNW
                     NBORG2(7,LFSW )=LFNW
                     NBORG2(8,LFSE )=LFNW
                     NBORG2(9,LFNE )=LFNW
C
      IF(LS.NE.LLS .AND. ICELG2(8,LLS).EQ.KSW) THEN
                       NBORG2(3,LFSW )=-LLSFW
         IF(LLSFW.NE.0) NBORG2(7,LLSFW)=-LFSW
                       NBORG2(4,LFSW )=0
         IF(LLSFE.NE.0) NBORG2(8,LLSFE)=0
                       NBORG2(2,LFSE )=0
         IF(LLSFW.NE.0) NBORG2(6,LLSFW)=0
                       NBORG2(3,LFSE )=0
         IF(LLSFE.NE.0) NBORG2(7,LLSFE)=0
                       NBORG2(4,LFSE )=0
         IF(LSEF .NE.0) NBORG2(8,LSEF )=0
      ENDIF
C
      IF(LS.NE.LLS .AND. ICELG2(6,LLS).EQ.KSE) THEN
                       NBORG2(3,LFSE )=-LLSFE
         IF(LLSFE.NE.0) NBORG2(7,LLSFE)=-LFSE
                       NBORG2(2,LFSE )=0
         IF(LLSFW.NE.0) NBORG2(6,LLSFW)=0
                       NBORG2(4,LFSW )=0
         IF(LLSFE.NE.0) NBORG2(8,LLSFE)=0
                       NBORG2(3,LFSW )=0
         IF(LLSFW.NE.0) NBORG2(7,LLSFW)=0
                       NBORG2(2,LFSW )=0
         IF(LSWF .NE.0) NBORG2(6,LSWF )=0
      ENDIF
C
      IF(LN.NE.LLN .AND. ICELG2(2,LLN).EQ.KNW) THEN
                       NBORG2(7,LFNW )=-LLNFW
         IF(LLNFW.NE.0) NBORG2(3,LLNFW)=-LFNW
                       NBORG2(6,LFNW )=0
         IF(LLNFE.NE.0) NBORG2(2,LLNFE)=0
                       NBORG2(8,LFNE )=0
         IF(LLNFW.NE.0) NBORG2(4,LLNFW)=0
                       NBORG2(7,LFNE )=0
         IF(LLNFE.NE.0) NBORG2(3,LLNFE)=0
                       NBORG2(6,LFNE )=0
         IF(LNEF .NE.0) NBORG2(2,LNEF )=0
      ENDIF
C
      IF(LN.NE.LLN .AND. ICELG2(4,LLN).EQ.KNE) THEN
                       NBORG2(7,LFNE )=-LLNFE
         IF(LLNFE.NE.0) NBORG2(3,LLNFE)=-LFNE
                       NBORG2(8,LFNE )=0
         IF(LLNFW.NE.0) NBORG2(4,LLNFW)=0
                       NBORG2(6,LFNW )=0
         IF(LLNFE.NE.0) NBORG2(2,LLNFE)=0
                       NBORG2(7,LFNW )=0
         IF(LLNFW.NE.0) NBORG2(3,LLNFW)=0
```

```
                        NBORG2(8,LFNW )=0
            IF(LNWF .NE.0) NBORG2(4,LNWF )=0
          ENDIF
C
C     THE DIVIDED CELL IS NO LONGER OUTSIDE THE EMBEDDED MESH
C
          ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFFOF)
C
C     CYCLE THROUGH ALL NEIGHBORING COARSE CELLS, NOTING THAT IF
C       NOT DIVIDED, IT IS JUST OUTSIDE EMBEDDED MESH
C
          IF(LSW.NE.0 .AND. LSWF.EQ.0)
     &               ICELG2(10,LSW)=IOR(ICELG2(10,LSW),HL0040)
C
          IF(LS .NE.0 .AND. LSFW.EQ.0)
     &               ICELG2(10,LS )=IOR(ICELG2(10,LS ),HL00C0)
          IF(LLS.NE.0 .AND. LLSFW.EQ.0 .AND. LSW.NE.0)
     &               ICELG2(10,LLS)=IOR(ICELG2(10,LLS),HL0080)
          IF(LLS.NE.0 .AND. LLSFE.EQ.0 .AND. LSE.NE.0)
     &               ICELG2(10,LLS)=IOR(ICELG2(10,LLS),HL0040)
C
          IF(LSE.NE.0 .AND. LSEF.EQ.0)
     &               ICELG2(10,LSE)=IOR(ICELG2(10,LSE),HL0080)
C
          IF(LE .NE.0 .AND. LEFS.EQ.0)
     &               ICELG2(10,LE )=IOR(ICELG2(10,LE ),HL0090)
C
          IF(LNE.NE.0 .AND. LNEF.EQ.0)
     &               ICELG2(10,LNE)=IOR(ICELG2(10,LNE),HL001C)
C
          IF(LN .NE.0 .AND. LNFW.EQ.0)
     &               ICELG2(10,LN )=IOR(ICELG2(10,LN ),HL00C0)
          IF(LLN.NE.0 .AND. LLNFE.EQ.0 .AND. LNE.NE.0)
     &               ICELG2(10,LLN)=IOR(ICELG2(10,LLN),HL0020)
          IF(LLN.NE.0 .AND. LLNFW.EQ.0 .AND. LNW.NE.0)
     &               ICELG2(10,LLN)=IOR(ICELG2(10,LLN),HL0010)
C
          IF(LNW.NE.0 .AND. LNWF.EQ.0)
     &               ICELG2(10,LNW)=IOR(ICELG2(10,LNW),HL0020)
C
          IF(LW .NE.0 .AND. LWFN.EQ.0)
     &               ICELG2(10,LW )=IOR(ICELG2(10,LW ),HL0060)
C
C     INITIALLY ASSUME THAT LCELL IS JUST INSIDE
C
          ICELG2(10,LCELL)=IOR(ICELG2(10,LCELL),HL0F00)
          ICELG2(10,LFSW )=IOR(ICELG2(10,LFSW ),HLB000)
          ICELG2(10,LFSE )=IOR(ICELG2(10,LFSE ),HL7000)
          ICELG2(10,LFNE )=IOR(ICELG2(10,LFNE ),HLE000)
          ICELG2(10,LFNW )=IOR(ICELG2(10,LFNW ),HLD000)
C
C     FOR ANY CORNER NODE OF LCELL WHICH IS SURROUNDED OR ON A
C       BOUNDARY, CANCEL THE INSIDE POINTERS
```

```
C
      IF    (IAND(ICELG2(10,LCELL),HL000B).EQ.HL0001) THEN
         IF(LSW.EQ.0) THEN
            IF(LWFS.NE.0                    .AND. LSFW.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFEFF)
               ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFDFF)
               ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLF7FF)
               ICELG2(10,LFSW )=IAND(ICELG2(10,LFSW ),HLEFFF)
               ICELG2(10,LWFS )=IAND(ICELG2(10,LWFS ),HLDFFF)
               ICELG2(10,LSFW )=IAND(ICELG2(10,LSFW ),HL7FFF)
            ENDIF
         ELSEIF(LSWF.NE.0) THEN
            IF(LWFS.NE.0                    .AND. LSFW.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFEFF)
               ICELG2(10,LSW  )=IAND(ICELG2(10,LSW  ),HLFBFF)
               ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFDFF)
               ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLF7FF)
               ICELG2(10,LFSW )=IAND(ICELG2(10,LFSW ),HLEFFF)
               ICELG2(10,LSWF )=IAND(ICELG2(10,LSWF ),HLBFFF)
               ICELG2(10,LWFS )=IAND(ICELG2(10,LWFS ),HLDFFF)
               ICELG2(10,LSFW )=IAND(ICELG2(10,LSFW ),HL7FFF)
            ENDIF
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000B).EQ.HL0003
     &                  .AND. LLS.NE.0 .AND. LSW.NE.0) THEN
         IF(LWFS.NE.0 .AND. LSWF.NE.0 .AND. LLSFW.NE.0) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFEFF)
            ICELG2(10,LSW  )=IAND(ICELG2(10,LSW  ),HLF8FF)
            ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFDFF)
            ICELG2(10,LLS  )=IAND(ICELG2(10,LLS  ),HLF7FF)
            ICELG2(10,LFSW )=IAND(ICELG2(10,LFSW ),HLEFFF)
            ICELG2(10,LSWF )=IAND(ICELG2(10,LSWF ),HLBFFF)
            ICELG2(10,LWFS )=IAND(ICELG2(10,LWFS ),HLDFFF)
            ICELG2(10,LLSFW)=IAND(ICELG2(10,LLSFW),HL7FFF)
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000B).EQ.HL0003) THEN
         IF(LWFS.NE.0                                    ) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFEFF)
            ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFDFF)
            ICELG2(10,LFSW )=IAND(ICELG2(10,LFSW ),HLEFFF)
            ICELG2(10,LWFS )=IAND(ICELG2(10,LWFS ),HLDFFF)
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000B).EQ.HL0009) THEN
         IF(                                LSFW.NE.0) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFEFF)
            ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLF7FF)
            ICELG2(10,LFSW )=IAND(ICELG2(10,LFSW ),HLEFFF)
            ICELG2(10,LSFW )=IAND(ICELG2(10,LSFW ),HL7FFF)
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000B).EQ.HL000B) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFEFF)
            ICELG2(10,LFSW )=IAND(ICELG2(10,LFSW ),HLEFFF)
      ELSE
```

```
            IF(LWFS.NE.0 .AND. LSWF.NE.0 .AND. LSFW.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFEFF)
               ICELG2(10,LSW  )=IAND(ICELG2(10,LSW  ),HLFBFF)
               ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFDFF)
               ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLF7FF)
               ICELG2(10,LFSW )=IAND(ICELG2(10,LFSW ),HLEFFF)
               ICELG2(10,LSWF )=IAND(ICELG2(10,LSWF ),HLBFFF)
               ICELG2(10,LWFS )=IAND(ICELG2(10,LWFS ),HLDFFF)
               ICELG2(10,LSFW )=IAND(ICELG2(10,LSFW ),HL7FFF)
            ENDIF
         ENDIF
C
         IF    (IAND(ICELG2(10,LCELL),HL0007).EQ.HL0002) THEN
            IF(LSE.NE.0) THEN
               IF(LEFS.NE.0                    .AND. LSFE.NE.0) THEN
                  ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFDFF)
                  ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLFEFF)
                  ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLFBFF)
                  ICELG2(10,LFSE )=IAND(ICELG2(10,LFSE ),HLDFFF)
                  ICELG2(10,LEFS )=IAND(ICELG2(10,LEFS ),HLEFFF)
                  ICELG2(10,LSFE )=IAND(ICELG2(10,LSFE ),HLBFFF)
               ENDIF
            ELSEIF(LSEF.NE.0) THEN
               IF(LEFS.NE.0                    .AND. LSFE.NE.0) THEN
                  ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFDFF)
                  ICELG2(10,LSE  )=IAND(ICELG2(10,LSE  ),HLF7FF)
                  ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLFEFF)
                  ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLFBFF)
                  ICELG2(10,LFSE )=IAND(ICELG2(10,LFSE ),HLDFFF)
                  ICELG2(10,LSEF )=IAND(ICELG2(10,LSEF ),HL7FFF)
                  ICELG2(10,LEFS )=IAND(ICELG2(10,LEFS ),HLEFFF)
                  ICELG2(10,LSFE )=IAND(ICELG2(10,LSFE ),HLBFFF)
               ENDIF
            ENDIF
         ELSEIF(IAND(ICELG2(10,LCELL),HL0007).EQ.HL0003
     &                  .AND. LLS.NE.0 .AND. LSE.NE.0) THEN
            IF(LEFS.NE.0 .AND. LSEF.NE.0 .AND. LLSFE.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFDFF)
               ICELG2(10,LSE  )=IAND(ICELG2(10,LSE  ),HLF7FF)
               ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLFEFF)
               ICELG2(10,LLS  )=IAND(ICELG2(10,LLS  ),HLFBFF)
               ICELG2(10,LFSE )=IAND(ICELG2(10,LFSE ),HLDFFF)
               ICELG2(10,LSEF )=IAND(ICELG2(10,LSEF ),HL7FFF)
               ICELG2(10,LEFS )=IAND(ICELG2(10,LEFS ),HLEFFF)
               ICELG2(10,LLSFE)=IAND(ICELG2(10,LLSFE),HLBFFF)
            ENDIF
         ELSEIF(IAND(ICELG2(10,LCELL),HL0007).EQ.HL0003) THEN
            IF(LEFS.NE.0                                  ) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFDFF)
               ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLFEFF)
               ICELG2(10,LFSE )=IAND(ICELG2(10,LFSE ),HLDFFF)
               ICELG2(10,LEFS )=IAND(ICELG2(10,LEFS ),HLEFFF)
            ENDIF
```

```
      ELSEIF(IAND(ICELG2(10,LCELL),HL0007).EQ.HL0006) THEN
         IF(                                    LSFE.NE.0) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFDFF)
            ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLFBFF)
            ICELG2(10,LFSE )=IAND(ICELG2(10,LFSE ),HLDFFF)
            ICELG2(10,LSFE )=IAND(ICELG2(10,LSFE ),HLBFFF)
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL0007).EQ.HL0007) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFDFF)
            ICELG2(10,LFSE )=IAND(ICELG2(10,LFSE ),HLDFFF)
      ELSE
         IF(LEFS.NE.0 .AND. LSEF.NE.0 .AND. LSFE.NE.0) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFDFF)
            ICELG2(10,LSE  )=IAND(ICELG2(10,LSE  ),HLF7FF)
            ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLFEFF)
            ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLFBFF)
            ICELG2(10,LFSE )=IAND(ICELG2(10,LFSE ),HLDFFF)
            ICELG2(10,LSEF )=IAND(ICELG2(10,LSEF ),HL7FFF)
            ICELG2(10,LEFS )=IAND(ICELG2(10,LEFS ),HLEFFF)
            ICELG2(10,LSFE )=IAND(ICELG2(10,LSFE ),HLBFFF)
         ENDIF
      ENDIF
C
      IF    (IAND(ICELG2(10,LCELL),HL000E).EQ.HL0004) THEN
         IF(LNE.EQ.0) THEN
            IF(LEFN.NE.0                    .AND. LNFE.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFBFF)
               ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLF7FF)
               ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFDFF)
               ICELG2(10,LFNE )=IAND(ICELG2(10,LFNE ),HLBFFF)
               ICELG2(10,LEFN )=IAND(ICELG2(10,LEFN ),HL7FFF)
               ICELG2(10,LNFE )=IAND(ICELG2(10,LNFE ),HLDFFF)
            ENDIF
         ELSEIF(LNEF.NE.0) THEN
            IF(LEFN.NE.0                    .AND. LNFE.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFBFF)
               ICELG2(10,LNE  )=IAND(ICELG2(10,LNE  ),HLFEFF)
               ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLF7FF)
               ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFDFF)
               ICELG2(10,LFNE )=IAND(ICELG2(10,LFNE ),HLBFFF)
               ICELG2(10,LNEF )=IAND(ICELG2(10,LNEF ),HLEFFF)
               ICELG2(10,LEFN )=IAND(ICELG2(10,LEFN ),HL7FFF)
               ICELG2(10,LNFE )=IAND(ICELG2(10,LNFE ),HLDFFF)
            ENDIF
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000E).EQ.HL000C
     &                  .AND. LLN.NE.0 .AND. LNE.NE.0) THEN
         IF(LEFN.NE.0 .AND. LNEF.NE.0 .AND. LLNFE.NE.0) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFBFF)
            ICELG2(10,LNE  )=IAND(ICELG2(10,LNE  ),HLFEFF)
            ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLF7FF)
            ICELG2(10,LLN  )=IAND(ICELG2(10,LLN  ),HLFDFF)
            ICELG2(10,LFNE )=IAND(ICELG2(10,LFNE ),HLBFFF)
```

```
            ICELG2(10,LNEF )=IAND(ICELG2(10,LNEF ),HLEFFF)
            ICELG2(10,LEFN )=IAND(ICELG2(10,LEFN ),HL7FFF)
            ICELG2(10,LLNFE)=IAND(ICELG2(10,LLNFE),HLDFFF)
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000E).EQ.HL000C) THEN
         IF(LEFN.NE.0                                    ) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFBFF)
            ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLF7FF)
            ICELG2(10,LFNE )=IAND(ICELG2(10,LFNE ),HLBFFF)
            ICELG2(10,LEFN )=IAND(ICELG2(10,LEFN ),HL7FFF)
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000E).EQ.HL0006) THEN
         IF(                                    LNFE.NE.0) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFBFF)
            ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFDFF)
            ICELG2(10,LFNE )=IAND(ICELG2(10,LFNE ),HLBFFF)
            ICELG2(10,LNFE )=IAND(ICELG2(10,LNFE ),HLDFFF)
         ENDIF
      ELSEIF(IAND(ICELG2(10,LCELL),HL000E).EQ.HL000E) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFBFF)
            ICELG2(10,LFNE )=IAND(ICELG2(10,LFNE ),HLBFFF)
      ELSE
         IF(LEFN.NE.0 .AND. LNEF.NE.0 .AND. LNFE.NE.0) THEN
            ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFBFF)
            ICELG2(10,LNE  )=IAND(ICELG2(10,LNE  ),HLFEFF)
            ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLF7FF)
            ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFDFF)
            ICELG2(10,LFNE )=IAND(ICELG2(10,LFNE ),HLBFFF)
            ICELG2(10,LNEF )=IAND(ICELG2(10,LNEF ),HLEFFF)
            ICELG2(10,LEFN )=IAND(ICELG2(10,LEFN ),HL7FFF)
            ICELG2(10,LNFE )=IAND(ICELG2(10,LNFE ),HLDFFF)
         ENDIF
      ENDIF
C
      IF    (IAND(ICELG2(10,LCELL),HL000D).EQ.HL0008) THEN
         IF(LNW.EQ.0) THEN
            IF(LWFN.NE.0                    .AND. LNFW.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLF7FF)
               ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFBFF)
               ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFEFF)
               ICELG2(10,LFNW )=IAND(ICELG2(10,LFNW ),HL7FFF)
               ICELG2(10,LWFN )=IAND(ICELG2(10,LWFN ),HLBFFF)
               ICELG2(10,LNFW )=IAND(ICELG2(10,LNFW ),HLEFFF)
            ENDIF
         ELSEIF(LNWF.NE.0) THEN
            IF(LWFN.NE.0                    .AND. LNFW.NE.0) THEN
               ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLF7FF)
               ICELG2(10,LNW  )=IAND(ICELG2(10,LNW  ),HLFDFF)
               ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFBFF)
               ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFEFF)
               ICELG2(10,LFNW )=IAND(ICELG2(10,LFNW ),HL7FFF)
               ICELG2(10,LNWF )=IAND(ICELG2(10,LNWF ),HLDFFF)
               ICELG2(10,LWFN )=IAND(ICELG2(10,LWFN ),HLBFFF)
```

```
            ICELG2(10,LNFW )=IAND(ICELG2(10,LNFW ),HLEFFF)
         ENDIF
      ENDIF
   ELSEIF(IAND(ICELG2(10,LCELL),HL000D).EQ.HL000C
     &                .AND. LLN.NE.0 .AND. LNW.NE.0) THEN
      IF(LWFN.NE.0 .AND. LNWF.NE.0 .AND. LLNFW.NE.0) THEN
         ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLF7FF)
         ICELG2(10,LNW  )=IAND(ICELG2(10,LNW  ),HLFDFF)
         ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFBFF)
         ICELG2(10,LLN  )=IAND(ICELG2(10,LLN  ),HLFEFF)
         ICELG2(10,LFNW )=IAND(ICELG2(10,LFNW ),HL7FFF)
         ICELG2(10,LNWF )=IAND(ICELG2(10,LNWF ),HLDFFF)
         ICELG2(10,LWFN )=IAND(ICELG2(10,LWFN ),HLBFFF)
         ICELG2(10,LLNFW)=IAND(ICELG2(10,LLNFW),HLEFFF)
      ENDIF
   ELSEIF(IAND(ICELG2(10,LCELL),HL000D).EQ.HL000C) THEN
      IF(LWFN.NE.0                                    ) THEN
         ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLF7FF)
         ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFBFF)
         ICELG2(10,LFNW )=IAND(ICELG2(10,LFNW ),HL7FFF)
         ICELG2(10,LWFN )=IAND(ICELG2(10,LWFN ),HLBFFF)
      ENDIF
   ELSEIF(IAND(ICELG2(10,LCELL),HL000D).EQ.HL0009) THEN
      IF(                                  LNFW.NE.0) THEN
         ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLF7FF)
         ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFEFF)
         ICELG2(10,LFNW )=IAND(ICELG2(10,LFNW ),HL7FFF)
         ICELG2(10,LNFW )=IAND(ICELG2(10,LNFW ),HLEFFF)
      ENDIF
   ELSEIF(IAND(ICELG2(10,LCELL),HL000D).EQ.HL000D) THEN
         ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLF7FF)
         ICELG2(10,LFNW )=IAND(ICELG2(10,LFNW ),HL7FFF)
   ELSE
      IF(LWFN.NE.0 .AND. LNWF.NE.0 .AND. LNFW.NE.0) THEN
         ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLF7FF)
         ICELG2(10,LNW  )=IAND(ICELG2(10,LNW  ),HLFDFF)
         ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFBFF)
         ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFEFF)
         ICELG2(10,LFNW )=IAND(ICELG2(10,LFNW ),HL7FFF)
         ICELG2(10,LNWF )=IAND(ICELG2(10,LNWF ),HLDFFF)
         ICELG2(10,LWFN )=IAND(ICELG2(10,LWFN ),HLBFFF)
         ICELG2(10,LNFW )=IAND(ICELG2(10,LNFW ),HLEFFF)
      ENDIF
   ENDIF
C
   IF(LS.EQ.0 .OR. LSFW.NE.0) THEN
                  ICELG2(10,LFSW)=IAND(ICELG2(10,LFSW),HLDFFF)
                  ICELG2(10,LFSE)=IAND(ICELG2(10,LFSE),HLEFFF)
      IF(LSFW.NE.0) ICELG2(10,LSFW)=IAND(ICELG2(10,LSFW),HLBFFF)
      IF(LSFE.NE.0) ICELG2(10,LSFE)=IAND(ICELG2(10,LSFE),HL7FFF)
   ENDIF
C
   IF(LE.EQ.0 .OR. LEFS.NE.0) THEN
```

```
                        ICELG2(10,LFSE)=IAND(ICELG2(10,LFSE),HLBFFF)
                        ICELG2(10,LFNE)=IAND(ICELG2(10,LFNE),HLDFFF)
           IF(LEFS.NE.0) ICELG2(10,LEFS)=IAND(ICELG2(10,LEFS),HL7FFF)
           IF(LEFN.NE.0) ICELG2(10,LEFN)=IAND(ICELG2(10,LEFN),HLEFFF)
         ENDIF
C
         IF(LN.EQ.0 .OR. LNFE.NE.0) THEN
                        ICELG2(10,LFNE)=IAND(ICELG2(10,LFNE),HL7FFF)
                        ICELG2(10,LFNW)=IAND(ICELG2(10,LFNW),HLBFFF)
           IF(LNFE.NE.0) ICELG2(10,LNFE)=IAND(ICELG2(10,LNFE),HLEFFF)
           IF(LNFW.NE.0) ICELG2(10,LNFW)=IAND(ICELG2(10,LNFW),HLDFFF)
         ENDIF
C
         IF(LW.EQ.0 .OR. LWFN.NE.0) THEN
                        ICELG2(10,LFNW)=IAND(ICELG2(10,LFNW),HLEFFF)
                        ICELG2(10,LFSW)=IAND(ICELG2(10,LFSW),HL7FFF)
           IF(LWFN.NE.0) ICELG2(10,LWFN)=IAND(ICELG2(10,LWFN),HLDFFF)
           IF(LWFS.NE.0) ICELG2(10,LWFS)=IAND(ICELG2(10,LWFS),HLBFFF)
         ENDIF
C
         ICHNG=1
         RETURN
C
         END
```

G2EXAM

```
         SUBROUTINE G2EXAM
C
         INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE ALLOWS A USER TO EXAMINE DATA STRUCTURE ENTRIES
C
C**********************************************************************
C
         INCLUDE '[.GRID2D]G2COMN.INC'
C
         INCLUDE '[.UTILITY]IOUNIT.INC'
C
         CHARACTER*2 CORNER
C
C**********************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
         IF(INBRG2.EQ.0) THEN
            CALL UTEROR(+1,REAL(INBRG2),0.0)
         ENDIF
C
10       WRITE(JTERMO,20) NNODG2,
```

```
     &                  NCELG2,
     &                  NBNDG2,
     &                  NBODG2,
     &                  MLVLG2
20      FORMAT(' 1. NODE VALUES     1 -> ',I6/
     &         ' 2. CELL VALUES     1 -> ',I6/
     &         ' 3. BOUNDARY VALUES 1 -> ',I6/
     &         ' 4. BODY VALUES     1 -> ',I6/
     &         ' 5. LEVEL VALUES    0 -> ',I6//
     &         ' 6. SEARCH NODES'           /
     &         ' 7. SEARCH CELLS'          //
     &         ' 0. EXIT                '    )
        CALL UTINPI('OPTION',ITYPE)
C
        IF(ITYPE.EQ.0) THEN
           RETURN
C
C.....EXAMINE/CHANGE NODES
C
        ELSEIF(ITYPE.EQ.1) THEN
           CALL UTINPI('NODE NUMBER',INODE)
           INODE=MAX(1,MIN(NNODG2,INODE))
C
           WRITE(JTERMO,30) INODE,
     &                      GEOMG2( 1,INODE),GEOMG2( 2,INODE),
     &                      DPENG2( 1,INODE),DPENG2( 2,INODE),
     &                      DPENG2( 3,INODE),DPENG2( 4,INODE),
     &                      DPENG2( 5,INODE),WORKG2(   INODE)
30         FORMAT(' NODE  ',I6                                    /
     &            '  1. GEOMG2( 1)',G15.7,5X,'  2. GEOMG2( 2)',G15.7/
     &            '  3. DPENG2( 1)',G15.7,5X,'  4. DPENG2( 2)',G15.7/
     &            '  5. DPENG2( 3)',G15.7,5X,'  6. DPENG2( 4)',G15.7/
     &            '  7. DPENG2( 5)',G15.7,5X,'  8. WORKG2(  )',G15.7)
C
           CALL UTINPI('VARIABLE NUMBER (OR 0. TO EXIT)',INUM)
           IF(INUM.EQ. 1) CALL UTINPF('GEOMG2( 1)',GEOMG2( 1,INODE))
           IF(INUM.EQ. 2) CALL UTINPF('GEOMG2( 2)',GEOMG2( 2,INODE))
           IF(INUM.EQ. 3) CALL UTINPF('DPENG2( 1)',DPENG2( 1,INODE))
           IF(INUM.EQ. 4) CALL UTINPF('DPENG2( 2)',DPENG2( 2,INODE))
           IF(INUM.EQ. 5) CALL UTINPF('DPENG2( 3)',DPENG2( 3,INODE))
           IF(INUM.EQ. 6) CALL UTINPF('DPENG2( 4)',DPENG2( 4,INODE))
           IF(INUM.EQ. 7) CALL UTINPF('DPENG2( 5)',DPENG2( 5,INODE))
           IF(INUM.EQ. 8) CALL UTINPF('WORKG2(  )',WORKG2(   INODE))
C
C.....EXAMINE/CHANGE CELLS
C
        ELSEIF(ITYPE.EQ.2) THEN
           CALL UTINPI('CELL NUMBER',ICELL)
           ICELL=MAX(1,MIN(NCELG2,ICELL))
C
           WRITE(JTERMO,40) ICELL,
     &                      ICELG2( 1,ICELL),ICELG2( 2,ICELL),
     &                      ICELG2( 3,ICELL),ICELG2( 4,ICELL),
```

```
     &                    ICELG2( 5,ICELL),ICELG2( 6,ICELL),
     &                    ICELG2( 7,ICELL),ICELG2( 8,ICELL),
     &                    ICELG2( 9,ICELL),ICELG2(10,ICELL),
     &                    NBORG2( 1,ICELL),NBORG2( 2,ICELL),
     &                    NBORG2( 3,ICELL),NBORG2( 4,ICELL),
     &                    NBORG2( 5,ICELL),NBORG2( 6,ICELL),
     &                    NBORG2( 7,ICELL),NBORG2( 8,ICELL),
     &                    NBORG2( 9,ICELL)
40        FORMAT(' CELL  ',I6                                          /
     &          '  1. ICELG2( 1)',I15  ,5X,'  2. ICELG2( 2)',I15  /
     &          '  3. ICELG2( 3)',I15  ,5X,'  4. ICELG2( 4)',I16  /
     &          '  5. ICELG2( 5)',I15  ,5X,'  6. ICELG2( 6)',I15  /
     &          '  7. ICELG2( 7)',I15  ,5X,'  8. ICELG2( 8)',I15  /
     &          '  9. ICELG2( 9)',I15  ,5X,' 10. ICELG2(10)',Z10  /
     &          ' 11. NBORG2( 1)',I15  ,5X,' 12. NBORG2( 2)',I15  /
     &          ' 13. NBORG2( 3)',I15  ,5X,' 14. NBORG2( 4)',I15  /
     &          ' 15. NBORG2( 5)',I15  ,5X,' 16. NBORG2( 6)',I15  /
     &          ' 17. NBORG2( 7)',I15  ,5X,' 18. NBORG2( 8)',I15  /
     &          ' 19. NBORG2( 9)',I15                              )
C
          CALL UTINPI('VARIABLE NUMBER (OR 0. TO EXIT)',INUM)
          IF(INUM.EQ. 1) CALL UTINPI('ICELG2( 1)',ICELG2( 1,ICELL))
          IF(INUM.EQ. 2) CALL UTINPI('ICELG2( 2)',ICELG2( 2,ICELL))
          IF(INUM.EQ. 3) CALL UTINPI('ICELG2( 3)',ICELG2( 3,ICELL))
          IF(INUM.EQ. 4) CALL UTINPI('ICELG2( 4)',ICELG2( 4,ICELL))
          IF(INUM.EQ. 5) CALL UTINPI('ICELG2( 5)',ICELG2( 5,ICELL))
          IF(INUM.EQ. 6) CALL UTINPI('ICELG2( 6)',ICELG2( 6,ICELL))
          IF(INUM.EQ. 7) CALL UTINPI('ICELG2( 7)',ICELG2( 7,ICELL))
          IF(INUM.EQ. 8) CALL UTINPI('ICELG2( 8)',ICELG2( 8,ICELL))
          IF(INUM.EQ. 9) CALL UTINPI('ICELG2( 9)',ICELG2( 9,ICELL))
          IF(INUM.EQ.10) CALL UTINPI('ICELG2(10)',ICELG2(10,ICELL))
          IF(INUM.EQ.11) CALL UTINPI('NBORG2( 1)',NBORG2( 1,ICELL))
          IF(INUM.EQ.12) CALL UTINPI('NBORG2( 2)',NBORG2( 2,ICELL))
          IF(INUM.EQ.13) CALL UTINPI('NBORG2( 3)',NBORG2( 3,ICELL))
          IF(INUM.EQ.14) CALL UTINPI('NBORG2( 4)',NBORG2( 4,ICELL))
          IF(INUM.EQ.15) CALL UTINPI('NBORG2( 5)',NBORG2( 5,ICELL))
          IF(INUM.EQ.16) CALL UTINPI('NBORG2( 6)',NBORG2( 6,ICELL))
          IF(INUM.EQ.17) CALL UTINPI('NBORG2( 7)',NBORG2( 7,ICELL))
          IF(INUM.EQ.18) CALL UTINPI('NBORG2( 8)',NBORG2( 8,ICELL))
          IF(INUM.EQ.19) CALL UTINPI('NBORG2( 9)',NBORG2( 9,ICELL))
C
C.....EXAMINE/CHANGE BOUNDARIES
C
        ELSEIF(ITYPE.EQ.3) THEN
          CALL UTINPI('BOUNDARY NUMBER',IBOUND)
          IBOUND=MAX(1,MIN(NBNDG2,IBOUND))
C
          WRITE(JTERMO,50) IBOUND,
     &                    IBNDG2( 1,IBOUND),IBNDG2( 2,IBOUND),
     &                    IBNDG2( 3,IBOUND),IBNDG2( 4,IBOUND),
     &                    IBNDG2( 5,IBOUND),IBNDG2( 6,IBOUND),
     &                    BONDG2( 1,IBOUND),BONDG2( 2,IBOUND),
     &                    BONDG2( 3,IBOUND),BONDG2( 4,IBOUND),
```

```
     &                    BONDG2( 5,IBOUND),BONDG2( 6,IBOUND),
     &                    BONDG2( 7,IBOUND),BONDG2( 8,IBOUND),
     &                    BONDG2( 9,IBOUND)
50        FORMAT(' BOUND ',I6                                          /
     &          '  1. IBNDG2( 1)',I15  ,5X,'  2. IBNDG2( 2)',I15  /
     &          '  3. IBNDG2( 3)',I15  ,5X,'  4. IBNDG2( 4)',I15  /
     &          '  5. IBNDG2( 5)',I15  ,5X,'  6. IBNDG2( 6)',I15  /
     &          '  7. BONDG2( 1)',G15.7,5X,'  8. BONDG2( 2)',G15.7/
     &          '  9. BONDG2( 3)',G15.7,5X,' 10. BONDG2( 4)',G15.7/
     &          ' 11. BONDG2( 5)',G15.7,5X,' 12. BONDG2( 6)',G15.7/
     &          ' 13. BONDG2( 7)',G15.7,5X,' 14. BONDG2( 8)',G15.7/
     &          ' 15. BONDG2( 9)',G15.7                          )
C
          CALL UTINPI('VARIABLE NUMBER (OR 0. TO EXIT)',INUM)
          IF(INUM.EQ. 1) CALL UTINPI('IBNDG2( 1)',IBNDG2( 1,IBOUND))
          IF(INUM.EQ. 2) CALL UTINPI('IBNDG2( 2)',IBNDG2( 2,IBOUND))
          IF(INUM.EQ. 3) CALL UTINPI('IBNDG2( 3)',IBNDG2( 3,IBOUND))
          IF(INUM.EQ. 4) CALL UTINPI('IBNDG2( 4)',IBNDG2( 4,IBOUND))
          IF(INUM.EQ. 5) CALL UTINPI('IBNDG2( 5)',IBNDG2( 5,IBOUND))
          IF(INUM.EQ. 6) CALL UTINPI('IBNDG2( 6)',IBNDG2( 6,IBOUND))
          IF(INUM.EQ. 7) CALL UTINPF('BONDG2( 1)',BONDG2( 1,IBOUND))
          IF(INUM.EQ. 8) CALL UTINPF('BONDG2( 2)',BONDG2( 2,IBOUND))
          IF(INUM.EQ. 9) CALL UTINPF('BONDG2( 3)',BONDG2( 3,IBOUND))
          IF(INUN.EQ.10) CALL UTINPF('BONDG2( 4)',BONDG2( 4,IBOUND))
          IF(INUM.EQ.11) CALL UTINPF('BONDG2( 5)',BONDG2( 5,IBOUND))
          IF(INUM.EQ.12) CALL UTINPF('BONDG2( 6)',BONDG2( 6,IBOUND))
          IF(INUN.EQ.13) CALL UTINPF('BONDG2( 7)',BONDG2( 7,IBOUND))
          IF(INUN.EQ.14) CALL UTINPF('BONDG2( 8)',BONDG2( 8,IBOUND))
          IF(INUM.EQ.15) CALL UTINPF('BONDG2( 9)',BONDG2( 9,IBOUND))
C
C.....EXAMINE/CHANGE BODIES
C
        ELSEIF(ITYPE.EQ.4) THEN
          CALL UTINPI('BODY POINT',IBODY)
          IBODY=MAX(1,MIN(NBODG2,IBODY))
C
          WRITE(JTERMO,60) IBODY,
     &                    BODYG2( 1,IBODY),BODYG2( 2,IBODY),
     &                    BODYG2( 3,IBODY),BODYG2( 4,IBODY),
     &                    BODYG2( 5,IBODY),BODYG2( 6,IBODY),
     &                    BODYG2( 7,IBODY),BODYG2( 8,IBODY),
     &                    BODYG2( 9,IBODY),BODYG2(10,IBODY),
     &                    BODYG2(11,IBODY)
60        FORMAT(' BODY  ',I6                                          /
     &          '  1. BODYG2( 1)',G15.7,5X,'  2. BODYG2( 2)',G15.7/
     &          '  3. BODYG2( 3)',G15.7,5X,'  4. BODYG2( 4)',G15.7/
     &          '  5. BODYG2( 5)',G15.7,5X,'  6. BODYG2( 6)',G15.7/
     &          '  7. BODYG2( 7)',G15.7,5X,'  8. BODYG2( 8)',G15.7/
     &          '  9. BODYG2( 9)',G15.7,5X,' 10. BODYG2(10)',G15.7/
     &          ' 11. BODYG2(11)',G15.7                          )
C
          CALL UTINPI('VARIABLE NUMBER (OR 0. TO EXIT)',INUM)
          IF(INUM.EQ. 1) CALL UTINPF('BODYG2( 1)',BODYG2( 1,IBODY))
```

```
            IF(INUM.EQ. 2) CALL UTINPF('BODYG2( 2)',BODYG2( 2,IBODY))
            IF(INUM.EQ. 3) CALL UTINPF('BODYG2( 3)',BODYG2( 3,IBODY))
            IF(INUM.EQ. 4) CALL UTINPF('BODYG2( 4)',BODYG2( 4,IBODY))
            IF(INUM.EQ. 5) CALL UTINPF('BODYG2( 5)',BODYG2( 5,IBODY))
            IF(INUM.EQ. 6) CALL UTINPF('BODYG2( 6)',BODYG2( 6,IBODY))
            IF(INUM.EQ. 7) CALL UTINPF('BODYG2( 7)',BODYG2( 7,IBODY))
            IF(INUM.EQ. 8) CALL UTINPF('BODYG2( 8)',BODYG2( 8,IBODY))
            IF(INUM.EQ. 9) CALL UTINPF('BODYG2( 9)',BODYG2( 9,IBODY))
            IF(INUM.EQ.10) CALL UTINPF('BODYG2(10)',BODYG2(10,IBODY))
            IF(INUM.EQ.11) CALL UTINPF('BODYG2(11)',BODYG2(11,IBODY))
C
C.....EXAMINE/CHANGE LEVELS
C
         ELSEIF(ITYPE.EQ.5) THEN
            CALL UTINPI('LEVEL NUMBER',ILEVEL)
            ILEVEL=MAX(-MLVLG2,MIN(MLVLG2,ILEVEL))
C
            WRITE(JTERMO,70) ILEVEL,
     &                       ILVLG2( 1,ILEVEL),ILVLG2( 2,ILEVEL),
     &                       ILVLG2( 3,ILEVEL),ILVLG2( 4,ILEVEL),
     &                       ILVLG2( 5,ILEVEL),ILVLG2( 6,ILEVEL)
70          FORMAT(' LEVEL ',I6                                          /
     &             '  1. ILVLG2( 1)',I15  ,5X,'  2. ILVLG2( 2)',I15  /
     &             '  3. ILVLG2( 3)',I15  ,5X,'  4. ILVLG2( 4)',I15  /
     &             '  5. ILVLG2( 5)',I15  ,5X,'  6. ILVLG2( 6)',I15  )
C
            CALL UTINPI('VARIABLE NUMBER (OR 0. TO EXIT)',INUM)
            IF(INUM.EQ. 1) CALL UTINPI('ILVLG2( 1)',ILVLG2( 1,ILEVEL))
            IF(INUM.EQ. 2) CALL UTINPI('ILVLG2( 2)',ILVLG2( 2,ILEVEL))
            IF(INUM.EQ. 3) CALL UTINPI('ILVLG2( 3)',ILVLG2( 3,ILEVEL))
            IF(INUM.EQ. 4) CALL UTINPI('ILVLG2( 4)',ILVLG2( 4,ILEVEL))
            IF(INUM.EQ. 5) CALL UTINPI('ILVLG2( 5)',ILVLG2( 5,ILEVEL))
            IF(INUM.EQ. 6) CALL UTINPI('ILVLG2( 6)',ILVLG2( 6,ILEVEL))
C
C.....SEARCH NODES
C
         ELSEIF(ITYPE.EQ.6) THEN
            CALL UTINPF('X-SEARCH',XSRCH)
            CALL UTINPF('Y-SEARCH',YSRCH)
C
            SMIN =(GEOMG2(1,1)-XSRCH)**2+(GEOMG2(2,1)-YSRCH)**2
            ISMIN=1
            DO 80 INODE=2,NNODG2
            STEST=(GEOMG2(1,INODE)-XSRCH)**2+(GEOMG2(2,INODE)-YSRCH)**2
            IF(STEST.LT.SMIN) THEN
              SMIN =STEST
              ISMIN=INODE
            ENDIF
80          CONTINUE
C
            WRITE(JTERMO,90) ISMIN,SQRT(SMIN)
90          FORMAT(' CLOSEST NODE IS ',I5,'  DS=',F10.5)
C
```

```
C.....SEARCH CELLS
C
        ELSEIF(ITYPE.EQ.7) THEN
           CALL UTINPI('INODE',INODE)
C
           DO 110 ICELL=1,NCELG2
           CORNER='  '
           IF(ICELG2(1,ICELL).EQ.INODE) CORNER='C '
           IF(ICELG2(2,ICELL).EQ.INODE) CORNER='SW'
           IF(ICELG2(3,ICELL).EQ.INODE) CORNER='S '
           IF(ICELG2(4,ICELL).EQ.INODE) CORNER='SE'
           IF(ICELG2(5,ICELL).EQ.INODE) CORNER='E '
           IF(ICELG2(6,ICELL).EQ.INODE) CORNER='NE'
           IF(ICELG2(7,ICELL).EQ.INODE) CORNER='N '
           IF(ICELG2(8,ICELL).EQ.INODE) CORNER='NW'
           IF(ICELG2(9,ICELL).EQ.INODE) CORNER='W '
           IF(CORNER.NE.'  ') WRITE(JTERMO,100) CORNER,ICELL,INODE
100        FORMAT(' THE ',A,' NODE OF CELL ',I5,' REFERENCES NODE ',I5)
110        CONTINUE
C
        ENDIF
C
        GOTO 10
C
        END
```

G2FUSE

```
        SUBROUTINE G2FUSE(LCELL,
     &
     &                    ICHNG)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE FUSES THE FOUR SUBCELLS WHICH MAKE UP
C        CELL 'LCELL' AND PERFORMS ALL NECESSARY POINTER SYSTEM
C        REALIGNMENTS
C
C**********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C**********************************************************************
C
        ICHNG=0
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
```

```
        IF(INBRG2.EQ.0) THEN
           CALL UTEROR(+1,REAL(INBRG2),0.0)
        ENDIF
C
C     SEARCH ALL PERTINENT LEVELS FOR LCELL
C
        DO 10 ILEVEL=0,MLVLG2-1
        KLEVEL=ILEVEL
        IF(LCELL.GE.ILVLG2(1,KLEVEL) .AND.
     &     LCELL.LE.ILVLG2(6,KLEVEL)      ) GOTO 20
10      CONTINUE
C
C     ERROR EXIT 1 (CELL NOT FOUND)
C
        CALL UTEROR(+2,REAL(LCELL),0.0)
C
C     PICK UP POINTERS TO NODES
C
20      KCENT=ICELG2( 1,LCELL)
        KSW  =ICELG2( 2,LCELL)
        KS   =ICELG2( 3,LCELL)
        KSE  =ICELG2( 4,LCELL)
        KE   =ICELG2( 5,LCELL)
        KNE  =ICELG2( 6,LCELL)
        KN   =ICELG2( 7,LCELL)
        KNW  =ICELG2( 8,LCELL)
        KW   =ICELG2( 9,LCELL)
        KAUX =ICELG2(10,LCELL)
C
C     HAS CELL BEEN PREVIOUSLY DIVIDED OR FUSED?
C
        IF(KCENT.EQ.0 .OR. KSW.EQ.0) RETURN
C
        KSSAVE=KS
        KESAVE=KE
        KNSAVE=KN
        KWSAVE=KW
C
C     FIND COARSE CELLS WHICH BOUND DIVIDED CELL (LL INDICATES
C        NEIGHBOR ACROSS A SLIT)
C
C                    *            *
C             LNW    *     LN     *    LNE
C                    *            *
C        ****************************************
C                    *            *
C             LW     *    LCELL   *    LE
C                    *            *
C        ****************************************
C                    *            *
C             LSW    *     LS     *    LSE
C                    *            *
C
```

```
      LSW=    NBORG2(2,LCELL)
      LLS=ABS(NBORG2(3,LCELL))
      LSE=    NBORG2(4,LCELL)
      LLE=ABS(NBORG2(5,LCELL))
      LNE=    NBORG2(6,LCELL)
      LLN=ABS(NBORG2(7,LCELL))
      LNW=    NBORG2(8,LCELL)
      LLW=ABS(NBORG2(9,LCELL))
C
      LS=MAX(NBORG2(3,LCELL),0)
      LE=MAX(NBORG2(5,LCELL),0)
      LN=MAX(NBORG2(7,LCELL),0)
      LW=MAX(NBORG2(9,LCELL),0)
C
C     FIND FINE CELLS WHICH BOUND DIVIDED CELL
C
C                  *           *
C                  *           *
C           LNWF   * LNFW  LNFE *   LNEF
C        ************************************
C           LWFN   * LFNW  LFNE *   LEFN
C                  *           *
C           LWFS   * LFSW  LFSE *   LEFS
C        ************************************
C           LSWF   * LSFW  LSFE *   LSEF
C                  *           *
C                  *           *
C
      LFSW=NBORG2(1,LCELL)
      LFSE=NBORG2(5, LFSW)
      LFNE=NBORG2(7, LFSE)
      LFNW=NBORG2(9, LFNE)
C
      IF(NBORG2(3,LFNW).NE.LFSW) THEN
         CALL UTEROR(-3,REAL(LFSW),REAL(LFSE))
         CALL UTEROR(+3,REAL(LFNE),REAL(LFNW))
      ENDIF
C
      LSWF =0
      LLSFW=0
      LLSFE=0
      LSEF =0
      LLEFS=0
      LLEFN=0
      LNEF =0
      LLNFE=0
      LLNFW=0
      LNWF =0
      LLWFN=0
      LLWFS=0
C
      IF(LSW.NE.0 .AND. NBORG2(1,LSW).NE.0) LSWF =NBORG2(2,LFSW)
      IF(LLS.NE.0 .AND. NBORG2(1,LLS).NE.0) LLSFW=NBORG2(3,LFSW)
```

```
      IF(LLS.NE.0 .AND. NBORG2(1,LLS).NE.0) LLSFE=NBORG2(3,LFSE)
      IF(LSE.NE.0 .AND. NBORG2(1,LSE).NE.0) LSEF =NBORG2(4,LFSE)
      IF(LLE.NE.0 .AND. NBORG2(1,LLE).NE.0) LLEFS=NBORG2(5,LFSE)
      IF(LLE.NE.0 .AND. NBORG2(1,LLE).NE.0) LLEFN=NBORG2(5,LFNE)
      IF(LNE.NE.0 .AND. NBORG2(1,LNE).NE.0) LNEF =NBORG2(6,LFNE)
      IF(LLN.NE.0 .AND. NBORG2(1,LLN).NE.0) LLNFE=NBORG2(7,LFNE)
      IF(LLN.NE.0 .AND. NBORG2(1,LLN).NE.0) LLNFW=NBORG2(7,LFNW)
      IF(LNW.NE.0 .AND. NBORG2(1,LNW).NE.0) LNWF =NBORG2(8,LFNW)
      IF(LLW.NE.0 .AND. NBORG2(1,LLW).NE.0) LLWFN=NBORG2(9,LFNW)
      IF(LLW.NE.0 .AND. NBORG2(1,LLW).NE.0) LLWFS=NBORG2(9,LFSW)
C
      IF(LS.EQ.LLS) THEN
         LSFW=LLSFW
         LSFE=LLSFE
      ELSE
         LSFW=0
         LSFE=0
         IF(LLSFW.NE.0 .AND. ICELG2(8,LLSFW).NE.KSW) LLSFW=0
         IF(LLSFE.NE.0 .AND. ICELG2(6,LLSFE).NE.KSE) LLSFE=0
      ENDIF
C
      IF(LE.EQ.LLE) THEN
         LEFS=LLEFS
         LEFN=LLEFN
      ELSE
         LEFS=0
         LEFN=0
         IF(LLEFS.NE.0 .AND. ICELG2(2,LLEFS).NE.KSE) LLEFS=0
         IF(LLEFN.NE.0 .AND. ICELG2(8,LLEFN).NE.KNE) LLEFN=0
      ENDIF
C
      IF(LN.EQ.LLN) THEN
         LNFE=LLNFE
         LNFW=LLNFW
      ELSE
         LNFE=0
         LNFW=0
         IF(LLNFE.NE.0 .AND. ICELG2(4,LLNFE).NE.KNE) LLNFE=0
         IF(LLNFW.NE.0 .AND. ICELG2(2,LLNFW).NE.KNW) LLNFW=0
      ENDIF
C
      IF(LW.EQ.LLW) THEN
         LWFN=LLWFN
         LWFS=LLWFS
      ELSE
         LWFN=0
         LWFS=0
         IF(LLWFN.NE.0 .AND. ICELG2(6,LLWFN).NE.KNW) LLWFN=0
         IF(LLWFS.NE.0 .AND. ICELG2(4,LLWFS).NE.KSW) LLWFS=0
      ENDIF
C
C     ARE ANY OF THE SUB-CELLS SUBDIVIDED?
C
```

```
      IF(                ICELG2(1,LFSW ).NE.0) GOTO 65
      IF(                ICELG2(1,LFSE ).NE.0) GOTO 65
      IF(                ICELG2(1,LFNE ).NE.0) GOTO 65
      IF(                ICELG2(1,LFNW ).NE.0) GOTO 65
      IF(LSWF .NE.0 .AND. ICELG2(1,LSWF ).NE.0) GOTO 65
      IF(LLSFW.NE.0 .AND. ICELG2(1,LLSFW).NE.0) GOTO 65
      IF(LLSFE.NE.0 .AND. ICELG2(1,LLSFE).NE.0) GOTO 65
      IF(LSEF .NE.0 .AND. ICELG2(1,LSEF ).NE.0) GOTO 65
      IF(LLEFS.NE.0 .AND. ICELG2(1,LLEFS).NE.0) GOTO 65
      IF(LLEFN.NE.0 .AND. ICELG2(1,LLEFN).NE.0) GOTO 65
      IF(LNEF .NE.0 .AND. ICELG2(1,LNEF ).NE.0) GOTO 65
      IF(LLNFE.NE.0 .AND. ICELG2(1,LLNFE).NE.0) GOTO 65
      IF(LLNFW.NE.0 .AND. ICELG2(1,LLNFW).NE.0) GOTO 65
      IF(LNWF .NE.0 .AND. ICELG2(1,LNWF ).NE.0) GOTO 65
      IF(LLWFN.NE.0 .AND. ICELG2(1,LLWFN).NE.0) GOTO 65
      IF(LLWFS.NE.0 .AND. ICELG2(1,LLWFS).NE.0) GOTO 65
C
      GOTO 70
C
C     A NEIGHBOR'S SUBCELL WAS ITSELF DIVIDED, SO JUST RETURN
C
65    RETURN
C
C     MARK NODE AT CENTER OF CELL FOR DELETION
C
70    DPENG2(1,KCENT)=-99.
C
C     MARK SOUTHERN NODE FOR DELETION IF LS IS NOT DIVIDED
C
      IF(LSFW.EQ.0) THEN
        DPENG2(1,KS)=-99.
        KS=0
      ENDIF
C
C     MARK EASTERN NODE FOR DELETION IF LE IS NOT DIVIDED
C
      IF(LEFS.EQ.0) THEN
        DPENG2(1,KE)=-99.
        KE=0
      ENDIF
C
C     MARK NORTHERN NODE FOR DELETION IF LN IS NOT DIVIDED
C
      IF(LNFE.EQ.0) THEN
        DPENG2(1,KN)=-99.
        KN=0
      ENDIF
C
C     MARK WESTERN NODE FOR DELETION IF LW IS NOT DIVIDED
C
      IF(LWFN.EQ.0) THEN
        DPENG2(1,KW)=-99.
        KW=0
```

```
        ENDIF
C
C     MARK THE FOUR FUSED CELLS FOR DELETE
C
        ICELG2(2,LFSW)=0
        ICELG2(2,LFSE)=0
        ICELG2(2,LFNE)=0
        ICELG2(2,LFNW)=0
C
C     UPDATE THE DIVIDED CELL
C
        ICELG2(1,LCELL)=0
        ICELG2(3,LCELL)=KS
        ICELG2(5,LCELL)=KE
        ICELG2(7,LCELL)=KN
        ICELG2(9,LCELL)=KW
C
C     RESET EDGE NODE POINTERS OF ALL NEIGHBORING CELLS
C
        IF(LS.NE.0) ICELG2(7,LS)=KS
        IF(LE.NE.0) ICELG2(9,LE)=KE
        IF(LN.NE.0) ICELG2(3,LN)=KN
        IF(LW.NE.0) ICELG2(5,LW)=KW
C
C     SKIP NEXT SECTION OF CODE IF LCELL IS NOT A BOUNDARY CELL
C
        IF(IAND(KAUX,HLOOOF).EQ.0) GOTO 440
C
C     SCAN THROUGH ALL BOUNDARY CONDITION POINTERS, LOOKING FOR POINTERS
C       TO THE DIVIDED CELL
C
        ILEFT=0
        ILFTC=0
        ICORN=0
        ICENT=0
        IRITC=0
        IRITE=0
C
C     BRANCH OUT DEPENDING ON BOUNDARY CONDITION TYPE
C
        IGOTO=IAND(KAUX,HLOOOF)+1
        GOTO (430,350,370,210,390,430,250,230,
     &        410,330,430,190,290,310,270,430), IGOTO
C
C     ...SOUTHWESTERN CORNER (OUTSIDE)
C
190     DO 200 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KSE   ) ILEFT=IBND
        IF(IBNDG2(1,IBND).EQ.KSSAVE) ILFTC=IBND
        IF(IBNDG2(1,IBND).EQ.KSW   ) ICORN=IBND
        IF(IBNDG2(1,IBND).EQ.KWSAVE) IRITC=IBND
        IF(IBNDG2(1,IBND).EQ.KNW   ) IRITE=IBND
200     CONTINUE
```

```
C
      IF(ILEFT.EQ.0 .OR. ILFTC.EQ.0 .OR. ICORN.EQ.0
     &               .OR. IRITC.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
        IBNDG2(1,ILFTC)=0
        IBNDG2(1,IRITC)=0
C
C     REASSIGN POINTERS
C
        IBNDG2(2,ILEFT)=LCELL
        IBNDG2(2,ICORN)=LCELL
        IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
C
        IBNDG2(6,ILEFT)=KLEVEL
        IBNDG2(6,ICORN)=KLEVEL
        IBNDG2(6,IRITE)=KLEVEL
C
        GOTO 440
C
C     ...SOUTHERN SIDE
C
210     DO 220 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KSE   ) ILEFT=IBND
        IF(IBNDG2(1,IBND).EQ.KSSAVE) ICENT=IBND
        IF(IBNDG2(1,IBND).EQ.KSW   ) IRITE=IBND
220     CONTINUE
C
        IF(ILEFT.EQ.0 .OR. ICENT.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
        IBNDG2(1,ICENT)=0
C
C     REASSIGN POINTERS
C
        IBNDG2(2,ILEFT)=LCELL
        IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
C
        IBNDG2(6,ILEFT)=KLEVEL
        IBNDG2(6,IRITE)=KLEVEL
C
        GOTO 440
C
C     ...SOUTHEASTERN CORNER (OUTSIDE)
C
230     DO 240 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KNE   ) ILEFT=IBND
```

```
        IF(IBNDG2(1,IBND).EQ.KFSAVE) ILFTC=IBND
        IF(IBNDG2(1,IBND).EQ.KSE   ) ICORN=IBND
        IF(IBNDG2(1,IBND).EQ.KSSAVE) IRITC=IBND
        IF(IBNDG2(1,IBND).EQ.KSW   ) IRITE=IBND
240     CONTINUE
C
        IF(ILEFT.EQ.0 .OR. ILFTC.EQ.0 .OR. ICORN.EQ.0
     &                .OR. IRITC.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
        IBNDG2(1,ILFTC)=0
        IBNDG2(1,IRITC)=0
C
C     REASSIGN POINTERS
C
        IBNDG2(2,ILEFT)=LCELL
        IBNDG2(2,ICORN)=LCELL
        IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
C
        IBNDG2(6,ILEFT)=KLEVEL
        IBNDG2(6,ICORN)=KLEVEL
        IBNDG2(6,IRITE)=KLEVEL
C
        GOTO 440
C
C     ...EASTERN SIDE
C
250     DO 260 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KSE   ) ILEFT=IBND
        IF(IBNDG2(1,IBND).EQ.KESAVE) ICENT=IBND
        IF(IBNDG2(1,IBND).EQ.KSE   ) IRITE=IBND
260     CONTINUE
C
        IF(ILEFT.EQ.0 .OR. ICENT.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
        IBNDG2(1,ICENT)=0
C
C     REASSIGN POINTERS
C
        IBNDG2(2,ILEFT)=LCELL
        IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
C
        IBNDG2(6,ILEFT)=KLEVEL
        IBNDG2(6,IRITE)=KLEVEL
C
        GOTO 440
```

```
C
C     ...NORTHEASTERN CORNER (OUTSIDE)
C
270     DO 280 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KNW   ) ILEFT=IBND
        IF(IBNDG2(1,IBND).EQ.KNSAVE) ILFTC=IBND
        IF(IBNDG2(1,IBND).EQ.KNE   ) ICORN=IBND
        IF(IBNDG2(1,IBND).EQ.KESAVE) IRITC=IBND
        IF(IBNDG2(1,IBND).EQ.KSE   ) IRITE=IBND
280     CONTINUE
C
        IF(ILEFT.EQ.0 .OR. ILFTC.EQ.0 .OR. ICORN.EQ.0
     &                .OR. IRITC.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
        IBNDG2(1,ILFTC)=0
        IBNDG2(1,IRITC)=0
C
C     REASSIGN POINTERS
C
        IBNDG2(2,ILEFT)=LCELL
        IBNDG2(2,ICORN)=LCELL
        IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
C
        IBNDG2(6,ILEFT)=KLEVEL
        IBNDG2(6,ICORN)=KLEVEL
        IBNDG2(6,IRITE)=KLEVEL
C
        GOTO 440
C
C     ...NORTHERN SIDE
C
290     DO 300 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KNW   ) ILEFT=IBND
        IF(IBNDG2(1,IBND).EQ.KNSAVE) ICENT=IBND
        IF(IBNDG2(1,IBND).EQ.KNE   ) IRITE=IBND
300     CONTINUE
C
        IF(ILEFT.EQ.0 .OR. ICENT.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
        IBNDG2(1,ICENT)=0
C
C     REASSIGN POINTERS
C
        IBNDG2(2,ILEFT)=LCELL
        IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
```

```
C
        IBNDG2(6,ILEFT)=KLEVEL
        IBNDG2(6,IRITE)=KLEVEL
C
        GOTO 440
C
C     ...NORTHWESTERN CORNER (OUTSIDE)
C
310     DO 320 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KSW   ) ILEFT=IBND
        IF(IBNDG2(1,IBND).EQ.KWSAVE) ILFTC=IBND
        IF(IBNDG2(1,IBND).EQ.KNW   ) ICORN=IBND
        IF(IBNDG2(1,IBND).EQ.KNSAVE) IRITC=IBND
        IF(IBNDG2(1,IBND).EQ.KNE   ) IRITE=IBND
320     CONTINUE
C
        IF(ILEFT.EQ.0 .OR. ILFTC.EQ.0 .OR. ICORN.EQ.0
     &                .OR. IRITC.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
          IBNDG2(1,ILFTC)=0
          IBNDG2(1,IRITC)=0
C
C     REASSIGN POINTERS
C
          IBNDG2(2,ILEFT)=LCELL
          IBNDG2(2,ICORN)=LCELL
          IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
C
          IBNDG2(5,ILEFT)=KLEVEL
          IBNDG2(6,ICORN)=KLEVEL
          IBNDG2(6,IRITE)=KLEVEL
C
          GOTO 440
C
C     ...WESTERN SIDE
C
330     DO 340 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KSW   ) ILEFT=IBND
        IF(IBNDG2(1,IBND).EQ.KWSAVE) ICENT=IBND
        IF(IBNDG2(1,IBND).EQ.KNW   ) IRITE=IBND
340     CONTINUE
C
        IF(ILEFT.EQ.0 .OR. ICENT.EQ.0 .OR. IRITE.EQ.0) GOTO 430
C
C     MARK FOR DELETE
C
          IBNDG2(1,ICENT)=0
C
C     REASSIGN POINTERS
```

```
C
        IBNDG2(2,ILEFT)=LCELL
        IBNDG2(3,IRITE)=LCELL
C
C     RESET LEVEL INDICATORS
C
        IBNDG2(6,ILEFT)=KLEVEL
        IBNDG2(6,IRITE)=KLEVEL
C
        GOTO 440
C
C     ...SOUTHWESTERN CORNER (INSIDE)
C
350     DO 360 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KSW) ICORN=IBND
360     CONTINUE
C
        IF(ICORN.EQ.0) GOTO 430
C
C     RESET LEVEL INDICATOR
C
        IBNDG2(6,ICORN)=KLEVEL
C
        GOTO 440
C
C     ...SOUTHEASTERN CORNER (INSIDE)
C
370     DO 380 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KSE) ICORN=IBND
380     CONTINUE
C
        IF(ICORN.EQ.0) GOTO 430
C
C     RESET LEVEL INDICATOR
C
        IBNDG2(6,ICORN)=KLEVEL
C
        GOTO 440
C
C     ...NORTHEASTERN CORNER (INSIDE)
C
390     DO 400 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KNE) ICORN=IBND
400     CONTINUE
C
        IF(ICORN.EQ.0) GOTO 430
C
C     RESET LEVEL INDICATOR
C
        IBNDG2(6,ICORN)=KLEVEL
C
        GOTO 440
C
```

```
C     ...NORTHWESTERN CORNER (INSIDE)
C
410     DO 420 IBND=1,NBNDG2
        IF(IBNDG2(1,IBND).EQ.KNW) ICORN=IBND
420     CONTINUE
C
        IF(ICORN.EQ.0) GOTO 430
C
C     RESET LEVEL INDICATOR
C
        IBNDG2(6,ICORN)=KLEVEL
C
        GOTO 440
C
C     ERROR IN BOUNDARY CELL POINTERS
C
430     CALL UTEROR(+4,REAL(LCELL),REAL(IGOTO))
C
C     RESET NEIGHBOR POINTERS FOR ALL CELLS
C
440     NBORG2(1,LCELL)=0
C
        IF(LLWFN.NE.0) NBORG2(4,LLWFN)=0
        IF(LLWFS.NE.0) NBORG2(5,LLWFS)=0
        IF(LSWF .NE.0) NBORG2(6,LSWF )=0
        IF(LLSFW.NE.0) NBORG2(7,LLSFW)=0
        IF(LLSFE.NE.0) NBORG2(8,LLSFE)=0
C
        IF(LLEFN.NE.0) NBORG2(2,LLEFN)=0
        IF(LLSFW.NE.0) NBORG2(6,LLSFW)=0
        IF(LSFE .NE.0) NBORG2(7,LSFE )=0
        IF(LLSEF.NE.0) NBORG2(8,LLSEF)=0
        IF(LLEFS.NE.0) NBORG2(9,LLEFS)=0
C
        IF(LLNEF.NE.0) NBORG2(2,LLNEF)=0
        IF(LLNFE.NE.0) NBORG2(3,LLNFE)=0
        IF(LNFW .NE.0) NBORG2(4,LNFW )=0
        IF(LLEFS.NE.0) NBORG2(8,LLEFS)=0
        IF(LLEFN.NE.0) NBORG2(9,LLEFN)=0
C
        IF(LLNFE.NE.0) NBORG2(2,LLNFE)=0
        IF(LLNFW.NE.0) NBORG2(3,LLNFW)=0
        IF(LNWF .NE.0) NBORG2(4,LNWF )=0
        IF(LLWFN.NE.0) NBORG2(5,LLWFN)=0
        IF(LLWFS.NE.0) NBORG2(6,LLWFS)=0
C
C     RESET ALL POINTERS SAYING CORNER IS JUST OUTSIDE EMBEDDED MESH
C
        ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFFOF)
C
        IF(LSW.NE.0) ICELG2(10,LSW  )=IAND(ICELG2(10,LSW  ),HLFFBF)
        IF(LS .NE.0) ICELG2(10,LS   )=IAND(ICELG2(10,LS   ),HLFF3F)
        IF(LSE.NE.0) ICELG2(10,LSE  )=IAND(ICELG2(10,LSE  ),HLFF7F)
```

```
      IF(LE .NE.0) ICELG2(10,LE   )=IAND(ICELG2(10,LE   ),HLFF8F)
      IF(LNE.NE.0) ICELG2(10,LNE  )=IAND(ICELG2(10,LNE  ),HLFFEF)
      IF(LN .NE.0) ICELG2(10,LN   )=IAND(ICELG2(10,LN   ),HLFFCF)
      IF(LNW.NE.0) ICELG2(10,LNW  )=IAND(ICELG2(10,LNW  ),HLFFDF)
      IF(LW .NE.0) ICELG2(10,LW   )=IAND(ICELG2(10,LW   ),HLFF9F)
C
      IF(LS.NE.LLS) THEN
         IF(ICELG2(8,LLS).EQ.KSW)
     &               ICELG2(10,LLS)=IAND(ICELG2(10,LLS),HLFF7F)
         IF(ICELG2(6,LLS).EQ.KSE)
     &               ICELG2(10,LLS)=IAND(ICELG2(10,LLS),HLFFBF)
      ENDIF
C
      IF(LE.NE.LLE) THEN
         IF(ICELG2(2,LLE).EQ.KSE)
     &               ICELG2(10,LLE)=IAND(ICELG2(10,LLE),HLFFEF)
         IF(ICELG2(8,LLE).EQ.KNE)
     &               ICELG2(10,LLE)=IAND(ICELG2(10,LLE),HLFF7F)
      ENDIF
C
      IF(LN.NE.LLN) THEN
         IF(ICELG2(4,LLN).EQ.KNE)
     &               ICELG2(10,LLN)=IAND(ICELG2(10,LLN),HLFFDF)
         IF(ICELG2(2,LLN).EQ.KNW)
     &               ICELG2(10,LLN)=IAND(ICELG2(10,LLN),HLFFEF)
      ENDIF
C
      IF(LW.NE.LLW) THEN
         IF(ICELG2(6,LLW).EQ.KNW)
     &               ICELG2(10,LLW)=IAND(ICELG2(10,LLW),HLFFBF)
         IF(ICELG2(4,LLW).EQ.KSW)
     &               ICELG2(10,LLW)=IAND(ICELG2(10,LLW),HLFFDF)
      ENDIF
C
C     SET ALL OUTSIDE POINTERS FOR SOUTHWEST CORNER
C
      IF(( LSWF.NE.0 .OR. LLSFW.NE.0) .AND.          LLWFS.EQ.0
     &                                .AND.            LLW.NE.0
     &                                .AND. ICELG2(4,LLW).EQ.KSW)
     &                  ICELG2(10,LLW  )=IOR(ICELG2(10,LLW  ),HL0020)
      IF((LLWFS.NE.0 .OR. LLSFW.NE.0) .AND.          LSWF.EQ.0
     &                                .AND.            LSW.NE.0  )
     &                  ICELG2(10,LSW  )=IOR(ICELG2(10,LSW  ),HL0040)
      IF((LLWFS.NE.0 .OR.  LSWF.NE.0) .AND.          LLSFW.EQ.0
     &                                .AND.            LLS.NE.0
     &                                .AND. ICELG2(8,LLS).EQ.KSW)
     &                  ICELG2(10,LLS  )=IOR(ICELG2(10,LLS  ),HL0080)
      IF( LLWFS.NE.0 .OR. LSWF.NE.0 .OR. LLSFW.NE.0)
     &                  ICELG2(10,LCELL)=IOR(ICELG2(10,LCELL),HL0010)
C
C     SET ALL OUTSIDE POINTERS FOR SOUTHEAST CORNER
C
      IF(( LSEF.NE.0 .OR. LLEFS.NE.0) .AND.          LLSFE.EQ.0
```

```
     &                                  .AND.           LLS.NE.0
     &                                  .AND. ICELG2(6,LLS).EQ.KSE)
     &                     ICELG2(10,LLS  )=IOR(ICELG2(10,LLS  ),HL0040)
        IF((LLSFE.NE.0 .OR. LLEFS.NE.0) .AND.          LSEF.EQ.0
     &                                  .AND.           LSE.NE.0  )
     &                     ICELG2(10,LSE  )=IOR(ICELG2(10,LSE  ),HL0080)
        IF((LLSFE.NE.0 .OR.  LSEF.NE.0) .AND.          LLEFS.EQ.0
     &                                  .AND.           LLE.NE.0
     &                                  .AND. ICELG2(2,LLE).EQ.KSE)
     &                     ICELG2(10,LLE  )=IOR(ICELG2(10,LLE  ),HL0010)
        IF( LLSFE.NE.0 .OR. LSEF.NE.0 .OR. LLEFS.NE.0)
     &                     ICELG2(10,LCELL)=IOR(ICELG2(10,LCELL),HL0020)
C
C     SET ALL OUTSIDE POINTERS FOR NORTHEAST CORNER
C
        IF(( LNEF.NE.0 .OR. LLNFE.NE.0) .AND.          LLEFN.EQ.0
     &                                  .AND.           LLE.NE.0
     &                                  .AND. ICELG2(8,LLE).EQ.KNE)
     &                     ICELG2(10,LLE  )=IOR(ICELG2(10,LLE  ),HL0080)
        IF((LLEFN.NE.0 .OR. LLNFE.NE.0) .AND.          LNEF.EQ.0
     &                                  .AND.           LNE.NE.0  )
     &                     ICELG2(10,LNE  )=IOR(ICELG2(10,LNE  ),HL0010)
        IF((LLEFN.NE.0 .OR.  LNEF.NE.0) .AND.          LLNFE.EQ.0
     &                                  .AND.           LLN.NE.0
     &                                  .AND. ICELG2(4,LLN).EQ.KNE)
     &                     ICELG2(10,LLN  )=IOR(ICELG2(10,LLN  ),HL0020)
        IF( LLEFN.NE.0 .OR. LNEF.NE.0 .OR. LLNFE.NE.0)
     &                     ICELG2(10,LCELL)=IOR(ICELG2(10,LCELL),HL0040)
C
C     SET ALL OUTSIDE POINTERS FOR NORTHWEST CORNER
C
        IF(( LNWF.NE.0 .OR. LLWFN.NE.0) .AND.          LLNFW.EQ.0
     &                                  .AND.           LLN.NE.0
     &                                  .AND. ICELG2(2,LLN).EQ.KNW)
     &                     ICELG2(10,LLN  )=IOR(ICELG2(10,LLN  ),HL0010)
        IF((LLNFW.NE.0 .OR. LLWFN.NE.0) .AND.          LNWF.EQ.0
     &                                  .AND.           LNW.NE.0  )
     &                     ICELG2(10,LNW  )=IOR(ICELG2(10,LNW  ),HL0020)
        IF((LLNFW.NE.0 .OR.  LNWF.NE.0) .AND.          LLWFN.EQ.0
     &                                  .AND.           LLW.NE.0
     &                                  .AND. ICELG2(6,LLW).EQ.KNW)
     &                     ICELG2(10,LLW  )=IOR(ICELG2(10,LLW  ),HL0040)
        IF( LLNFW.NE.0 .OR. LNWF.NE.0 .OR. LLWFN.NE.0)
     &                     ICELG2(10,LCELL)=IOR(ICELG2(10,LCELL),HL0080)
C
C     CELL LCELL IS NO LONGER INSIDE
C
        ICELG2(10,LCELL)=IAND(ICELG2(10,LCELL),HLFOFF)
C
C     MOVE AROUND LCELL, SETING JUST INSIDE (COARSE) POINTERS FOR ANY
C        OF LCELL'S NEIGHBORS WHICH IS DIVIDED
C
        IF(LSWF .NE.0) ICELG2(10,LSW)=IOR(ICELG2(10,LSW),HL0400)
```

```
      IF(LLSFW.NE.0) ICELG2(10,LLS)=IOR(ICELG2(10,LLS),HL0800)
      IF(LLSFE.NE.0) ICELG2(10,LLS)=IOR(ICELG2(10,LLS),HL0400)
      IF(LSEF .NE.0) ICELG2(10,LSE)=IOR(ICELG2(10,LSE),HL0800)
      IF(LLEFS.NE.0) ICELG2(10,LLE)=IOR(ICELG2(10,LLE),HL0100)
      IF(LLEFN.NE.0) ICELG2(10,LLE)=IOR(ICELG2(10,LLE),HL0800)
      IF(LNEF .NE.0) ICELG2(10,LNE)=IOR(ICELG2(10,LNE),HL0100)
      IF(LLNFE.NE.0) ICELG2(10,LLN)=IOR(ICELG2(10,LLN),HL0200)
      IF(LLNFW.NE.0) ICELG2(10,LLN)=IOR(ICELG2(10,LLN),HL0100)
      IF(LNWF .NE.0) ICELG2(10,LNW)=IOR(ICELG2(10,LNW),HL0200)
      IF(LLWFN.NE.0) ICELG2(10,LLW)=IOR(ICELG2(10,LLW),HL0400)
      IF(LLWFS.NE.0) ICELG2(10,LLW)=IOR(ICELG2(10,LLW),HL0200)
C
C     MOVE AROUND CELL, SETTING JUST INSIDE (FINE) POINTERS FOR ANY
C        OF FINE CELLS WHICH EXIST
C
      IF(LSWF .NE.0) ICELG2(10,LSWF )=IOR(ICELG2(10,LSWF ),HL4000)
      IF(LLSFW.NE.0) ICELG2(10,LLSFW)=IOR(ICELG2(10,LLSFW),HLC000)
      IF(LLSFE.NE.0) ICELG2(10,LLSFE)=IOR(ICELG2(10,LLSFE),HLC000)
      IF(LSEF .NE.0) ICELG2(10,LSEF )=IOR(ICELG2(10,LSEF ),HL8000)
      IF(LLEFS.NE.0) ICELG2(10,LLEFS)=IOR(ICELG2(10,LLEFS),HL9000)
      IF(LLEFN.NE.0) ICELG2(10,LLEFN)=IOR(ICELG2(10,LLEFN),HL9000)
      IF(LNEF .NE.0) ICELG2(10,LNEF )=IOR(ICELG2(10,LNEF ),HL1000)
      IF(LLNFE.NE.0) ICELG2(10,LLNFE)=IOR(ICELG2(10,LLNFE),HL3000)
      IF(LLNFW.NE.0) ICELG2(10,LLNFW)=IOR(ICELG2(10,LLNFW),HL3000)
      IF(LNWF .NE.0) ICELG2(10,LNWF )=IOR(ICELG2(10,LNWF ),HL2000)
      IF(LLWFN.NE.0) ICELG2(10,LLWFN)=IOR(ICELG2(10,LLWFN),HL6000)
      IF(LLWFS.NE.0) ICELG2(10,LLWFS)=IOR(ICELG2(10,LLWFS),HL6000)
C
C     CORRECT THESE IF A NEIGHBOR ACROSS A SLIT
C
      IF(LS.NE.LLS) THEN
         IF(LLSFW.NE.0) ICELG2(10,LLSFW)=IAND(ICELG2(10,LLSFW),HLBFFF)
         IF(LLSFE.NE.0) ICELG2(10,LLSFE)=IAND(ICELG2(10,LLSFE),HL7FFF)
      ENDIF
C
      IF(LE.NE.LLE) THEN
         IF(LLEFS.NE.0) ICELG2(10,LLEFS)=IAND(ICELG2(10,LLEFS),HL7FFF)
         IF(LLEFN.NE.0) ICELG2(10,LLEFN)=IAND(ICELG2(10,LLEFN),HLEFFF)
      ENDIF
C
      IF(LN.NE.LLN) THEN
         IF(LLNFE.NE.0) ICELG2(10,LLNFE)=IAND(ICELG2(10,LLNFE),HLEFFF)
         IF(LLNFW.NE.0) ICELG2(10,LLNFW)=IAND(ICELG2(10,LLNFW),HLDFFF)
      ENDIF
C
      IF(LW.NE.LLW) THEN
         IF(LLWFN.NE.0) ICELG2(10,LLWFN)=IAND(ICELG2(10,LLWFN),HLDFFF)
         IF(LLWFS.NE.0) ICELG2(10,LLWFS)=IAND(ICELG2(10,LLWFS),HLBFFF)
      ENDIF
C
      ICHNG=1
      RETURN
C
```

```
      END
```

G2GRBG

```
      SUBROUTINE G2GRBG
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE COLLECTS ALL NODES AND BOUNDARY CONDITION
C       POINTERS MARKED FOR DELETE BY SUBROUTINE G2FUSE AND RECLAIMS
C       SPACE INSERTED BY G2XPND
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.UTILITY]UTCOMN.INC'
C
      DIMENSION ILTEMP(0:MLVLG2+1)
C
C*********************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
      IF(INBRG2.EQ.0) THEN
        CALL UTEROR(+1,REAL(INBRG2),0.0)
      ENDIF
C
C     CREATE TEMPORARY ARRAY FOR NODES
C
      CALL UTBASE(+1,NNODG2+1,IBASE)
      NBASE=IBASE+1
      VARUT(NBASE)=0.
C
C     MOVE NODE ARRAYS TO OVERLAY DELETED NODES, KEEPING TRACK OF
C       THE MAPPING IN VARUT
C
      NNEW=0
C
      DO 10 NOLD=1,NNODG2
      IF(DPENG2(1,NOLD).EQ.-99.) THEN
        VARUT(NBASE+NOLD)=0.
      ELSE
        NNEW=NNEW+1
        VARUT(NBASE+NOLD)=NNEW
C
        IF(NOLD.NE.NNEW) THEN
          GEOMG2(1,NNEW)=GEOMG2(1,NOLD)
          GEOMG2(2,NNEW)=GEOMG2(2,NOLD)
          DPENG2(1,NNEW)=DPENG2(1,NOLD)
```

```
            DPENG2(2,NNEW)=DPENG2(2,NOLD)
            DPENG2(3,NNEW)=DPENG2(3,NOLD)
            DPENG2(4,NNEW)=DPENG2(4,NOLD)
            DPENG2(5,NNEW)=DPENG2(5,NOLD)
          ENDIF
C
        ENDIF
10      CONTINUE
C
C     RESET NUMBER OF NODES
C
        NNODG2=NNEW
C
C     DELETE ALL BOUNDARY CONDITION POINTERS MARKED FOR DELETE
C
        NNEW=0
C
        DO 20 NOLD=1,NBNDG2
        IF(IBNDG2(1,NOLD).EQ.0) GOTO 20
C
        NNEW=NNEW+1
C
C     MOVE POINTER INFORMATION
C
        IF(NOLD.EQ.NNEW) GOTO 20
C
        IBNDG2(1,NNEW)=IBNDG2(1,NOLD)
        IBNDG2(2,NNEW)=IBNDG2(2,NOLD)
        IBNDG2(3,NNEW)=IBNDG2(3,NOLD)
        IBNDG2(4,NNEW)=IBNDG2(4,NOLD)
        IBNDG2(5,NNEW)=IBNDG2(5,NOLD)
        IBNDG2(6,NNEW)=IBNDG2(6,NOLD)
C
        BONDG2(1,NNEW)=BONDG2(1,NOLD)
        BONDG2(2,NNEW)=BONDG2(2,NOLD)
        BONDG2(3,NNEW)=BONDG2(3,NOLD)
        BONDG2(4,NNEW)=BONDG2(4,NOLD)
        BONDG2(5,NNEW)=BONDG2(5,NOLD)
        BONDG2(6,NNEW)=BONDG2(6,NOLD)
        BONDG2(7,NNEW)=BONDG2(7,NOLD)
        BONDG2(8,NNEW)=BONDG2(8,NOLD)
        BONDG2(9,NNEW)=BONDG2(9,NOLD)
C
20      CONTINUE
C
C     RESET NUMBER OF BOUNDARY CONDITION POINTERS
C
        NBNDG2=NNEW
C
C     STEP THROUGH ALL CELL POINTERS, REALIGNING TO NEW NODE NUMBERS
C
        DO 30 ICELL=ILVLG2(1,0),ILVLG2(6,MLVLG2)
        ICELG2(1,ICELL)=NINT(VARUT(NBASE+ICELG2(1,ICELL)))
```

```
        ICELG2(2,ICELL)=NINT(VARUT(NBASE+ICELG2(2,ICELL)))
        ICELG2(3,ICELL)=NINT(VARUT(NBASE+ICELG2(3,ICELL)))
        ICELG2(4,ICELL)=NINT(VARUT(NBASE+ICELG2(4,ICELL)))
        ICELG2(5,ICELL)=NINT(VARUT(NBASE+ICELG2(5,ICELL)))
        ICELG2(6,ICELL)=NINT(VARUT(NBASE+ICELG2(6,ICELL)))
        ICELG2(7,ICELL)=NINT(VARUT(NBASE+ICELG2(7,ICELL)))
        ICELG2(8,ICELL)=NINT(VARUT(NBASE+ICELG2(8,ICELL)))
        ICELG2(9,ICELL)=NINT(VARUT(NBASE+ICELG2(9,ICELL)))
30      CONTINUE
C
C     STEP THROUGH ALL BOUNDARY CONDITION POINTERS, REALIGNING TO
C        NEW NODE NUMBERS
C
        DO 40 IBOUND=1,NBNDG2
        IBNDG2(1,IBOUND)=NINT(VARUT(NBASE+IBNDG2(1,IBOUND)))
40      CONTINUE
C
C     RELEASE BASED VARIABLES FOR NODE MOVEMENT
C
        CALL UTBASE(-1,0,IBASE)
C
C     COUNT THE NUMBER OF CELLS ON EACH FINE GRID LEVEL
C
        DO 45 IMGL=0,MLVLG2+1
        ILTEMP(IMGL)=0
C
        DO 45 ICELL=ILVLG2(1,IMGL),ILVLG2(6,IMGL)
        IF(ICELG2(2,ICELL).NE.0) ILTEMP(IMGL)=ILTEMP(IMGL)+1
C
45      CONTINUE
C
C     CREATE TEMPORARY ARRAY FOR CELLS
C
        CALL UTBASE(+1,(ILVLG2(6,MLVLG2)+1),IBASE)
        NBASE=IBASE+1
        VARUT(NBASE)=0.
C
C     MOVE CELL ARRAYS TO OVERLAY DELETED CELLS, KEEPING TRACK
C        OF THE MAPPING IN VARUT
C
        NNEW=0
C
        DO 50 NOLD=1,ILVLG2(6,MLVLG2)
        IF(ICELG2(2,NOLD).EQ.0) THEN
           VARUT(NBASE+NOLD)=0.
        ELSE
           NNEW=NNEW+1
           VARUT(NBASE+NOLD)=NNEW
C
           IF(NOLD.NE.NNEW) THEN
              ICELG2( 1,NNEW)=ICELG2( 1,NOLD)
              ICELG2( 2,NNEW)=ICELG2( 2,NOLD)
              ICELG2( 3,NNEW)=ICELG2( 3,NOLD)
```

```
            ICELG2( 4,NNEW)=ICELG2( 4,NOLD)
            ICELG2( 5,NNEW)=ICELG2( 5,NOLD)
            ICELG2( 6,NNEW)=ICELG2( 6,NOLD)
            ICELG2( 7,NNEW)=ICELG2( 7,NOLD)
            ICELG2( 8,NNEW)=ICELG2( 8,NOLD)
            ICELG2( 9,NNEW)=ICELG2( 9,NOLD)
            ICELG2(10,NNEW)=ICELG2(10,NOLD)
            NBORG2( 1,NNEW)=NBORG2( 1,NOLD)
            NBORG2( 2,NNEW)=NBORG2( 2,NOLD)
            NBORG2( 3,NNEW)=NBORG2( 3,NOLD)
            NBORG2( 4,NNEW)=NBORG2( 4,NOLD)
            NBORG2( 5,NNEW)=NBORG2( 5,NOLD)
            NBORG2( 6,NNEW)=NBORG2( 6,NOLD)
            NBORG2( 7,NNEW)=NBORG2( 7,NOLD)
            NBORG2( 8,NNEW)=NBORG2( 8,NOLD)
            NBORG2( 9,NNEW)=NBORG2( 9,NOLD)
          ENDIF
C
        ENDIF
50      CONTINUE
C
C     RESET NUMBER OF CELLS
C
        NCELG2=NNEW
C
C     RECOMPUTE THE MULTIPLE-GRID LEVEL ENTRIES
C
        DO 55 IMGL=0,MLVLG2+1
        ILVLG2(1,IMGL)=ILVLG2(6,IMGL-1)+1
        ILVLG2(2,IMGL)=ILVLG2(1,IMGL)+ILTEMP(IMGL)-1
        ILVLG2(3,IMGL)=ILVLG2(2,IMGL)+1
        ILVLG2(4,IMGL)=ILVLG2(2,IMGL)
        ILVLG2(5,IMGL)=ILVLG2(2,IMGL)+1
        ILVLG2(6,IMGL)=ILVLG2(2,IMGL)
55      CONTINUE
C
C     STEP THROUGH ALL NEIGHBOR POINTERS, REALIGNING TO NEW CELL
C       NUMBERS
C
        DO 60 ICELL=ILVLG2(1,0),ILVLG2(6,MLVLG2)
        NBORG2(1,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(1,ICELL)))),
     &                                              NBORG2(1,ICELL)   )
        NBORG2(2,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(2,ICELL)))),
     &                                              NBORG2(2,ICELL)   )
        NBORG2(3,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(3,ICELL)))),
     &                                              NBORG2(3,ICELL)   )
        NBORG2(4,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(4,ICELL)))),
     &                                              NBORG2(4,ICELL)   )
        NBORG2(5,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(5,ICELL)))),
     &                                              NBORG2(5,ICELL)   )
        NBORG2(6,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(6,ICELL)))),
     &                                              NBORG2(6,ICELL)   )
        NBORG2(7,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(7,ICELL)))),
```

```
     &                                                NBORC2(7,ICELL)   )
        NBORG2(8,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(8,ICELL)))),
     &                                                NBORG2(8,ICELL)   )
        NBORG2(9,ICELL)=SIGN(NINT(VARUT(NBASE+ABS(NBORG2(9,ICELL)))),
     &                                                NBORG2(9,ICELL)   )
60      CONTINUE
C
C     STEP THROUGH ALL BOUNDARY CONDITION POINTERS, REALIGNING TO
C        NEW CELL NUMBERS
C
        DO 70 IBOUND=1,NBNDG2
        IBNDG2(2,IBOUND)=NINT(VARUT(NBASE+IBNDG2(2,IBOUND)))
        IBNDG2(3,IBOUND)=NINT(VARUT(NBASE+IBNDG2(3,IBOUND)))
70      CONTINUE
C
C     RELEASE THE BASED VARIABLES FOR THE CELLS
C
        CALL UTBASE(-1,0,IBASE)
C
C     SORT THE CELLS IN THE DATA STRUCTURE
C
        CALL G2SORT
C
        RETURN
        END
```

G2GROW

```
        SUBROUTINE G2GROW(NGROW,ITYPE,
     &
     &                    NCHNG      )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GROWS THE EMBEDDED REGIONS
C        IF ITYPE =0 EACH REGION GROWS NGROW LEVEL-0 CELLS
C                 =1 EACH REGION GROWS NGROW         CELLS
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************************
C
        NCHNG=0
C
C     LOOP THROUGH ALL EMBEDDED LEVELS
C
```

```
      DO 40 ILEVEL=0,MLVLG2-1
      IF(ILVLG2(1,ILEVEL).GT.ILVLG2(6,ILEVEL)) GOTO 40
C
C     LOOP THROUGH THIS LEVEL ENOUGH TIMES TO GROW REGION NGROW LEVEL-O
C       CELLS
C
      IF(ITYPE.EQ.0) THEN
        KGROW=NGROW*(2**ILEVEL)
      ELSE
        KGROW=NGROW
      ENDIF
C
      DO 30 IGROW=1,KGROW
C
C     LOOP THROUGH ALL CELLS, MARKING THOSE THAT ARE JUST OUTSIDE AN
C       EMBEDDED REGION
C
      DO 10 ICELL=ILVLG2(1,ILEVEL),ILVLG2(6,ILEVEL)
      IF(ICELG2(1,ICELL).NE.0) GOTO 10
C
      IF(IAND(ICELG2(10,ICELL),HL00F0).NE.0) THEN
        ICELG2(10,ICELL)= IOR(ICELG2(10,ICELL),HU8000)
      ELSE
        ICELG2(10,ICELL)=IAND(ICELG2(10,ICELL),HU7FFF)
      ENDIF
C
10    CONTINUE
C
C     LOOP THROUGH ALL CELLS, DIVIDING THOSE THAT HAVE JUST BEEN MARKED
C
      DO 20 ICELL=ILVLG2(1,ILEVEL),ILVLG2(6,ILEVEL)
      IF(ICELG2(1,ICELL).NE.0) GOTO 20
C
      IF(IAND(ICELG2(10,ICELL),HU8000).NE.0) THEN
        ICELG2(10,ICELL)=IAND(ICELG2(10,ICELL),HU7FFF)
        CALL G2DIVD(ICELL,ICHNG)
        NCHNG=NCHNG+ICHNG
      ENDIF
C
20    CONTINUE
C
C     NEXT GROWTH CYCLE AT THIS LEVEL
C
30    CONTINUE
C
C     NEXT EMBEDDED LEVEL
C
40    CONTINUE
C
C     ENSURE THAT THERE ARE NO VOIDS OR ISLANDS
C
      CALL G2VOID(ICHNG)
      NCHNG=NCHNG+ICHNG
```

```
C
C     GARBAGE COLLECT
C
        CALL G2GRBG
C
        RETURN
        END
```

G2INP0

```
        SUBROUTINE G2INP0(IUNIT,ITYPE
     &
     &                                )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 0 GEOMETRY DEFINITION, WHICH
C       IS SIMPLY A RE-START FILE
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMM.INC'
C
        INCLUDE '[.GRAFIC]GRCOMM.INC'
C
        CHARACTER*9 IDATEN
        CHARACTER*8 ITIMEN
C
C*****************************************************************************
C
        IF(ITYPE.EQ.1) GOTO 120
C
C*****FORMATTED FILE
C
C     READ TITLE-TYPE INFORMATION
C
        READ(IUNIT,10) NNODG2,NCELG2,NBEDG2,NBODG2,NDEFG2,
     &                 IDATEN,ITIMEN
10      FORMAT(5I5,25X,2A)
C
C     IF THIS IS A RESTART CASE, USE THE OLD DATE AND TIME
C
        IF(IDATEN.NE.'         ') THEN
           DATEGR=IDATEN
           TIMEGR=ITIMEN
        ENDIF
C
C     READ ALL DATA STRUCTURE VARIABLES
C
        DO 30 INODE=1,NNODG2
```

```
      READ(IUNIT,20) (GEOMG2(K,INODE),K=1,2),
     &               (DPENG2(K,INODE),K=1,5)
20    FORMAT(5X,7G15.7)
30    CONTINUE
C
      DO 50 ICELL=1,NCELG2
      READ(IUNIT,40) (ICELG2(I,ICELL),I=1,10),
     &               (NBORG2(I,ICELL),I=1, 9)
40    FORMAT(5X,9I5,5X,Z10,9I6)
50    CONTINUE
C
      DO 70 ILEVEL=-MLVLG2,MLVLG2
      READ(IUNIT,60) (ILVLG2(I,ILEVEL),I=1,6)
60    FORMAT(5X,3I5)
70    CONTINUE
C
      ILVLG2(1,MLVLG2+1)=ILVLG2(6,MLVLG2)+1
      ILVLG2(2,MLVLG2+1)=ILVLG2(6,MLVLG2)
      ILVLG2(3,MLVLG2+1)=ILVLG2(6,MLVLG2)+1
      ILVLG2(4,MLVLG2+1)=ILVLG2(6,MLVLG2)
      ILVLG2(5,MLVLG2+1)=ILVLG2(6,MLVLG2)+1
      ILVLG2(6,MLVLG2+1)=ILVLG2(6,MLVLG2)
C
      DO 90 IBOUND=1,NBNDG2
      READ(IUNIT,80) (IBNDG2(K,IBOUND),K=1,6),(BONDG2(K,IBOUND),K=1,9)
80    FORMAT(5X,6I5,4G15.7/35X,5G15.7)
90    CONTINUE
C
      DO 110 IBODY=1,NBODG2
      READ(IUNIT,100) (BODYG2(K,IBODY),K=1,11)
100   FORMAT(5X,6G15.7/20X,5G15.7)
110   CONTINUE
C
      RETURN
C
C*****UNFORMATTED FILE
C
120   READ(IUNIT) NNODG2,NCELG2,NBNDG2,NBODG2,KDEFG2,
     &            DATEGR,TIMEGR
C
      READ(IUNIT) ((GEOMG2(K,INODE),K=1,2),
     &             (DPENG2(K,INODE),K=1,5),INODE=1,NNODG2)
C
      READ(IUNIT) ((ICELG2(K,ICELL),K=1,10),
     &             (NBORG2(K,ICELL),K=1, 9),ICELL=1,NCELG2)
C
      READ(IUNIT) ((ILVLG2(K,ILEVEL),K=1,6),ILEVEL=-MLVLG2,MLVLG2)
C
      READ(IUNIT) ((IBNDG2(K,IBOUND),K=1,6),
     &             (BONDG2(K,IBOUND),K=1,9),IBOUND=1,NBNDG2)
C
      READ(IUNIT) ((BODYG2(K,IBODY),K=1,11),IBODY=1,NBODG2)
C
```

```
      RETURN
C
      END
```

G2INP1

```
      SUBROUTINE G2INP1(IUNIT,IOUT
     &
     &                              )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 1 GEOMETRY DEFINITION, WHICH
C       IS A SIMPLY-CONNECTED RECTANGLE.  THE RECTANGLE'S SIDES
C       ARE READ IN IN THE ORDER:
C          SOUTH (W->E)
C          NORTH (W->E)
C          WEST  (S->N)
C          EAST  (S->N)
C       AFTER THE BOUNDARY DEFINITIONS READ IN AND AN INITIAL GRID IS
C          GENERATED ALGEBRAICALLY, THE DATA STRUCTURE IS INITAILIZED
C          AND THE GRID IS SMOOTHED BY USE OF A LAPLACE GRID GENERATOR
C
C*****************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************************
C
C     READ GLOBAL MESH SIZE AND OTHER CONTROLLING PARAMETERS
C
      READ(IUNIT,10) NXNODE,NYNODE,KDEFG2,NITER ,NCOARS,
     &               IBCSW ,IBCS  ,IBCSE ,IBCE  ,
     &               IBCNE ,IBCN  ,IBCNW ,IBCW
10    FORMAT(13I6)
C
C     COMPUTE NUMBER OF POINTS INPUT IN EACH DIRECTION
C
      NFACT =2**KDEFG2
      NXBODY=1+(NXNODE-1)*NFACT
      NYBODY=1+(NYNODE-1)*NFACT
C
C     COMPUTE TOTAL NUMBER OF GLOBAL NODES
C
      NNODG2=NXNODE*NYNODE
C
C     COMPUTE NUMBER OF CELLS IN EACH DIRECTION ON THE GLOBAL MESH
C
```

```
        NXCELL=NXNODE-1
        NYCELL=NYNODE-1
C
C     SET UP THE SPLINE COEFFICIENTS
C
        ISBEG=     +1
        ISEND=     +NXBODY
        INBEG=ISEND+1
        INEND=ISEND+NXBODY
        IEBEG=INEND+1
        IEEND=INEND+NYBODY
        IWBEG=IEEND+1
        IWEND=IEEND+NYBODY
C
C*****READ IN ALL BOUNDARY INFORMATION
C
C     ...READ THE SOUTHERN BOUNDARY AND SPLINE FIT IT
C
        DO 30 IPNT=1,NXBODY
        IBODY=ISBEG-1+IPNT
        INODE=(IPNT-1)/NFACT+1
C
        READ(IUNIT,20) (BODYG2(K,IBODY),K=1,7)
20      FORMAT(7F10.0)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=BODYG2(3,IBODY)
           DPENG2(2,INODE)=BODYG2(4,IBODY)
           DPENG2(3,INODE)=BODYG2(5,IBODY)
           DPENG2(4,INODE)=BODYG2(6,IBODY)
           DPENG2(5,INODE)=BODYG2(7,IBODY)
        ENDIF
C
30      CONTINUE
C
        CALL G2BOND(ISBEG,ISEND,1)
C
C     ...READ THE NORTHERN BOUNDARY AND SPLINE FIT IT
C
        DO 40 IPNT=1,NXBODY
        IBODY=INEND+1-IPNT
        INODE=(IPNT-1)/NFACT+1+NXNODE*(NYNODE-1)
C
        READ(IUNIT,20) (BODYG2(K,IBODY),K=1,7)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=BODYG2(3,IBODY)
           DPENG2(2,INODE)=BODYG2(4,IBODY)
           DPENG2(3,INODE)=BODYG2(5,IBODY)
```

```
            DPENG2(4,INODE)=BODYG2(6,IBODY)
            DPENG2(5,INODE)=BODYG2(7,IBODY)
         ENDIF
C
40       CONTINUE
C
         CALL G2BOND(INBEG,INEND,1)
C
C     ...READ THE WESTERN BOUNDARY AND SPLINE FIT IT
C
         DO 50 IPNT=1,NYBODY
         IBODY=IWEND+1-IPNT
         INODE=((IPNT-1)/NFACT+1)*NXNODE+1-NXNODE
C
         READ(IUNIT,20) (BODYG2(K,IBODY),K=1,7)
C
         IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
            GEOMG2(1,INODE)=BODYG2(1,IBODY)
            GEOMG2(2,INODE)=BODYG2(2,IBODY)
            DPENG2(1,INODE)=BODYG2(3,IBODY)
            DPENG2(2,INODE)=BODYG2(4,IBODY)
            DPENG2(3,INODE)=BODYG2(5,IBODY)
            DPENG2(4,INODE)=BODYG2(6,IBODY)
            DPENG2(5,INODE)=BODYG2(7,IBODY)
         ENDIF
C
         CALL G2BOND(IWBEG,IWEND,1)
C
50       CONTINUE
C
C     ...READ THE EASTERN BOUNDARY AND SPLINE FIT IT
C
         DO 60 IPNT=1,NYBODY
         IBODY=IEBEG-1+IPNT
         INODE=((IPNT-1)/NFACT+1)*NXNODE
C
         READ(IUNIT,20) (BODYG2(K,IBODY),K=1,7)
C
         IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
            GEOMG2(1,INODE)=BODYG2(1,IBODY)
            GEOMG2(2,INODE)=BODYG2(2,IBODY)
            DPENG2(1,INODE)=BODYG2(3,IBODY)
            DPENG2(2,INODE)=BODYG2(4,IBODY)
            DPENG2(3,INODE)=BODYG2(5,IBODY)
            DPENG2(4,INODE)=BODYG2(6,IBODY)
            DPENG2(5,INODE)=BODYG2(7,IBODY)
         ENDIF
C
60       CONTINUE
C
         CALL G2BOND(IEBEG,IEEND,1)
C
         NBODG2=2*(NXBODY+NYBODY)
```

```
C
C*****ALGEBRAIC GRID GENERATION
C
C     CALCULATE THE TOTAL ARCLENGTH ALONG THE WEST AND EAST BOUNDARIES
C
        SWEST=0.0
        SEAST=0.0
C
        DO 70 J=2,NYNODE
        INDXJ =1+(J-1)*NXNODE
        INDXJM=1+(J-2)*NXNODE
        SWEST=SWEST+SQRT((GEOMG2(1,INDXJ)-GEOMG2(1,INDXJM))**2
     &                 +(GEOMG2(2,INDXJ)-GEOMG2(2,INDXJM))**2)
C
        INDXJ =(J  )*NXNODE
        INDXJM=(J-1)*NXNODE
        SEAST=SEAST+SQRT((GEOMG2(1,INDXJ)-GEOMG2(1,INDXJM))**2
     &                 +(GEOMG2(2,INDXJ)-GEOMG2(2,INDXJM))**2)
70      CONTINUE
C
C     STEP THROUGH EACH INTERIOR SOUTH->NORTH LINE
C
        DO 90 I=2,NXNODE-1
C
C     CALCULATE FRACTIONAL DISTANCES FOR EACH INTERIOR POINT
C
        TWEST=0.0
        TEAST=0.0
C
        DO 80 J=2,NYNODE-1
        INDXJ =1+(J-1)*NXNODE
        INDXJM=1+(J-2)*NXNODE
        TWEST=TWEST+SQRT((GEOMG2(1,INDXJ)-GEOMG2(1,INDXJM))**2
     &                 +(GEOMG2(2,INDXJ)-GEOMG2(2,INDXJM))**2)
C
        INDXJ =(J  )*NXNODE
        INDXJM=(J-1)*NXNODE
        TEAST=TEAST+SQRT((GEOMG2(1,INDXJ)-GEOMG2(1,INDXJM))**2
     &                 +(GEOMG2(2,INDXJ)-GEOMG2(2,INDXJM))**2)
C
        FRAC=(TWEST/SWEST)
     &        +(I-1.0)/(NXNODE-1.0)*((TEAST/SEAST)-(TWEST/SWEST))
C
        INDX =I+(     J-1)*NXNODE
        INDXN=I+(NYNODE-1)*NXNODE
        INDXS=I
C
C     COMPUTE LOCATION OF INTERIOR POINT AND ALL OTHER NODAL VARIABLES
C
        GEOMG2(1,INDX)=GEOMG2(1,INDXS)
     &                 +FRAC*(GEOMG2(1,INDXN)-GEOMG2(1,INDXS))
        GEOMG2(2,INDX)=GEOMG2(2,INDXS)
     &                 +FRAC*(GEOMG2(2,INDXN)-GEOMG2(2,INDXS))
```

```
C
        DPENG2(1,INDX)=DPENG2(1,INDXS)
     &                       +FRAC*(DPENG2(1,INDXN)-DPENG2(1,INDXS))
        DPENG2(2,INDX)=DPENG2(2,INDXS)
     &                       +FRAC*(DPENG2(2,INDXN)-DPENG2(2,INDXS))
        DPENG2(3,INDX)=DPENG2(3,INDXS)
     &                       +FRAC*(DPENG2(3,INDXN)-DPENG2(3,INDXS))
        DPENG2(4,INDX)=DPENG2(4,INDXS)
     &                       +FRAC*(DPENG2(4,INDXN)-DPENG2(4,INDXS))
        DPENG2(5,INDX)=DPENG2(5,INDXS)
     &                       +FRAC*(DPENG2(5,INDXN)-DPENG2(5,INDXS))
C
80      CONTINUE
90      CONTINUE
C
C*****SET UP POINTER SYSTEM (DATA STRUCTURE) FOR CELLS ON ALL LEVELS
C
C     INITIALIZE THE NUMBER OF CELLS AND BOUNDARY CONDITION POINTERS
C
        NCELG2=0
        NBNDG2=0
C
C     INITIALIZE POINTERS FOR ALL LEVELS
C
        DO 100 ILEVEL=-MLVLG2,MLVLG2
        ILVLG2(1,ILEVEL)=1
        ILVLG2(2,ILEVEL)=0
        ILVLG2(3,ILEVEL)=1
        ILVLG2(4,ILEVEL)=0
        ILVLG2(5,ILEVEL)=1
        ILVLG2(6,ILEVEL)=0
100     CONTINUE
C
C     LOOP THROUGH ALL COARSER GRID LEVELS (IF ANY)
C
        DO 120 ICOARS=MIN(NCOARS,MLVLG2-1),1,-1
        ISIZE=2**ICOARS
C
C     LOOP THROUGH EACH CELL ON THIS LEVEL
C
        DO 110 JCELL=1,NYCELL,ISIZE
        DO 110 ICELL=1,NXCELL,ISIZE
C
        NCELG2=NCELG2+1
C
C     FIND THE CENTER OF THIS CELL
C
        ICELG2(1,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE/2-1)*NXNODE
C
C     COMPUTE INDICES OF ALL BOUNDING NODES
C
        ICELG2(2,NCELG2)=(ICELL        )+(JCELL        -1)*NXNODE
        ICELG2(3,NCELG2)=(ICELL+ISIZE/2)+(JCELL        -1)*NXNODE
```

```
        ICELG2(4,NCELG2)=(ICELL+ISIZE  )+(JCELL        -1)*NXNODE
        ICELG2(5,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE/2-1)*NXNODE
        ICELG2(6,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(7,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(8,NCELG2)=(ICELL        )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(9,NCELG2)=(ICELL        )+(JCELL+ISIZE/2-1)*NXNODE
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C      THE NEIGHBOR INFORMATION IS NOT USED ON COARSE GLOBAL LEVELS,
C        SO SET THEM ALL TO ZERO
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=0
        NBORG2(3,NCELG2)=0
        NBORG2(4,NCELG2)=0
        NBORG2(5,NCELG2)=0
        NBORG2(6,NCELG2)=0
        NBORG2(7,NCELG2)=0
        NBORG2(8,NCELG2)=0
        NBORG2(9,NCELG2)=0
C
110     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THIS LEVEL
C
        ILEVEL=-ICOARS
        ILVLG2(1,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(2,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(3,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(4,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(5,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(6,ILEVEL)=NCELG2
C
C     GO BACK FOR A FINER COARSE LEVEL
C
120     CONTINUE
C
C     LOOP THROUGH EACH GLOBAL CELL
C
        IOFFN=NCELG2-NXCELL
C
        DO 130 JCELL=1,NYCELL
        DO 130 ICELL=1,NXCELL
C
        NCELG2=NCELG2+1
C
C     COMPUTE INDICES OF EACH CORNER OF CELL
C
        ICELG2(2,NCELG2)=ICELL  +(JCELL-1)*NXNODE
        ICELG2(4,NCELG2)=ICELL+1+(JCELL-1)*NXNODE
```

```
        ICELG2(6,NCELG2)=ICELL+1+(JCELL  )*NXNODE
        ICELG2(8,NCELG2)=ICELL  +(JCELL  )*NXNODE
C
C     INITIALLY, THERE IS NO NODE IN THE CENTER OF A FINE CELL
C
        ICELG2(1,NCELG2)=0
C
C     THERE ARE NO NODES IN THE CENTER OF THE SIDES OF A FINE CELL
C
        ICELG2(3,NCELG2)=0
        ICELG2(5,NCELG2)=0
        ICELG2(7,NCELG2)=0
        ICELG2(9,NCELG2)=0
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C     INITIALIZE ALL NEIGHBOR INFORMATION FOR THIS LEVEL
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL-1)
        NBORG2(3,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL  )
        NBORG2(4,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL+1)
        NBORG2(5,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL+1)
        NBORG2(6,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL+1)
        NBORG2(7,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL  )
        NBORG2(8,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL-1)
        NBORG2(9,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL-1)
C
C     CORRECT NEIGHBOR INFORMATION FOR BOUNDARY CELLS
C
        IF(ICELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(8,NCELG2)=0
           NBORG2(9,NCELG2)=0
        ENDIF
C
        IF(ICELL.EQ.NXCELL) THEN
           NBORG2(4,NCELG2)=0
           NBORG2(5,NCELG2)=0
           NBORG2(6,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(3,NCELG2)=0
           NBORG2(4,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYCELL) THEN
           NBORG2(6,NCELG2)=0
           NBORG2(7,NCELG2)=0
```

```
          NBORG2(8,NCELG2)=0
        ENDIF
C
130     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THE GLOBAL FINE LEVEL
C
        ILVLG2(1,0)=ILVLG2(6,-1)+1
        ILVLG2(2,0)=NCELG2
        ILVLG2(3,0)=NCELG2+1
        ILVLG2(4,0)=NCELG2
        ILVLG2(5,0)=NCELG2+1
        ILVLG2(6,0)=NCELG2
C
C     INITIALIZE THE MULTIPLE-GRID-LEVEL ARRAY FOR ALL EMBEDDED MESHES
C
        DO 140 ILEVEL=1,MLVLG2+1
        ILVLG2(1,ILEVEL)=NCELG2+1
        ILVLG2(2,ILEVEL)=NCELG2
        ILVLG2(3,ILEVEL)=NCELG2+1
        ILVLG2(4,ILEVEL)=NCELG2
        ILVLG2(5,ILEVEL)=NCELG2+1
        ILVLG2(6,ILEVEL)=NCELG2
140     CONTINUE
C
C*****ADD BOUNDARY CONDITION INFORMATION TO POINTER SYSTEM
C
C     SET UP THE BOUNDARY CONDITION ARRAYS AND SET BOUNDARY POINTERS FOR
C        BOUNDARY NODES...
C         ...FOR SOUTHWESTERN CORNER
C
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=1
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+1
        IBNDG2(3,NBNDG2)=0
        IBNDG2(4,NBNDG2)=11
        IBNDG2(5,NBNDG2)=IBCSW
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IWEND)
        BONDG2(2,NBNDG2)=BODYG2( 9,IWEND)
        BONDG2(3,NBNDG2)=BODYG2(10,ISBEG)
        BONDG2(4,NBNDG2)=BODYG2(11,ISBEG)
        BONDG2(5,NBNDG2)=BODYG2( 3,ISBEG)
        BONDG2(6,NBNDG2)=BODYG2( 4,ISBEG)
        BONDG2(7,NBNDG2)=BODYG2( 5,ISBEG)
        BONDG2(8,NBNDG2)=BODYG2( 6,ISBEG)
        BONDG2(9,NBNDG2)=BODYG2( 7,ISBEG)
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HLOCOB)
C
```

```
C         ...FOR SOUTHERN EDGE
C
          DO 150 IBOUND=2,NXNODE-1
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=IBOUND
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+IBOUND-1
          IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+IBOUND
          IBNDG2(4,NBNDG2)=3
          IBNDG2(5,NBNDG2)=IBCS
          IBNDG2(6,NBNDG2)=0
C
          IBODY=(IBOUND-1)+NFACT+ISBEG
          BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
          BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
          BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
          BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
          BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
          BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
          BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
          BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
          BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
          ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
          ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
150       CONTINUE
C
C         ...FOR SOUTHEASTERN CORNER
C
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=NXNODE
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL
          IBNDG2(3,NBNDG2)=0
          IBNDG2(4,NBNDG2)=7
          IBNDG2(5,NBNDG2)=IBCSE
          IBNDG2(6,NBNDG2)=0
C
          BONDG2(1,NBNDG2)=BODYG2( 8,ISEND)
          BONDG2(2,NBNDG2)=BODYG2( 9,ISEND)
          BONDG2(3,NBNDG2)=BODYG2(10,IEBEG)
          BONDG2(4,NBNDG2)=BODYG2(11,IEBEG)
          BONDG2(5,NBNDG2)=BODYG2( 3,IEBEG)
          BONDG2(6,NBNDG2)=BODYG2( 4,IEBEG)
          BONDG2(7,NBNDG2)=BODYG2( 5,IEBEG)
          BONDG2(8,NBNDG2)=BODYG2( 6,IEBEG)
          BONDG2(9,NBNDG2)=BODYG2( 7,IEBEG)
C
          ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0007)
```

```
C
C        ...FOR EASTERN EDGE
C
         DO 160 IBOUND=2,NYNODE-1
         NBNDG2=NBNDG2+1
C
         IBNDG2(1,NBNDG2)=NXNODE* IBOUND
         IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)
         IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL* IBOUND
         IBNDG2(4,NBNDG2)=6
         IBNDG2(5,NBNDG2)=IBCE
         IBNDG2(6,NBNDG2)=0
C
         IBODY=(IBOUND-1)*NFACT+IEBEG
         BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
         BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
         BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
         BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
         BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
         BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
         BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
         BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
         BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
         ICELG2(10,IBNDG2(2,NBNDG2))
     &                   =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0006)
         ICELG2(10,IBNDG2(3,NBNDG2))
     &                   =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0006)
C
160      CONTINUE
C
C        ...FOR NORTHEASTERN CORNER
C
         NBNDG2=NBNDG2+1
C
         IBNDG2(1,NBNDG2)=NXNODE*NYNODE
         IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*NYCELL
         IBNDG2(3,NBNDG2)=0
         IBNDG2(4,NBNDG2)=14
         IBNDG2(5,NBNDG2)=IBCNE
         IBNDG2(6,NBNDG2)=0
C
         BONDG2(1,NBNDG2)=BODYG2( 8,IEEND)
         BONDG2(2,NBNDG2)=BODYG2( 9,IEEND)
         BONDG2(3,NBNDG2)=BODYG2(10,INBEG)
         BONDG2(4,NBNDG2)=BODYG2(11,INBEG)
         BONDG2(5,NBNDG2)=BODYG2( 3,INBEG)
         BONDG2(6,NBNDG2)=BODYG2( 4,INBEG)
         BONDG2(7,NBNDG2)=BODYG2( 5,INBEG)
         BONDG2(8,NBNDG2)=BODYG2( 6,INBEG)
         BONDG2(9,NBNDG2)=BODYG2( 7,INBEG)
C
         ICELG2(10,IBNDG2(2,NBNDG2))
```

```
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000E)
C
C         ...FOR NORTHERN EDGE
C
          DO 170 IBOUND=NXNODE-1,2,-1
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+IBOUND
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND
          IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND-1
          IBNDG2(4,NBNDG2)=12
          IBNDG2(5,NBNDG2)=IBCN
          IBNDG2(6,NBNDG2)=0
C
          IBODY=(NXNODE-IBOUND)*NFACT+INBEG
          BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
          BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
          BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
          BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
          BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
          BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
          BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
          BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
          BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
          ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
          ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
170       CONTINUE
C
C         ...FOR NORTHWESTERN CORNER
C
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+1
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+1
          IBNDG2(3,NBNDG2)=0
          IBNDG2(4,NBNDG2)=13
          IBNDG2(5,NBNDG2)=IBCNW
          IBNDG2(6,NBNDG2)=0
C
          BONDG2(1,NBNDG2)=BODYG2( 8,INEND)
          BONDG2(2,NBNDG2)=BODYG2( 9,INEND)
          BONDG2(3,NBNDG2)=BODYG2(10,IWBEG)
          BONDG2(4,NBNDG2)=BODYG2(11,IWBEG)
          BONDG2(5,NBNDG2)=BODYG2( 3,IWBEG)
          BONDG2(6,NBNDG2)=BODYG2( 4,IWBEG)
          BONDG2(7,NBNDG2)=BODYG2( 5,IWBEG)
          BONDG2(8,NBNDG2)=BODYG2( 6,IWBEG)
          BONDG2(9,NBNDG2)=BODYG2( 7,IWBEG)
C
```

```
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0000)
C
C       ...FOR WESTERN EDGE
C
      DO 180 IBOUND=NYNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(IBOUND-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)+1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-2)+1
      IBNDG2(4,NBNDG2)=9
      IBNDG2(5,NBNDG2)=IBCW
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(NYNODE-IBOUND)*NFACT+IWBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
      BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
      BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
      BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
      BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
      BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0009)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0009)
C
180   CONTINUE
C
C*****USE LAPLACE'S EQUATION FOR SMOOTH GRID VARIATION
C
      CALL G2LAPL(NITER,IOUT,0,IER)
C
C*****SORT THE CELLS
C
      CALL G2SORT
C
      RETURN
      END
```

G2INP2

```
      SUBROUTINE G2INP2(IUNIT,IOUT
     &
     &                              )
C
```

```
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 2 GEOMETRY DEFINITION, WHICH
C       IS AN O-TYPE MESH AROUND A SIMPLE-PART AIRFOIL.  THE AIRFOIL
C       IS INPUT CLOCKWISE, STARTING AT THE TRAILING EDGE
C
C*************************************************************************
C
      INCLUDE '[.GRID2D]G2COMM.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*************************************************************************
C
C     READ GLOBAL MESH SIZE AND MESH INDICATOR
C
      READ(IUNIT,10) NAIRFL,KDEFG2,NITER ,NCOARS,NSURF ,NGROW1,NGROW2,
     &               DPEN1 ,DPEN2 ,DPEN3 ,DPEN4 ,DPEN5 ,
     &               RADIUS,NYNODE
10    FORMAT(7I5/
     &       E10.0/
     &       F10.0,I5)
C
C     COMPUTE TOTAL NUMBER OF POINTS INPUT ON THE AIRFOIL
C
      NFACT =2**KDEFG2
      NXBODY=1+(NAIRFL-1)*NFACT
C
C     COMPUTE TOTAL NUMBER OF GLOBAL NODES
C
      NXNODE=NAIRFL-1
      NNODG2=NXNODE*NYNODE
C
C     COMPUTE NUMBER OF CELLS IN EACH DIRECTION ON THE GLOBAL MESH
C
      NXCELL=NXNODE
      NYCELL=NYNODE-1
C
C*****READ THE AIRFOIL SHAPE AND SPLINE FIT IT
C
      DO 30 IPNT=1,NXBODY
      IBODY=IPNT
      INODE=(IPNT-1)/NFACT+1
C
      READ(IUNIT,20) (BODYG2(K,IBODY),K=1,2)
20    FORMAT(7F10.0)
C
      IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1) .AND. IPNT.NE.NXBODY) THEN
        GEOMG2(1,INODE)=BODYG2(1,IBODY)
        GEOMG2(2,INODE)=BODYG2(2,IBODY)
      ENDIF
C
30    CONTINUE
```

```
C
        BODYG2(1,1)=BODYG2(1,NXBODY)
        BODYG2(2,1)=BODYG2(2,NXBODY)
C
        CALL G2BOND(1,NXBODY,1)
C
C*****GENERATE THE FARFIELD BOUNDARY AND SPLINE FIT IT
C
        DO 40 IPNT=1,NXBODY
        IBODY=2*NXBODY+1-IPNT
        INODE=(IPNT-1)/NFACT+1+NXNODE*(NYNODE-1)
C
        ANGLE=6.2831853*(1.0-IPNT)/(NXBODY-1.0)
        BODYG2(1,IBODY)=RADIUS*COS(ANGLE)
        BODYG2(2,IBODY)=RADIUS*SIN(ANGLE)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1) .AND. IPNT.NE.NXBODY) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
        ENDIF
C
40      CONTINUE
C
        BODYG2(1,NXBODY+1)=BODYG2(1,NXBODY+NXBODY)
        BODYG2(2,NXBODY+1)=BODYG2(2,NXBODY+NXBODY)
C
        CALL G2BOND(NXBODY+1,NXBODY+NXBODY,0)
C
        NBODG2=2*NXBODY
C
C*****SET UP INTERIOR MESH POINTS USING AN ALGEBRAIC TECHNIQUE
C
C     LOOP THROUGH EACH RADIAL LINE
C
        DO 60 I=1,NXNODE
C
C     LOOP THROUGH EACH CIRCUMFERENTIAL LINE
C
        DO 50 J=2,NYNODE-1
C
C     USE EXPONENTIAL STRETCHING FOR INITIAL GRID
C
        TERM=LOG(RADIUS)/(NYNODE-1.0)
        FRAC=(EXP((J-1.0)*TERM)-1.0)/(RADIUS-1.0)
C
        INDX =I+(     J-1)*NXNODE
        INDXN=I+(NYNODE-1)*NXNODE
        INDXS=I
C
C     COMPUTE LOCATION OF INTERIOR POINT AND ALL OTHER NODAL VARIABLES
C
        GEOMG2(1,INDX)=GEOMG2(1,INDXS)
     &                      +FRAC*(GEOMG2(1,INDXN)-GEOMG2(1,INDXS))
```

```
        GEOMG2(2,INDX)=GEOMG2(2,INDXS)
     &                       +FRAC*(GEOMG2(2,INDXN)-GEOMG2(2,INDXS))
C
50      CONTINUE
60      CONTINUE
C
C     INITIALIZE DEPENDENT VARIABLES AT ALL NODES
C
        DO 70 INODE=1,NNODG2
        DPENG2(1,INODE)=DPEN1
        DPENG2(2,INODE)=DPEN2
        DPENG2(3,INODE)=DPEN3
        DPENG2(4,INODE)=DPEN4
        DPENG2(5,INODE)=DPEN5
70      CONTINUE
C
C     INITIALIZE DEPENDENT VARIABLES AT AT BOUNDARY POINTS
C
        DO 80 IBODY=1,NBODG2
        BODYG2(3,IBODY)=DPEN1
        BODYG2(4,IBODY)=DPEN2
        BODYG2(5,IBODY)=DPEN3
        BODYG2(6,IBODY)=DPEN4
        BODYG2(7,IBODY)=DPEN5
80      CONTINUE
C
C*****INITIALIZE ALL POINTER ARRAYS
C
C     INITIALIZE THE NUMBER OF CELLS AND BOUNDARY CONDITION POINTERS
C
        NCELG2=0
        NBNDG2=0
C
C     INITIALIZE POINTERS FOR ALL LEVELS
C
        DO 90 ILEVEL=-MLVLG2,MLVLG2
        ILVLG2(1,ILEVEL)=1
        ILVLG2(2,ILEVEL)=0
        ILVLG2(3,ILEVEL)=1
        ILVLG2(4,ILEVEL)=0
        ILVLG2(5,ILEVEL)=1
        ILVLG2(6,ILEVEL)=0
90      CONTINUE
C
C     LOOP THROUGH ALL COARSER GRID LEVELS (IF ANY)
C
        DO 110 ICOARS=MIN(NCOARS,MLVLG2-1),1,-1
        ISIZE=2**ICOARS
C
C     LOOP THROUGH EACH CELL ON THIS LEVEL
C
        DO 100 JCELL=1,NYCELL,ISIZE
        DO 100 ICELL=1,NXCELL,ISIZE
```

```
C
        NCELG2=NCELG2+1
C
C     COMPUTE OFFSET FOR O-TYPE MESH WRAP AROUND
C
        IF((ICELL+ISIZE).GT.NXCELL) THEN
           IOFF=-NXCELL
        ELSE
           IOFF=0
        ENDIF
C
C     FIND THE CENTER OF THIS CELL
C
        ICELG2(1,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE/2-1)*NXNODE
C
C     COMPUTE INDICES OF ALL BOUNDING NODES
C
        ICELG2(2,NCELG2)=(ICELL        )+(JCELL        -1)*NXNODE
        ICELG2(3,NCELG2)=(ICELL+ISIZE/2)+(JCELL        -1)*NXNODE
        ICELG2(4,NCELG2)=(ICELL+ISIZE  )+(JCELL        -1)*NXNODE+IOFF
        ICELG2(5,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE/2-1)*NXNODE+IOFF
        ICELG2(6,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE  -1)*NXNODE+IOFF
        ICELG2(7,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(8,NCELG2)=(ICELL        )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(9,NCELG2)=(ICELL        )+(JCELL+ISIZE/2-1)*NXNODE
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C      THE NEIGHBOR INFORMATION IS NOT USED ON COARSE GLOBAL LEVELS,
C        SO SET THEM ALL TO ZERO
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=0
        NBORG2(3,NCELG2)=0
        NBORG2(4,NCELG2)=0
        NBORG2(5,NCELG2)=0
        NBORG2(6,NCELG2)=0
        NBORG2(7,NCELG2)=0
        NBORG2(8,NCELG2)=0
        NBORG2(9,NCELG2)=0
C
100     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THIS LEVEL
C
        ILEVEL=-ICOARS
        ILVLG2(1,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(2,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(3,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(4,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(5,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
```

```
        ILVLG2(6,ILEVEL)=NCELG2
C
C     GO BACK FOR A FINER COARSE LEVEL
C
110     CONTINUE
C
        IOFFN=NCELG2-NXCELL
C
C     LOOP THROUGH EACH GLOBAL CELL
C
        DO 120 JCELL=1,NYCELL
        DO 120 ICELL=1,NXCELL
C
        NCELG2=NCELG2+1
C
C     COMPUTE OFFSET FOR O-TYPE MESH WRAP AROUND
C
        IF(ICELL.EQ.NXCELL) THEN
           IOFF=-NXCELL
        ELSE
           IOFF=0
        ENDIF
C
C     COMPUTE INDICES OF EACH CORNER OF CELL
C
        ICELG2(2,NCELG2)=ICELL  +(JCELL-1)*NXNODE
        ICELG2(4,NCELG2)=ICELL+1+(JCELL-1)*NXNODE+IOFF
        ICELG2(6,NCELG2)=ICELL+1+(JCELL  )*NXNODE+IOFF
        ICELG2(8,NCELG2)=ICELL  +(JCELL  )*NXNODE
C
C     INITIALLY, THERE IS NO NODE IN THE CENTER OF A FINE CELL
C
        ICELG2(1,NCELG2)=0
C
C     THERE ARE NO NODES IN THE CENTER OF THE SIDES OF A FINE CELL
C
        ICELG2(3,NCELG2)=0
        ICELG2(5,NCELG2)=0
        ICELG2(7,NCELG2)=0
        ICELG2(9,NCELG2)=0
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C     INITIALIZE ALL NEIGHBOR INFORMATION FOR THIS LEVEL
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL-1)
        NBORG2(3,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL  )
        NBORG2(4,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL+1)
        NBORG2(5,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL+1)
        NBORG2(6,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL+1)
```

```
        NBORG2(7,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL  )
        NBORG2(8,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL-1)
        NBORG2(9,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL-1)
C
C     CORRECT NEIGHBOR INFORMATION FOR BOUNDARY CELLS AND CELLS
C        NEAR THE PERIODIC BOUNDARY
C
        IF(ICELL.EQ.1) THEN
           NBORG2(2,NCELG2)=NBORG2(2,NCELG2)+NXCELL
           NBORG2(8,NCELG2)=NBORG2(8,NCELG2)+NXCELL
           NBORG2(9,NCELG2)=NBORG2(9,NCELG2)+NXCELL
        ENDIF
C
        IF(ICELL.EQ.NXCELL) THEN
           NBORG2(4,NCELG2)=NBORG2(4,NCELG2)-NXCELL
           NBORG2(5,NCELG2)=NBORG2(5,NCELG2)-NXCELL
           NBORG2(6,NCELG2)=NBORG2(6,NCELG2)-NXCELL
        ENDIF
C
        IF(JCELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(3,NCELG2)=0
           NBORG2(4,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYCELL) THEN
           NBORG2(6,NCELG2)=0
           NBORG2(7,NCELG2)=0
           NBORG2(8,NCELG2)=0
        ENDIF
C
120     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THE GLOBAL FINE LEVEL
C
        ILVLG2(1,0)=ILVLG2(6,-1)+1
        ILVLG2(2,0)=NCELG2
        ILVLG2(3,0)=NCELG2+1
        ILVLG2(4,0)=NCELG2
        ILVLG2(5,0)=NCELG2+1
        ILVLG2(6,0)=NCELG2
C
C     INITIALIZE THE MULTIPLE-GRID-LEVEL ARRAY FOR ALL EMBEDDED MESHES
C
        DO 130 ILEVEL=1,MLVLG2+1
        ILVLG2(1,ILEVEL)=NCELG2+1
        ILVLG2(2,ILEVEL)=NCELG2
        ILVLG2(3,ILEVEL)=NCELG2+1
        ILVLG2(4,ILEVEL)=NCELG2
        ILVLG2(5,ILEVEL)=NCELG2+1
        ILVLG2(6,ILEVEL)=NCELG2
130     CONTINUE
C
```

```
C*****SET UP BOUNDARY ENTRIES INTO POINTER SYSTEM
C
C     SET UP THE BOUNDARY CONDITION ARRAYS AND SET BOUNDARY POINTERS FOR
C        BOUNDARY NODES...
C
C         ...FOR SOUTHERN EDGE
C
        DO 140 IBOUND=1,NXCELL
        IBODY=(IBOUND-1)*NFACT+1
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=IBOUND
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+IBOUND-1
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+IBOUND
        IBNDG2(4,NBNDG2)=3
        IBNDG2(5,NBNDG2)=3
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
        BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
        BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
        BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
        BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
        BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
        BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
        BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
        BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
        IF(IBOUND.EQ.1) THEN
           IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL
           IBNDG2(5,NBNDG2)=1
        ENDIF
C
        IF(IBOUND.EQ.1) THEN
           BONDG2(1,NBNDG2)=BODYG2( 8,NXBODY)
           BONDG2(2,NBNDG2)=BODYG2( 9,NXBODY)
        ENDIF
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
        ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
140     CONTINUE
C
C         ...FOR NORTHERN EDGE
C
        DO 150 IBOUND=NXCELL,1,-1
        IBODY=2*NXBODY-(IBOUND-1)*NFACT
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+IBOUND
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND
```

```
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND-1
        IBNDG2(4,NBNDG2)=12
        IBNDG2(5,NBNDG2)=4
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
        BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
        BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
        BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
        BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
        BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
        BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
        BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
        BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
        IF(IBOUND.EQ.1) IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*NYCELL
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HLOOOC)
        ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HLOOOC)
C
150     CONTINUE
C
C*****USE LAPLACE'S EQUATION FOR SMOOTH VARIATION
C
        CALL G2LAPL(NITER,IOUT,0,IER)
C
C     SPECIAL FIX-UP TO ENSURE THAT TRAILING EDGE CELLS ARE CONVEX
C
        INEW=NXNODE+1
        YNEWUP=GEOMG2(2,1)+(GEOMG2(1,  INEW)-GEOMG2(1,1))
     &                  *(GEOMG2(2,     2)-GEOMG2(2,1))
     &                  /(GEOMG2(1,     2)-GEOMG2(1,1))
        YNEWLO=GEOMG2(2,1)+(GEOMG2(1,  INEW)-GEOMG2(1,1))
     &                  *(GEOMG2(2,NXNODE)-GEOMG2(2,1))
     &                  /(GEOMG2(1,NXNODE)-GEOMG2(1,1))
        GEOMG2(2,INEW)=0.60*(YNEWUP+YNEWLO)
C
C     PRE-EMBED NEAR AIRFOIL SURFACE IF DESIRED
C
        DO 160 ISURF=1,NSURF
        CALL G2SURF(1,3,NCELL)
160     CONTINUE
C
C     GROW THE EMBEDDED REGION IF DESIRED
C
        IF(NGROW1.NE.0) CALL G2GROW(NGROW1,NGROW2,NCELL)
C
C*****SORT THE CELLS
C
        CALL G2SORT
C
```

```
C     CHECK THE GRID FOR NON-POSITIVE AREAS AND CONCAVITY
C
        CALL G2CHEK(IOUT,IER)
C
        RETURN
        END
```

G2INP3

```
        SUBROUTINE G2INP3(IUNIT,IOUT
     &
     &                            )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 3 GEOMETRY DEFINITION, WHICH
C        IS A C-TYPE MESH AROUND A SIMPLE-PART AIRFOIL.  THE AIRFOIL
C        IS INPUT CLOCKWISE, STARTING AT THE TRAILING EDGE
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
C*********************************************************************
        RETURN
        END
```

G2INP4

```
        SUBROUTINE G2INP4(IUNIT,IOUT
     &
     &                            )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 4 GEOMETRY DEFINITION, WHICH
C        IS A H-TYPE MESH AROUND A SIMPLE-PART AIRFOIL.  THE AIRFOIL
C        IS INPUT CLOCKWISE, STARTING AT THE TRAILING EDGE.  THE NUMBER
C        OF POINTS ON THE UPPER AND LOWER AIRFOIL SURFACES ARE ASSUMED
C        TO BE THE SAME
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
```

```
C     STATEMENT FUNCTION FOR STRETCHING FUNCTION
C
        OMG(B,I,IMAX)=(EXP(B*REAL(I)/REAL(IMAX))-1.0)
     &               /(EXP(         B          )-1.0)
C
C*****************************************************************
C
C     READ GLOBAL MESH SIZE AND MESH INDICATOR
C
        READ(IUNIT,10) NAIRFL,KDEFG2,NITER ,NCOARS,NSURF ,NGROW1,NGROW2,
     &                 DPEN1 ,DPEN2 ,DPEN3 ,DPEN4 ,DPEN5 ,
     &                 XUP   ,BXUP  ,CXUP  ,NXUP  ,
     &                 XDN   ,BXDN  ,CXDN  ,NXDN  ,
     &                 YLOUP ,YLODN ,CYLO  ,NYLO  ,
     &                 YUPUP ,YUPDN ,CYUP  ,NYUP
10      FORMAT(7I5/
     &         5F10.0/
     &         3F10.0,I5/
     &         3F10.0,I5/
     &         3F10.0,I5/
     &         3F10.0,I5)
C
C     COMPUTE TOTAL NUMBER OF POINTS INPUT ON THE AIRFOIL
C
        NFACT =2**KDEFG2
        NABODY=1+(NAIRFL-1)*NFACT
C
C     FIND THE NUMBER OF POINTS ON THE UPPER AND LOWER AIRFOIL SURFACE
C        (EXCLUDING THE LEADING AND TRAILING EDGES)
C
        NAIR=(NABODY-1)/(2*NFACT)-1
C
C     COMPUTE NUMBER OF NODES IN EACH DIRECTION
C
        NXNODE=NXUP+NAIR+NXDN
        NYNODE=NYLO+NYUP-1
C
C     COMPUTE NUMNER OF BODY POINTS IN EACH DIRECTION
C
        NXBODY=1+(NXNODE-1)*NFACT
        NYBODY=1+(NYNODE-1)*NFACT
C
C     COMPUTE TOTAL NUMBER OF GLOBAL NODES
C
        NNODG2=NXNODE*(NYLO+NYUP)-NXUP-NXDN
C
C     COMPUTE NUMBER OF CELLS IN EACH DIRECTION ON THE GLOBAL MESH
C
        NXCELL=NXNODE-1
        NYCELL=NYNODE-1
C
C     INDICES FOR SURFACE FITTING
C
```

```
        IABEG=     +1
        IAEND=     +NABODY
        ISBEG=IAEND+1
        ISEND=IAEND+NXBODY
        IEBEG=ISEND+1
        IEEND=ISEND+NYBODY
        INBEG=IEEND+1
        INEND=IEEND+NXBODY
        IWBEG=INEND+1
        IWEND=INEND+NYBODY
C
        IF(IWEND.GT.MBODG2) THEN
           CALL UTEROR(+1,REAL(IWEND),REAL(MBODG2))
        ENDIF
C
C*****READ THE AIRFOIL SHAPE AND SPLINE FIT IT
C
        IALE=(IABEG+IAEND)/2
C
        IF((IALE-IABEG)/NFACT*NFACT.NE.(IALE-IABEG)) THEN
           CALL UTEROR(+2,REAL(IALE),REAL(NFACT))
        ENDIF
C
        DO 30 IBODY=IABEG,IAEND
        READ(IUNIT,20) (BODYG2(K,IBODY),K=1,2)
20      FORMAT(2F10.0)
30      CONTINUE
C
        CALL G2BOND(IABEG,IAEND,1)
C
C     STORE THE AIRFOIL LEADING AND TRAILING EDGE LOCATIONS
C
        XALE=BODYG2(1,IALE)
        YALE=BODYG2(2,IALE)
C
        XATE=BODYG2(1,IABEG)
        YATE=BODYG2(2,IABEG)
C
C     COMPUTE THE XI AT THE UPSTREAM AND DOWNSTREAM BOUNDARIES
C        SO THAT DELTA-XI IS CONTINUOUS AT THE LEADING AND
C        TRAILING EDGES
C
        DXIAF=(XATE-XALE)/(NAIR+1)
C
        OMGUP=OMG(CXUP,1,(NXUP-1))
        OMGDN=OMG(CXDN,1,(NXDN-1))
        XIUP=XALE-DXIAF/OMGUP
        XIDN=XATE+DXIAF/OMGDN
C
C*****GENERATE THE FAR-FIELD BOUNDARIES AND SPLINE FIT THEM
C
C     ...SOUTHERN BOUNDARY
C
```

```
      DO 40 IPNT=1,NXBODY
      IBODY=ISBEG-1+IPNT
      INODE=(IPNT-1)/NFACT+1
C
      OMGAI=REAL(IPNT-1)/REAL(NXBODY-1)
      BODYG2(1,IBODY)=XDN  *OMGAI+XUP  *(1.0-OMGAI)
      BODYG2(2,IBODY)=YLODN*OMGAI+YLOUP*(1.0-OMGAI)
C
      IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
         GEOMG2(1,INODE)=BODYG2(1,IBODY)
         GEOMG2(2,INODE)=BODYG2(2,IBODY)
         DPENG2(1,INODE)=INODE
         DPENG2(2,INODE)=BODYG2(2,IBODY)
         DPENG2(3,INODE)=0.0
      ENDIF
C
40    CONTINUE
C
      CALL G2BOND(ISBEG,ISEND,1)
C
C     ...NORTHERN BOUNDARY
C
      DO 50 IPNT=1,NXBODY
      IBODY=INEND+1-IPNT
      INODE=(IPNT-1)/NFACT+1+NXNODE*(NYNODE-1)
C
      OMGAI=REAL(IPNT-1)/REAL(NXBODY-1)
      BODYG2(1,IBODY)=XDN  *OMGAI+XUP  *(1.0-OMGAI)
      BODYG2(2,IBODY)=YUPDN*OMGAI+YUPUP*(1.0-OMGAI)
C
      IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
         GEOMG2(1,INODE)=BODYG2(1,IBODY)
         GEOMG2(2,INODE)=BODYG2(2,IBODY)
         DPENG2(1,INODE)=INODE-NXNODE*(NYNODE-1)
         DPENG2(2,INODE)=BODYG2(2,IBODY)
         DPENG2(3,INODE)=0.0
      ENDIF
C
50    CONTINUE
C
      CALL G2BOND(INBEG,INEND,1)
C
C     ...WESTERN BOUNDARY
C
      DO 60 IPNT=1,NYBODY
      IBODY=IWEND+1-IPNT
      INODE=((IPNT-1)/NFACT+1)*NXNODE+1-NXNODE
C
      OMGAJ=REAL(IPNT-1)/REAL(NYBODY-1)
      BODYG2(1,IBODY)=XUP
      BODYG2(2,IBODY)=YUPUP*OMGAJ+YLOUP*(1.0-OMGAJ)
C
      IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
```

```
            GEOMG2(1,INODE)=BODYG2(1,IBODY)
            GEOMG2(2,INODE)=BODYG2(2,IBODY)
            DPENG2(1,INODE)=1
            DPENG2(2,INODE)=BODYG2(2,IBODY)
            DPENG2(3,INODE)=0.0
          ENDIF
C
          CALL G2BOND(IWBEG,IWEND,1)
C
60        CONTINUE
C
C       ...EASTERN BOUNDARY
C
          DO 70 IPNT=1,NYBODY
          IBODY=IEBEG-1+IPNT
          INODE=((IPNT-1)/NFACT+1)*NXNODE
C
          OMGAJ=REAL(IPNT-1)/REAL(NYBODY-1)
          BODYG2(1,IBODY)=XDN
          BODYG2(2,IBODY)=YUPDN*OMGAJ+YLODN*(1.0-OMGAJ)
C
          IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
            GEOMG2(1,INODE)=BODYG2(1,IBODY)
            GEOMG2(2,INODE)=BODYG2(2,IBODY)
            DPENG2(1,INODE)=NXNODE
            DPENG2(2,INODE)=BODYG2(2,IBODY)
            DPENG2(3,INODE)=0.0
          ENDIF
C
70        CONTINUE
C
          CALL G2BOND(IEBEG,IEEND,1)
C
          NBODG2=IWEND
C
C       SET UP THE PSEUDO-STAGNATION STREAMLINES
C
C       ...UPSTREAM OF AIRFOIL
C
          ISTAG=(NYLO-1)*NXNODE
          INODLE=ISTAG+NXUP
          DO 80 INODE=1,NXUP
          OMGAX=OMG(BXUP,NXUP-INODE,NXUP-1)
          OMGAI=OMG(CXUP,NXUP-INODE,NXUP-1)
C
          GEOMG2(1,ISTAG+INODE)=XUP *OMGAX+XALE*(1.0-OMGAX)
          GEOMG2(2,ISTAG+INODE)=0.0 *OMGAX+YALE*(1.0-OMGAX)
          DPENG2(1,ISTAG+INODE)=INODE
          DPENG2(2,ISTAG+INODE)=0.0 *OMGAI+YALE*(1.0-OMGAI)
          DPENG2(3,ISTAG+INODE)=0.0
80        CONTINUE
C
C       ...DOWNSTREAM OF AIRFOIL
```

```
C
        ISTAG=(NYLO-1)*NXNODE+NXUP+NAIR
        INODTE=ISTAG+1
        DO 90 INODE=1,NXDN
        OMGAX=OMG(BXDN,INODE-1,NXDN-1)
        OMGAI=OMG(CXDN,INODE-1,NXDN-1)
C
        GEOMG2(1,ISTAG+INODE)=XDN *OMGAX+XATE*(1.0-OMGAX)
        GEOMG2(2,ISTAG+INODE)=0.0 *OMGAX+YATE*(1.0-OMGAX)
        DPENG2(1,ISTAG+INODE)=INODE+NXUP+NAIR
        DPENG2(2,ISTAG+INODE)=0.0 *OMGAI+YATE*(1.0-OMGAI)
        DPENG2(3,ISTAG+INODE)=0.0
90      CONTINUE
C
        DPENG2(3,INODTE)=0.0
C
C     SET UP THE AIRFOIL SURFACE
C
C     ...LOWER SURFACE
C
        DO 100 I=1,NAIR
        INODE=I+NXUP+(NYLO-1)*NXNODE
        IBODY=(NAIR-I+1)*NFACT+1
        FRAC=(BODYG2(1,IBODY)-XALE)/(XATE-XALE)
C
        GEOMG2(1,INODE)=BODYG2(1,IBODY)
        GEOMG2(2,INODE)=BODYG2(2,IBODY)
        DPENG2(1,INODE)=I+NXUP
        DPENG2(2,INODE)=YALE*(1.0-FRAC)+YATE*FRAC
        DPENG2(3,INODE)=0.0
100     CONTINUE
C
C     ...UPPER SURFACE
C
        DO 110 I=1,NAIR
        INODE=I+NYNODE*NXNODE
        IBODY=(NAIR+I+1)*NFACT+1
        FRAC=(BODYG2(1,IBODY)-XALE)/(XATE-XALE)
C
        GEOMG2(1,INODE)=BODYG2(1,IBODY)
        GEOMG2(2,INODE)=BODYG2(2,IBODY)
        DPENG2(1,INODE)=I+NXUP
        DPENG2(2,INODE)=YALE*(1.0-FRAC)+YATE*FRAC
        DPENG2(3,INODE)=0.0
110     CONTINUE
C
C*****SET UP ALL INTERIOR NODES USING AN ALGEBRAIC TECHNIQUE
C
C     ABOVE THE AIRFOIL
C
        DO 120 J=2,NYUP-1
C
        OMGL=REAL(J-1)/REAL(NYUP-1)
```

```
      OMGC=OMG(CYUP,J-1,NYUP-1)
      OMGR=REAL(J-1)/REAL(NYUP-1)
C
      DO 115 I=2,NXNODE-1
C
      IBOT =NXNODE*(NYLO     -1)+I
      ITOP =NXNODE*(NYLO+NYUP-2)+I
      INODE=NXNODE*(NYLO+J   -2)+I
      IF(I.GT.NXUP .AND. I.LT.(NXNODE-NXDN)) IBOT=NXNODE*NYNODE+I-NXUP
C
      IF(I.LT.NXUP) THEN
         FRAC=REAL(I-1)/REAL(NXUP-1)
         OMGAY=OMGC*FRAC+OMGL*(1.0-FRAC)
         OMGAX=OMGAY**2
      ELSEIF(I.GT.NXUP+NAIR+1) THEN
         FRAC=REAL(NXNODE-I)/REAL(NXDN-1)
         OMGAY=OMGC*FRAC+OMGR*(1.0-FRAC)
         OMGAX=OMGAY**2
      ELSE
         OMGAY=OMGC
         OMGAX=OMGAY**2
      ENDIF
C
      GEOMG2(1,INODE)=OMGAX*GEOMG2(1,ITOP)+(1.0-OMGAX)*GEOMG2(1,IBOT)
      GEOMG2(2,INODE)=OMGAY*GEOMG2(2,ITOP)+(1.0-OMGAY)*GEOMG2(2,IBOT)
      DPENG2(1,INODE)=I
      DPENG2(2,INODE)=OMGAY*DPENG2(2,ITOP)+(1.0-OMGAY)*DPENG2(2,IBOT)
      DPENG2(3,INODE)=1.00
115   CONTINUE
C
120   CONTINUE
C
C     BELOW THE AIRFOIL
C
      DO 130 J=2,NYLO-1
C
      OMGL=REAL(NYLO-J)/REAL(NYLO-1)
      OMGC=OMG(CYLO,NYLO-J,NYLO-1)
      OMGR=REAL(NYLO-J)/REAL(NYLO-1)
C
      DO 125 I=2,NXNODE-1
C
      IBOT =                 I
      ITOP =NXNODE*(NYLO-1)+I
      INODE=NXNODE*(J   -1)+I
C
      IF(I.LT.NXUP) THEN
         FRAC=REAL(I-1)/REAL(NXUP-1)
         OMGAY=OMGC*FRAC+OMGL*(1.0-FRAC)
         OMGAX=OMGAY**2
      ELSEIF(I.GT.NXUP+NAIR+1) THEN
         FRAC=REAL(NXNODE-I)/REAL(NXDN-1)
         OMGAY=OMGC*FRAC+OMGR*(1.0-FRAC)
```

```
            OMGAX=OMGAY**2
         ELSE
            OMGAY=OMGC
            OMGAX=OMGAY**2
         ENDIF
C
         GEOMG2(1,INODE)=OMGAX*GEOMG2(1,IBOT)+(1.0-OMGAX)*GEOMG2(1,ITOP)
         GEOMG2(2,INODE)=OMGAY*GEOMG2(2,IBOT)+(1.0-OMGAY)*GEOMG2(2,ITOP)
         DPENG2(1,INODE)=I
         DPENG2(2,INODE)=OMGAY*DPENG2(2,IBOT)+(1.0-OMGAY)*DPENG2(2,ITOP)
         DPENG2(3,INODE)=1.00
125      CONTINUE
C
130      CONTINUE
C
C*****SET UP POINTER SYSTEM (DATA STRUCTURE) FOR CELLS ON ALL LEVELS
C
C     INITIALIZE THE NUMBER OF CELLS AND BOUNDARY CONDITION POINTERS
C
         NCELG2=0
         NBNDG2=0
C
C     COMPUTE INDICES OF FIRST AND LAST POINTS WHICH ARE ON THE LOWER
C        AIRFOIL SURFACE SO THAT THEY CAN BE TRANSFERRED TO POINT TO
C        UPPER SURFACE NODES (IF ALONG THE SOUTH SIDE OF A CELL)
C
         IBEG=NXUP+   1+(NYLO-1)*NXNODE
         IEND=NXUP+NAIR+(NYLO-1)*NXNODE
         IOFF=NYUP*NXNODE-NXUP
C
C     INITIALIZE POINTERS FOR ALL LEVELS
C
         DO 140 ILEVEL=-MLVLG2,MLVLG2
         ILVLG2(1,ILEVEL)=1
         ILVLG2(2,ILEVEL)=0
         ILVLG2(3,ILEVEL)=1
         ILVLG2(4,ILEVEL)=0
         ILVLG2(5,ILEVEL)=1
         ILVLG2(6,ILEVEL)=0
140      CONTINUE
C
C     LOOP THROUGH ALL COARSER GRID LEVELS (IF ANY)
C
         DO 160 ICOARS=MIN(NCOARS,MLVLG2-1),1,-1
         ISIZE=2**ICOARS
C
C     LOOP THROUGH EACH CELL ON THIS LEVEL
C
         DO 150 JCELL=1,NYCELL,ISIZE
         DO 150 ICELL=1,NXCELL,ISIZE
C
         NCELG2=NCELG2+1
C
```

```
C     FIND THE CENTER OF THIS CELL
C
        ICELG2(1,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE/2-1)*NXNODE
C
C     COMPUTE INDICES OF ALL BOUNDING NODES
C
        ICELG2(2,NCELG2)=(ICELL        )+(JCELL        -1)*NXNODE
        ICELG2(3,NCELG2)=(ICELL+ISIZE/2)+(JCELL        -1)*NXNODE
        ICELG2(4,NCELG2)=(ICELL+ISIZE  )+(JCELL        -1)*NXNODE
        ICELG2(5,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE/2-1)*NXNODE
        ICELG2(6,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(7,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(8,NCELG2)=(ICELL        )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(9,NCELG2)=(ICELL        )+(JCELL+ISIZE/2-1)*NXNODE
C
C     ADJUST SOUTH NODES TO POINT TO UPPER SURFACE OF AIRFOIL INSTEAD
C
        IF(ICELG2(2,NCELG2).GE.IBEG .AND. ICELG2(2,NCELG2).LE.IEND)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFF
        IF(ICELG2(3,NCELG2).GE.IBEG .AND. ICELG2(3,NCELG2).LE.IEND)
     &                          ICELG2(3,NCELG2)=ICELG2(3,NCELG2)+IOFF
        IF(ICELG2(4,NCELG2).GE.IBEG .AND. ICELG2(4,NCELG2).LE.IEND)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFF
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C      THE NEIGHBOR INFORMATION IS NOT USED ON COARSE GLOBAL LEVELS,
C        SO SET THEM ALL TO ZERO
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=0
        NBORG2(3,NCELG2)=0
        NBORG2(4,NCELG2)=0
        NBORG2(5,NCELG2)=0
        NBORG2(6,NCELG2)=0
        NBORG2(7,NCELG2)=0
        NBORG2(8,NCELG2)=0
        NBORG2(9,NCELG2)=0
C
150     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THIS LEVEL
C
        ILEVEL=-ICOARS
        ILVLG2(1,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(2,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(3,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(4,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(5,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(6,ILEVEL)=NCELG2
C
```

```
C     GO BACK FOR A FINER COARSE LEVEL
C
160     CONTINUE
C
C     LOOP THROUGH EACH GLOBAL CELL
C
        IOFFN=NCELG2-NXCELL
C
        DO 170 JCELL=1,NYCELL
        DO 170 ICELL=1,NXCELL
C
        NCELG2=NCELG2+1
C
C     COMPUTE INDICES OF EACH CORNER OF CELL
C
        ICELG2(2,NCELG2)=ICELL  +(JCELL-1)*NXNODE
        ICELG2(4,NCELG2)=ICELL+1+(JCELL-1)*NXNODE
        ICELG2(6,NCELG2)=ICELL+1+(JCELL  )*NXNODE
        ICELG2(8,NCELG2)=ICELL  +(JCELL  )*NXNODE
C
C     ADJUST SOUTH NODES TO POINT TO UPPER SURFACE OF AIRFOIL INSTEAD
C
        IF(ICELG2(2,NCELG2).GE.IBEG .AND. ICELG2(2,NCELG2).LE.IEND)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFF
        IF(ICELG2(4,NCELG2).GE.IBEG .AND. ICELG2(4,NCELG2).LE.IEND)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFF
C
C     INITIALLY, THERE IS NO NODE IN THE CENTER OF A FINE CELL
C
        ICELG2(1,NCELG2)=0
C
C     THERE ARE NO NODES IN THE CENTER OF THE SIDES OF A FINE CELL
C
        ICELG2(3,NCELG2)=0
        ICELG2(5,NCELG2)=0
        ICELG2(7,NCELG2)=0
        ICELG2(9,NCELG2)=0
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C     INITIALIZE ALL NEIGHBOR INFORMATION FOR THIS LEVEL
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL-1)
        NBORG2(3,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL  )
        NBORG2(4,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL+1)
        NBORG2(5,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL+1)
        NBORG2(6,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL+1)
        NBORG2(7,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL  )
        NBORG2(8,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL-1)
        NBORG2(9,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL-1)
```

```
C
C     CORRECT NEIGHBOR INFORMATION FOR BOUNDARY CELLS
C
        IF(ICELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(8,NCELG2)=0
           NBORG2(9,NCELG2)=0
        ENDIF
C
        IF(ICELL.EQ.NXCELL) THEN
           NBORG2(4,NCELG2)=0
           NBORG2(5,NCELG2)=0
           NBORG2(6,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(3,NCELG2)=0
           NBORG2(4,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYCELL) THEN
           NBORG2(6,NCELG2)=0
           NBORG2(7,NCELG2)=0
           NBORG2(8,NCELG2)=0
        ENDIF
C
C     CORRECT NEIGHBOR INFORMATION FOR CELLS AROUND AIRFOIL
C
        IF(JCELL.EQ.NYLO-1 .AND. ICELL.EQ.NXUP) THEN
           NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
           NBORG2(6,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYLO-1 .AND. ICELL.GT.NXUP
     &                     .AND. ICELL.LT.NXUP+NAIR) THEN
           NBORG2(6,NCELG2)=0
           NBORG2(7,NCELG2)=0
           NBORG2(8,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYLO-1 .AND. ICELL.EQ.NXUP+NAIR) THEN
           NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
           NBORG2(8,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYLO   .AND. ICELL.EQ.NXUP) THEN
           NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
           NBORG2(4,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYLO   .AND. ICELL.GT.NXUP
     &                     .AND. ICELL.LT.NXUP+NAIR) THEN
```

```
            NBORG2(2,NCELG2)=0
            NBORG2(3,NCELG2)=0
            NBORG2(4,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO    .AND. ICELL.EQ.NXUP+NAIR) THEN
            NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
            NBORG2(2,NCELG2)=0
         ENDIF
C
170      CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THE GLOBAL FINE LEVEL
C
         ILVLG2(1,0)=ILVLG2(6,-1)+1
         ILVLG2(2,0)=NCELG2
         ILVLG2(3,0)=NCELG2+1
         ILVLG2(4,0)=NCELG2
         ILVLG2(5,0)=NCELG2+1
         ILVLG2(6,0)=NCELG2
C
C     INITIALIZE THE MULTIPLE-GRID-LEVEL ARRAY FOR ALL EMBEDDED MESHES
C
         DO 180 ILEVEL=1,MLVLG2+1
         ILVLG2(1,ILEVEL)=NCELG2+1
         ILVLG2(2,ILEVEL)=NCELG2
         ILVLG2(3,ILEVEL)=NCELG2+1
         ILVLG2(4,ILEVEL)=NCELG2
         ILVLG2(5,ILEVEL)=NCELG2+1
         ILVLG2(6,ILEVEL)=NCELG2
180      CONTINUE
C
C*****ADD BOUNDARY CONDITION INFORMATION TO POINTER SYSTEM
C
C     SET UP THE BOUNDARY CONDITION ARRAYS AND SET BOUNDARY POINTERS FOR
C        BOUNDARY NODES AT THE FAR-FIELD
C        ...FOR SOUTHWESTERN CORNER
C
         NBNDG2=NBNDG2+1
C
         IBNDG2(1,NBNDG2)=1
         IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+1
         IBNDG2(3,NBNDG2)=0
         IBNDG2(4,NBNDG2)=11
         IBNDG2(5,NBNDG2)=4
         IBNDG2(6,NBNDG2)=0
C
         BONDG2(1,NBNDG2)=BODYG2( 8,IWEND)
         BONDG2(2,NBNDG2)=BODYG2( 9,IWEND)
         BONDG2(3,NBNDG2)=BODYG2(10,ISBEG)
         BONDG2(4,NBNDG2)=BODYG2(11,ISBEG)
C
         ICELG2(10,IBNDG2(2,NBNDG2))
```

```
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000B)
C
C        ...FOR SOUTHERN EDGE
C
         DO 190 IBOUND=2,NXNODE-1
         NBNDG2=NBNDG2+1
C
         IBNDG2(1,NBNDG2)=IBOUND
         IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+IBOUND-1
         IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+IBOUND
         IBNDG2(4,NBNDG2)=3
         IBNDG2(5,NBNDG2)=4
         IBNDG2(6,NBNDG2)=0
C
         IBODY=(IBOUND-1)*NFACT+ISBEG
         BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
         BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
         BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
         BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
         ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
         ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
190      CONTINUE
C
C        ...FOR SOUTHEASTERN CORNER
C
         NBNDG2=NBNDG2+1
C
         IBNDG2(1,NBNDG2)=NXNODE
         IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL
         IBNDG2(3,NBNDG2)=0
         IBNDG2(4,NBNDG2)=7
         IBNDG2(5,NBNDG2)=4
         IBNDG2(6,NBNDG2)=0
C
         BONDG2(1,NBNDG2)=BODYG2( 8,ISEND)
         BONDG2(2,NBNDG2)=BODYG2( 9,ISEND)
         BONDG2(3,NBNDG2)=BODYG2(10,IEBEG)
         BONDG2(4,NBNDG2)=BODYG2(11,IEBEG)
C
         ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0007)
C
C        ...FOR EASTERN EDGE
C
         DO 200 IBOUND=2,NYNODE-1
         NBNDG2=NBNDG2+1
C
         IBNDG2(1,NBNDG2)=NXNODE* IBOUND
         IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)
```

```
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL* IBOUND
      IBNDG2(4,NBNDG2)=6
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(IBOUND-1)*NFACT+IEBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0006)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0006)
C
200   CONTINUE
C
C       ...FOR NORTHEASTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*NYNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*NYCELL
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=14
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG3)=BODYG2( 8,IEEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,IEEND)
      BONDG2(3,NBNDG2)=BODYG2(10,INBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,INBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000E)
C
C       ...FOR NORTHERN EDGE
C
      DO 210 IBOUND=NXNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+IBOUND
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(NXNODE-IBOUND)*NFACT+INBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
```

```
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
210   CONTINUE
C
C      ...FOR NORTHWESTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+1
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=13
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,INEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,INEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IWBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IWBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000D)
C
C      ...FOR WESTERN EDGE
C
      DO 220 IBOUND=NYNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(IBOUND-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)+1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-2)+1
      IBNDG2(4,NBNDG2)=9
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(NYNODE-IBOUND)*NFACT+IWBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0009)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0009)
C
220   CONTINUE
C
```

```
C         ...FOR LEADING EDGE
C
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=NXUP+(NYLO-1)*NXNODE
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-2)*NXCELL+NXUP
          IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-1)*NXCELL+NXUP
          IBNDG2(4,NBNDG2)=5
          IBNDG2(5,NBNDG2)=3
          IBNDG2(6,NBNDG2)=0
C
          BONDG2(1,NBNDG2)=BODYG2( 8,IALE)
          BONDG2(2,NBNDG2)=BODYG2( 9,IALE)
          BONDG2(3,NBNDG2)=BODYG2(10,IALE)
          BONDG2(4,NBNDG2)=BODYG2(11,IALE)
C
          ICELG2(10,IBNDG2(2,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)-1),HL0004)
          ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL000C)
          ICELG2(10,IBNDG2(3,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)-1),HL0002)
          ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL0003)
C
C         ...FOR TRAILING EDGE
C
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=NXUP+NAIR+1+(NYLO-1)*NXNODE
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-1)*NXCELL+NXUP+NAIR
          IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-2)*NXCELL+NXUP+NAIR
          IBNDG2(4,NBNDG2)=10
          IBNDG2(5,NBNDG2)=1
          IBNDG2(6,NBNDG2)=0
C
          BONDG2(1,NBNDG2)=BODYG2( 8,IAEND)
          BONDG2(2,NBNDG2)=BODYG2( 9,IAEND)
          BONDG2(3,NBNDG2)=BODYG2(10,IABEG)
          BONDG2(4,NBNDG2)=BODYG2(11,IABEG)
C
          ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL000C)
          ICELG2(10,IBNDG2(3,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)+1),HL0008)
          ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL0003)
          ICELG2(10,IBNDG2(2,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)+1),HL0001)
C
C         ...LOWER SURFACE OF AIRFOIL
C
          DO 230 IBOUND=1,NAIR
```

```
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXUP+(NYLO-1)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-2)*NXCELL
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-2)*NXCELL-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IALE-IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
230   CONTINUE
C
C      ...UPPER SURFACE OF AIRFOIL
C
      DO 240 IBOUND=1,NAIR
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXNODE*NYNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-1)*NXCELL-1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-1)*NXCELL
      IBNDG2(4,NBNDG2)=3
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IALE+IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBCDY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
240   CONTINUE
C
C*****USE LAPLACE'S EQUATION FOR SMOOTH GRID VARIATION
C
      IF(NITER.GE.0) THEN
        CALL G2LAPL(NITER,IOUT,1,IER)
C
C     UN-COMMENT OUT THE NEXT FOUR LINES TO LOOK AT THE XI-ETA
```

```
C        DISTRIBUTION USED IN THE GRID GENERATION
C
         ELSE
            DO 245 INODE=1,NNODG2
            GEOMG2(1,INODE)=DPENG2(1,INODE)
            GEOMG2(2,INODE)=DPENG2(2,INODE)
245         CONTINUE
         ENDIF
C
C*****INITIALIZE THE DEPENDENT VARIABLES AT ALL NODES AND BOUNDARY
C        POINTERS
C
         DO 250 INODE=1,NNODG2
         DPENG2(1,INODE)=DPEN1
         DPENG2(2,INODE)=DPEN2
         DPENG2(3,INODE)=DPEN3
         DPENG2(4,INODE)=DPEN4
         DPENG2(5,INODE)=DPEN5
250      CONTINUE
C
         DO 260 IBODY=1,NBODG2
         BODYG2(3,IBODY)=DPEN1
         BODYG2(4,IBODY)=DPEN2
         BODYG2(5,IBODY)=DPEN3
         BODYG2(6,IBODY)=DPEN4
         BODYG2(7,IBODY)=DPEN5
260      CONTINUE
C
         DO 270 IBOUND=1,NBNDG2
         BONDG2(5,IBOUND)=DPEN1
         BONDG2(6,IBOUND)=DPEN2
         BONDG2(7,IBOUND)=DPEN3
         BONDG2(8,IBOUND)=DPEN4
         BONDG2(9,IBOUND)=DPEN5
270      CONTINUE
C
C     PRE-EMBED NEAR AIRFOIL SURFACE IF DESIRED
C
         DO 280 ISURF=1,NSURF
         CALL G2SURF(1,3,NCELL)
280      CONTINUE
C
C     GROW THE EMBEDDED REGION IF DESIRED
C
         IF(NGROW1.NE.0) CALL G2GROW(NGROW1,NGROW2,NCELL)
C
C*****SORT THE CELLS
C
         CALL G2SORT
C
         RETURN
         END
```

G2INP5

```
      SUBROUTINE G2INP5(IUNIT,IOUT
     &
     &                              )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 5 GEOMETRY DEFINITION, WHICH
C       IS A TWO-PART AIRFOIL. THE AIRFOIL IS INPUT CLOCKWISE,
C       STARTING AT THE TRAILING EDGE.  NEXT, THE FLAP IS DEFINED
C       IN A SIMILAR MANNER.  THE NUMBER OF POINTS ON THE UPPER AND
C       LOWER SURFACES OF BOTH PARTS ARE ASSUMED TO BE THE SAME
C
C*************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C     STATEMENT FUNCTION FOR STRETCHING FUNCTION
C
      OMG(B,I,IMAX)=(EXP(B*REAL(I)/REAL(IMAX))-1.0)
     &              /(EXP(         B          )-1.0)
C
C*************************************************************************
C
C     READ GLOBAL MESH SIZE AND MESH INDICATOR
C
      READ(IUNIT,10) NAIRFL,KDEFG2,NITER ,NCOARS,NSURF ,NGROW1,NGROW2,
     &               NFLAP ,NXOFF ,NYOFF ,
     &               DPEN1 ,DPEN2 ,DPEN3 ,DPEN4 ,DPEN5 ,
     &               XUP   ,BXUP  ,NXUP  ,
     &               XDN   ,BXDN  ,NXDN  ,
     &               YLO   ,BYLO  ,NYLO  ,
     &               YUP   ,BYUP  ,NYUP
10    FORMAT(7I5/
     &       3I5/
     &       5F10.0/
     &       2F10.0,I5/
     &       2F10.0,I5/
     &       2F10.0,I5/
     &       2F10.0,I5)
C
C     COMPUTE TOTAL NUMBER OF POINTS INPUT ON THE AIRFOIL AND FLAP
C
      NFACT =2**KDEFG2
      NABODY=1+(NAIRFL-1)*NFACT
      NFBODY=1+(NFLAP -1)*NFACT
C
C     FIND THE NUMBER OF POINTS ON THE UPPER AND LOWER AIRFOIL AND FLAP
```

```
C        SURFACES (EXCLUDING THE LEADING AND TRAILING EDGES)
C
         NAIR=(NABODY-1)/(2*NFACT)-1
         NFLP=(NFBODY-1)/(2*NFACT)-1
C
C     COMPUTE NUMBER OF NODES IN EACH DIRECTION
C
         NXNODE=NXUP+NAIR+NXDN
         NYNODE=NYLO+NYUP-1
C
C     COMPUTE NUMBER OF BODY POINTS IN THE FAR FIELD IN EACH DIRECTION
C
         NXBODY=1+(NXNODE-1)*NFACT
         NYBODY=1+(NYNODE-1)*NFACT
C
C     COMPUTE WHICH BODY POINTS COINCIDE WITH THE AIRFOIL AND FLAP
C
         JAIR=(NYLO      -1)*NFACT+1
         JFLP=(NYLO-NYOFF-1)*NFACT+1
C
C     COMPUTE TOTAL NUMBER OF GLOBAL NODES
C
         NNODG2=NXNODE*NYNODE+NAIR+NFLP
C
C     COMPUTE NUMBER OF CELLS IN EACH DIRECTION ON THE GLOBAL MESH
C
         NXCELL=NXNODE-1
         NYCELL=NYNODE-1
C
C     INDICES FOR SURFACE FITTING
C
         IABEG=     +1
         IAEND=     +NABODY
         IFBEG=IAEND+1
         IFEND=IAEND+NFBODY
         ISBEG=IFEND+1
         ISEND=IFEND+NXBODY
         IEBEG=ISEND+1
         IEEND=ISEND+NYBODY
         INBEG=IEEND+1
         INEND=IEEND+NXBODY
         IWBEG=INEND+1
         IWEND=INEND+NYBODY
C
         IF(IWEND.GT.MBODG2) THEN
            CALL UTEROR(+1,REAL(IWEND),REAL(MBODG2))
         ENDIF
C
C     SET UP OFFSET DISTANCES FOR INDEXING UPPER SURFACES OF
C        THE AIRFOIL AND FLAP AS WELL AS THE FIRST AND LAST INDICES
C        WHICH MUST BE OFFSET
C
         IOFFA=(NYUP        )*NXNODE-NXUP
```

```
        IBEGA=(NYLO      -1)*NXNODE+NXUP+1
        IENDA=(NYLO      -1)*NXNODE+NXUP+NAIR
C
        IOFFF=(NYUP+NYOFF  )*NXNODE-NXUP-NXOFF+NAIR
        IBEGF=(NYLO-NYOFF-1)*NXNODE+NXUP+NXOFF+1
        IENDF=(NYLO-NYOFF-1)*NXNODE+NXUP+NXOFF+NFLP
C
C*****READ THE AIRFOIL AND FLAP SHAPES AND SPLINE FIT THEM
C
C     ...AIRFOIL
C
        IALE=(IABEG+IAEND)/2
C
        IF((IALE-IABEG)/NFACT*NFACT.NE.(IALE-IABEG)) THEN
           CALL UTERROR(+2,REAL(IALE),REAL(NFACT))
        ENDIF
C
        DO 30 IBODY=IABEG,IAEND
        READ(IUNIT,20) (BODYG2(K,IBODY),K=1,2)
20      FORMAT(2F10.0)
30      CONTINUE
C
        CALL G2BOND(IABEG,IAEND,1)
C
C     STORE THE AIRFOIL LEADING AND TRAILING EDGE LOCATIONS
C
        XALE=BODYG2(1,IALE)
        YALE=BODYG2(2,IALE)
C
        XATE=BODYG2(1,IABEG)
        YATE=BODYG2(2,IABEG)
C
C     ...FLAP
C
        IFLE=(IFBEG+IFEND)/2
C
        IF((IFLE-IFBEG)/NFACT*NFACT.NE.(IFLE-IFBEG)) THEN
           CALL UTERROR(+3,REAL(IFLE),REAL(NFACT))
        ENDIF
C
        DO 35 IBODY=IFBEG,IFEND
        READ(IUNIT,20) (BODYG2(K,IBODY),K=1,2)
35      CONTINUE
C
        CALL G2BOND(IFBEG,IFEND,1)
C
C     STORE THE FLAP LEADING AND TRAILING EDGE LOCATIONS
C
        XFLE=BODYG2(1,IFLE)
        YFLE=BODYG2(2,IFLE)
C
        XFTE=BODYG2(1,IFBEG)
        YFTE=BODYG2(2,IFBEG)
```

```
C
C*****GENERATE THE FAR-FIELD BOUNDARIES AND SPLINE FIT THEM
C
C      ...SOUTHERN BOUNDARY
C
        DO 40 IPNT=1,NXBODY
        IBODY=ISBEG-1+IPNT
        INODE=(IPNT-1)/NFACT+1
C
        OMEGAI=REAL(IPNT-1)/REAL(NXBODY-1)
        BODYG2(1,IBODY)=XDN*OMEGAI+XUP*(1.0-OMEGAI)
        BODYG2(2,IBODY)=YLO
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=(IPNT-1)/NFACT+1
           DPENG2(2,INODE)=YLO
           DPENG2(3,INODE)=0.0
        ENDIF
C
40      CONTINUE
C
        CALL G2BOND(ISBEG,ISEND,1)
C
C      ...NORTHERN BOUNDARY
C
        DO 50 IPNT=1,NXBODY
        IBODY=INEND+1-IPNT
        INODE=(IPNT-1)/NFACT+1+NXNODE*(NYNODE-1)
C
        OMEGAI=REAL(IPNT-1)/REAL(NXBODY-1)
        BODYG2(1,IBODY)=XDN*OMEGAI+XUP*(1.0-OMEGAI)
        BODYG2(2,IBODY)=YUP
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=(IPNT-1)/NFACT+1
           DPENG2(2,INODE)=YUP
           DPENG2(3,INODE)=0.0
        ENDIF
C
50      CONTINUE
C
        CALL G2BOND(INBEG,INEND,1)
C
C      ...WESTERN BOUNDARY
C
        DO 60 IPNT=1,NYBODY
        IBODY=IWEND+1-IPNT
        INODE=((IPNT-1)/NFACT)*NXNODE+1
C
```

```
        OMEGAJ=REAL(IPNT-1)/REAL(NYBODY-1)
        BODYG2(1,IBODY)=XUP
        BODYG2(2,IBODY)=YUP*OMEGAJ+YLO*(1.0-OMEGAJ)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=1.0
           DPENG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(3,INODE)=0.0
        ENDIF
C
60      CONTINUE
C
        CALL G2BOND(IWBEG,IWEND,1)
C
C     ...EASTERN BOUNDARY
C
        DO 70 IPNT=1,NYBODY
        IBODY=IEBEG-1+IPNT
        INODE=((IPNT-1)/NFACT+1)*NXNODE
C
        OMEGAJ=REAL(IPNT-1)/REAL(NYBODY-1)
        BODYG2(1,IBODY)=XDN
        BODYG2(2,IBODY)=YUP*OMEGAJ+YLO*(1.0-OMEGAJ)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=NXNODE
           DPENG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(3,INODE)=0.0
        ENDIF
C
70      CONTINUE
C
        CALL G2BOND(IEBEG,IEEND,1)
C
        NBODG2=IWEND
C
C     SET UP ETA FOR THE AIRFOIL AND FLAP
C
        ETAIR=0.0
        ETFLP=YLO*OMG(BYLO,NYOFF,NYLO-1)
C
C     SET UP THE AIRFOIL AND FLAP SURFACES
C
C     ...LOWER SURFACE OF AIRFOIL
C
        DO 100 I=1,NAIR
        INODE=I+NXUP+(NYLO-1)*NXNODE
        IBODY=(NAIR-I+1)*NFACT+IABEG
C
```

```
        GEOMG2(1,INODE)=BODYG2(1,IBODY)
        GEOMG2(2,INODE)=BODYG2(2,IBODY)
        DPENG2(1,INODE)=I+NXUP
        DPENG2(2,INODE)=ETAIR
        DPENG2(3,INODE)= 0.0
100     CONTINUE
C
C      ...LOWER SURFACE OF FLAP
C
        DO 105 I=1,NFLP
        INODE=I+NXUP+NXOFF+(NYLO-NYOFF-1)*NXNODE
        IBODY=(NFLP-I+1)*NFACT+IFBEG
C
        GEOMG2(1,INODE)=BODYG2(1,IBODY)
        GEOMG2(2,INODE)=BODYG2(2,IBODY)
        DPENG2(1,INODE)=I+NXUP+NXOFF
        DPENG2(2,INODE)=ETFLP
        DPENG2(3,INODE)=0.0
105     CONTINUE
C
C      ...UPPER SURFACE OF AIRFOIL
C
        DO 110 I=1,NAIR
        INODE=I+NYNODE*NXNODE
        IBODY=(NAIR+I+1)*NFACT+IABEG
C
        GEOMG2(1,INODE)=BODYG2(1,IBODY)
        GEOMG2(2,INODE)=BODYG2(2,IBODY)
        DPENG2(1,INODE)=I+NXUP
        DPENG2(2,INODE)=ETAIR
        DPENG2(3,INODE)= 0.0
110     CONTINUE
C
C      ...UPPER SURFACE OF FLAP
C
        DO 115 I=1,NFLP
        INODE=I+NAIR+NYNODE*NXNODE
        IBODY=(NFLP+I+1)*NFACT+IFBEG
C
        GEOMG2(1,INODE)=BODYG2(1,IBODY)
        GEOMG2(2,INODE)=BODYG2(2,IBODY)
        DPENG2(1,INODE)=I+NXUP+NXOFF
        DPENG2(2,INODE)=ETFLP
        DPENG2(3,INODE)= 0.0
115     CONTINUE
C
C      ...AIRFOIL LEADING EDGE
C
        INODE=(NYLO-1)*NXNODE+NXUP
        GEOMG2(1,INODE)=XALE
        GEOMG2(2,INODE)=YALE
        DPENG2(1,INODE)=NXUP
        DPENG2(2,INODE)=ETAIR
```

```
        DPENG2(3,INODE)=0.0
C
C     ...AIRFOIL TRAILING EDGE
C
        INODE=(NYLO-1)*NXNODE+NXUP+NAIR+1
        GEOMG2(1,INODE)=XATE
        GEOMG2(2,INODE)=YATE
        DPENG2(1,INODE)=NXUP+NAIR+1
        DPENG2(2,INODE)=ETAIR
        DPENG2(3,INODE)=0.0
C
C     ...FLAP LEADING EDGE
C
        INODE=(NYLO-NYOFF-1)*NXNODE+NXUP+NXOFF
        GEOMG2(1,INODE)=XFLE
        GEOMG2(2,INODE)=YFLE
        DPENG2(1,INODE)=NXUP+NXOFF
        DPENG2(2,INODE)=ETFLP
        DPENG2(3,INODE)=0.0
C
C     ...FLAP TRAILING EDGE
C
        INODE=(NYLO-NYOFF-1)*NXNODE+NXUP+NXOFF+NFLP+1
        GEOMG2(1,INODE)=XFTE
        GEOMG2(2,INODE)=YFTE
        DPENG2(1,INODE)=NXUP+NXOFF+NFLP+1
        DPENG2(2,INODE)=ETFLP
        DPENG2(3,INODE)=0.0
C
C    SET UP THE PSEUDO-STAGNATION STREAMLINES
C
        NXTE=NXUP+NXOFF+NFLP+1
C
C     ...UPSTREAM OF AIRFOIL
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        IOFF=(NYLO-1)*NXNODE
        DO 80 INODE=2,NXUP-1
        OMEGAI=OMG(BXUP,NXUP-INODE,NXUP-1)
C
        GEOMG2(1,IOFF+INODE)=GEOMG2(1,IOFF+1)*OMEGAI+ XALE*(1.0-OMEGAI)
        GEOMG2(2,IOFF+INODE)=GEOMG2(2,IOFF+1)*OMEGAI+ YALE*(1.0-OMEGAI)
        DPENG2(1,IOFF+INODE)=INODE
        DPENG2(2,IOFF+INODE)=DPENG2(2,IOFF+1)*OMEGAI+ETAIR*(1.0-OMEGAI)
        DPENG2(3,IOFF+INODE)= 0.0
80      CONTINUE
C
C     ...DOWNSTREAM OF FLAP
C
        IOFF=(NYLO-NYOFF-1)*NXNODE
        DO 95 INODE=NXUP+NXOFF+NFLP+2,NXNODE-1
        OMEGAI=OMG(BXDN,INODE-NXUP-NXOFF-NFLP-1,
     &                  NXNODE-NXUP-NXOFF-NFLP-1)
```

```
C
      GEOMG2(1,IOFF+INODE)=GEOMG2(1,IOFF+NXNODE)*OMEGAI
     &                                          +XFTE*(1.0-OMEGAI)
      GEOMG2(2,IOFF+INODE)=GEOMG2(2,IOFF+NXNODE)*OMEGAI
     &                                          +YFTE*(1.0-OMEGAI)
      DPENG2(1,IOFF+INODE)=INODE
      DPENG2(2,IOFF+INODE)=DPENG2(2,IOFF+NXNODE)*OMEGAI
     &                                          +ETFLP*(1.0-OMEGAI)
      DPENG2(3,IOFF+INODE)= 0.0
95    CONTINUE
C
C     ...UPSTREAM OF FLAP
C
      IOFF=(NYLO-NYOFF-1)*NXNODE
      JOFF=(NYLO      -1)*NXNODE
      DO 90 INODE=2,NXUP+NXOFF-1
      OMEGAI=(GEOMG2(1,JOFF+INODE     )-GEOMG2(1,JOFF+1))
     &       /(GEOMG2(1,JOFF+NXUP+NXOFF)-GEOMG2(1,JOFF+1))
      OMEGAK=(REAL(NXUP-INODE)/REAL(NXUP-1))**2
C
      GEOMG2(1,IOFF+INODE)=XFLE*OMEGAI+GEOMG2(1,IOFF+1)*(1.0-OMEGAI)
      GEOMG2(2,IOFF+INODE)=YFLE*OMEGAI+GEOMG2(2,IOFF+1)*(1.0-OMEGAI)
      DPENG2(1,IOFF+INODE)=INODE
      IF(INODE.GE.NXUP) THEN
         DPENG2(2,IOFF+INODE)=ETFLP
      ELSE
         DPENG2(2,IOFF+INODE)=DPENG2(2,IOFF+1)*OMEGAK
     &                                          +ETFLP*(1.0-OMEGAK)
      ENDIF
      DPENG2(3,IOFF+INODE)=1.0
90    CONTINUE
C
C     ...DOWNSTREAM OF AIRFOIL
C
      IOFF=(NYLO      -1)*NXNODE
      JOFF=(NYLO-NYOFF-1)*NXNODE
      DO 85 INODE=NXUP+NAIR+2,NXNODE-1
      OMEGAI=(GEOMG2(1,JOFF+INODE )-GEOMG2(1,JOFF+NXUP+NAIR+1))
     &       /(GEOMG2(1,JOFF+NXNODE)-GEOMG2(1,JOFF+NXUP+NAIR+1))
      OMEGAK=(REAL(INODE-NXTE)/REAL(NXNODE-NXTE))**2
C
      GEOMG2(1,IOFF+INODE)=GEOMG2(1,IOFF+NXNODE)*OMEGAI
     &                                          +XATE*(1.0-OMEGAI)
      GEOMG2(2,IOFF+INODE)=GEOMG2(2,IOFF+NXNODE)*OMEGAI
     &                                          +YATE*(1.0-OMEGAI)
      DPENG2(1,IOFF+INODE)=INODE
      IF(INODE.LE.NXTE) THEN
         DPENG2(2,IOFF+INODE)=ETAIR
      ELSE
         DPENG2(2,IOFF+INODE)=DPENG2(2,IOFF+NXNODE)*OMEGAK
     &                                          +ETAIR*(1.0-OMEGAK)
      ENDIF
      DPENG2(3,IOFF+INODE)=1.0
```

```
85      CONTINUE
C
C*****SET UP ALL INTERIOR NODES USING AN ALGEBRAIC TECHNIQUE
C
C     ...BELOW FLAP
C
        DO 120 INODE=2,NXNODE-1
        IBOT=INODE
        ITOP=INODE+(NYLO-NYOFF-1)*NXNODE
C
        DO 120 JNODE=2,NYLO-NYOFF-1
        KNODE=(JNODE-1)*NXNODE+INODE
C
        OMEGAJ=OMG(BYLO,NYLO-NYOFF-JNODE,NYLO-NYOFF-1)
        OMEGAK=REAL(NYLO-NYOFF-JNODE)/REAL(NYLO-NYOFF-1)
C
        IF(INODE.LT.NXUP) THEN
           OMEGAL=(REAL(NXUP-INODE)/REAL(NXUP-1))**2
        ELSEIF(INODE.GT.NXTE) THEN
           OMEGAL=(REAL(INODE-NXTE)/REAL(NXNODE-NXTE))**2
        ELSE
           OMEGAL=0.0
        ENDIF
C
        OMEGA=OMEGAK*OMEGAL+OMEGAJ*(1.0-OMEGAL)
C
        GEOMG2(1,KNODE)=GEOMG2(1,IBOT)*OMEGA+GEOMG2(1,ITOP)*(1.-OMEGA)
        GEOMG2(2,KNODE)=GEOMG2(2,IBOT)*OMEGA+GEOMG2(2,ITOP)*(1.-OMEGA)
        DPENG2(1,KNODE)=INODE
        DPENG2(2,KNODE)=DPENG2(2,IBOT)*OMEGA+DPENG2(2,ITOP)*(1.-OMEGA)
        DPENG2(3,KNODE)= 1.0
120     CONTINUE
C
C     ...ABOVE AIRFOIL
C
        DO 130 INODE=2,NXNODE-1
        IBOT=INODE+(NYLO  -1)*NXNODE
        ITOP=INODE+(NYNODE-1)*NXNODE
C
        IF(IBOT.GE.IBEGA .AND. IBOT.LE.IENDA) IBOT=IBOT+IOFFA
C
        DO 130 JNODE=NYLO+1,NYNODE-1
        KNODE=(JNODE-1)*NXNODE+INODE
C
        OMEGAJ=OMG(BYUP,JNODE-NYLO,NYUP-1)
        OMEGAK=REAL(JNODE-NYLO)/REAL(NYUP-1)
C
        IF(INODE.LT.NXUP) THEN
           OMEGAL=(REAL(NXUP-INODE)/REAL(NXUP-1))**2
        ELSEIF(INODE.GT.NXTE) THEN
           OMEGAL=(REAL(INODE-NXTE)/REAL(NXNODE-NXTE))**2
        ELSE
           OMEGAL=0.0
```

```
        ENDIF
C
        OMEGA=OMEGAK*OMEGAL+OMEGAJ*(1.0-OMEGAL)
C
        GEOMG2(1,KNODE)=GEOMG2(1,ITOP)*OMEGA+GEOMG2(1,IBOT)*(1.-OMEGA)
        GEOMG2(2,KNODE)=GEOMG2(2,ITOP)*OMEGA+GEOMG2(2,IBOT)*(1.-OMEGA)
        DPENG2(1,KNODE)=INODE
        DPENG2(2,KNODE)=DPENG2(2,ITOP)*OMEGA+DPENG2(2,IBOT)*(1.-OMEGA)
        DPENG2(3,KNODE)= 1.0
130     CONTINUE
C
C     ...BETWEEN FLAP AND AIRFOIL
C
        DO 135 INODE=2,NXNODE-1
        IBOT=INODE+(NYLO-NYOFF-1)*NXNODE
        ITOP=INODE+(NYLO      -1)*NXNODE
C
        IF(IBOT.GE.IBEGF .AND. IBOT.LE.IENDF) IBOT=IBOT+IOFFF
C
        DO 135 JNODE=NYLO-NYOFF+1,NYLO-1
        KNODE=(JNODE-1)*NXNODE+INODE
C
        OMEGAJ=REAL(JNODE-NYLO+NYOFF)/REAL(NYOFF)
C
        GEOMG2(1,KNODE)=GEOMG2(1,ITOP)*OMEGAJ+GEOMG2(1,IBOT)*(1.-OMEGAJ)
        GEOMG2(2,KNODE)=GEOMG2(2,ITOP)*OMEGAJ+GEOMG2(2,IBOT)*(1.-OMEGAJ)
        DPENG2(1,KNODE)=INODE
        DPENG2(2,KNODE)=DPENG2(2,ITOP)*OMEGAJ+DPENG2(2,IBOT)*(1.-OMEGAJ)
        DPENG2(3,KNODE)= 1.0
135     CONTINUE
CCCCCCCCCCCCCCCCCCCCCCCCC
C
C*****SET UP POINTER SYSTEM (DATA STRUCTURE) FOR CELLS ON ALL LEVELS
C
C     INITIALIZE THE NUMBER OF CELLS AND BOUNDARY CONDITION POINTERS
C
        NCELG2=0
        NBNDG2=0
C
C     INITIALIZE POINTERS FOR ALL LEVELS
C
        DO 140 ILEVEL=-MLVLG2,MLVLG2
        ILVLG2(1,ILEVEL)=1
        ILVLG2(2,ILEVEL)=0
        ILVLG2(3,ILEVEL)=1
        ILVLG2(4,ILEVEL)=0
        ILVLG2(5,ILEVEL)=1
        ILVLG2(6,ILEVEL)=0
140     CONTINUE
C
C     LOOP THROUGH ALL COARSER GRID LEVELS (IF ANY)
C
        DO 160 ICOARS=MIN(NCOARS,MLVLG2-1),1,-1
```

```
        ISIZE=2**ICOARS
C
C     LOOP THROUGH EACH CELL ON THIS LEVEL
C
        DO 150 JCELL=1,NYCELL,ISIZE
        DO 150 ICELL=1,NXCELL,ISIZE
C
        NCELG2=NCELG2+1
C
C     FIND THE CENTER OF THIS CELL
C
        ICELG2(1,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE/2-1)*NXNODE
C
C     COMPUTE INDICES OF ALL BOUNDING NODES
C
        ICELG2(2,NCELG2)=(ICELL        )+(JCELL        -1)*NXNODE
        ICELG2(3,NCELG2)=(ICELL+ISIZE/2)+(JCELL        -1)*NXNODE
        ICELG2(4,NCELG2)=(ICELL+ISIZE  )+(JCELL        -1)*NXNODE
        ICELG2(5,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE/2-1)*NXNODE
        ICELG2(6,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(7,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(8,NCELG2)=(ICELL        )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(9,NCELG2)=(ICELL        )+(JCELL+ISIZE/2-1)*NXNODE
C
C     ADJUST SOUTH NODES TO POINT TO UPPER SURFACE OF AIRFOIL OR FLAP
C       INSTEAD
C
        IF(ICELG2(2,NCELG2).GE.IBEGA .AND. ICELG2(2,NCELG2).LE.IENDA)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFA
        IF(ICELG2(3,NCELG2).GE.IBEGA .AND. ICELG2(3,NCELG2).LE.IENDA)
     &                          ICELG2(3,NCELG2)=ICELG2(3,NCELG2)+IOFFA
        IF(ICELG2(4,NCELG2).GE.IBEGA .AND. ICELG2(4,NCELG2).LE.IENDA)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFA
C
        IF(ICELG2(2,NCELG2).GE.IBEGF .AND. ICELG2(2,NCELG2).LE.IENDF)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFF
        IF(ICELG2(3,NCELG2).GE.IBEGF .AND. ICELG2(3,NCELG2).LE.IENDF)
     &                          ICELG2(3,NCELG2)=ICELG2(3,NCELG2)+IOFFF
        IF(ICELG2(4,NCELG2).GE.IBEGF .AND. ICELG2(4,NCELG2).LE.IENDF)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFF
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C     THE NEIGHBOR INFORMATION IS NOT USED ON COARSE GLOBAL LEVELS,
C       SO SET THEM ALL TO ZERO
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=0
        NBORG2(3,NCELG2)=0
        NBORG2(4,NCELG2)=0
        NBORG2(5,NCELG2)=0
```

```
        NBORG2(6,NCELG2)=0
        NBORG2(7,NCELG2)=0
        NBORG2(8,NCELG2)=0
        NBORG2(9,NCELG2)=0
C
150     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THIS LEVEL
C
        ILEVEL=-ICOARS
        ILVLG2(1,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(2,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(3,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(4,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(5,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(6,ILEVEL)=NCELG2
C
C     GO BACK FOR A FINER COARSE LEVEL
C
160     CONTINUE
C
C     LOOP THROUGH EACH GLOBAL CELL
C
        IOFFN=NCELG2-NXCELL
C
        DO 170 JCELL=1,NYCELL
        DO 170 ICELL=1,NXCELL
C
        NCELG2=NCELG2+1
C
C     COMPUTE INDICES OF EACH CORNER OF CELL
C
        ICELG2(2,NCELG2)=ICELL  +(JCELL-1)*NXNODE
        ICELG2(4,NCELG2)=ICELL+1+(JCELL-1)*NXNODE
        ICELG2(6,NCELG2)=ICELL+1+(JCELL  )*NXNODE
        ICELG2(8,NCELG2)=ICELL  +(JCELL  )*NXNODE
C
C     ADJUST SOUTH NODES TO POINT TO UPPER SURFACE OF AIRFOIL OR FLAP
C       INSTEAD
C
        IF(ICELG2(2,NCELG2).GE.IBEGA .AND. ICELG2(2,NCELG2).LE.IENDA)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFA
        IF(ICELG2(4,NCELG2).GE.IBEGA .AND. ICELG2(4,NCELG2).LE.IENDA)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFA
C
        IF(ICELG2(2,NCELG2).GE.IBEGF .AND. ICELG2(2,NCELG2).LE.IENDF)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFF
        IF(ICELG2(4,NCELG2).GE.IBEGF .AND. ICELG2(4,NCELG2).LE.IENDF)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFF
C
C     INITIALLY, THERE IS NO NODE IN THE CENTER OF A FINE CELL
C
        ICELG2(1,NCELG2)=0
```

```
C
C     THERE ARE NO NODES IN THE CENTER OF THE SIDES OF A FINE CELL
C
        ICELG2(3,NCELG2)=0
        ICELG2(5,NCELG2)=0
        ICELG2(7,NCELG2)=0
        ICELG2(9,NCELG2)=0
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C     INITIALIZE ALL NEIGHBOR INFORMATION FOR THIS LEVEL
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL-1)
        NBORG2(3,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL  )
        NBORG2(4,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL+1)
        NBORG2(5,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL+1)
        NBORG2(6,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL+1)
        NBORG2(7,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL  )
        NBORG2(8,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL-1)
        NBORG2(9,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL-1)
C
C     CORRECT NEIGHBOR INFORMATION FOR BOUNDARY CELLS
C
        IF(ICELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(8,NCELG2)=0
           NBORG2(9,NCELG2)=0
        ENDIF
C
        IF(ICELL.EQ.NXCELL) THEN
           NBORG2(4,NCELG2)=0
           NBORG2(5,NCELG2)=0
           NBORG2(6,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(3,NCELG2)=0
           NBORG2(4,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYCELL) THEN
           NBORG2(6,NCELG2)=0
           NBORG2(7,NCELG2)=0
           NBORG2(8,NCELG2)=0
        ENDIF
C
C     CORRECT NEIGHBOR INFORMATION FOR CELLS AROUND AIRFOIL
C
        IF(JCELL.EQ.NYLO-1 .AND. ICELL.EQ.NXUP) THEN
```

```
            NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
            NBORG2(6,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO-1 .AND. ICELL.GT.NXUP
     &                      .AND. ICELL.LT.NXUP+NAIR) THEN
            NBORG2(6,NCELG2)=0
            NBORG2(7,NCELG2)=0
            NBORG2(8,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO-1 .AND. ICELL.EQ.NXUP+NAIR) THEN
            NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
            NBORG2(8,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO   .AND. ICELL.EQ.NXUP) THEN
            NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
            NBORG2(4,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO   .AND. ICELL.GT.NXUP
     &                      .AND. ICELL.LT.NXUP+NAIR) THEN
            NBORG2(2,NCELG2)=0
            NBORG2(3,NCELG2)=0
            NBORG2(4,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO   .AND. ICELL.EQ.NXUP+NAIR) THEN
            NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
            NBORG2(2,NCELG2)=0
         ENDIF
C
C     CORRECT NEIGHBOR INFORMATION FOR CELLS AROUND FLAP
C
         IF(JCELL.EQ.NYLO-NYOFF-1 .AND. ICELL.EQ.NXUP+NXOFF) THEN
            NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
            NBORG2(6,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO-NYOFF-1 .AND. ICELL.GT.NXUP+NXOFF
     &                      .AND. ICELL.LT.NXUP+NXOFF+NFLP) THEN
            NBORG2(6,NCELG2)=0
            NBORG2(7,NCELG2)=0
            NBORG2(8,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO-NYOFF-1 .AND. ICELL.EQ.NXUP+NXOFF+NFLP) THEN
            NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
            NBORG2(8,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO-NYOFF   .AND. ICELL.EQ.NXUP+NXOFF) THEN
```

```
            NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
            NBORG2(4,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO-NYOFF    .AND. ICELL.GT.NXUP+NXOFF
     &                         .AND. ICELL.LT.NXUP+NXOFF+NFLP) THEN
            NBORG2(2,NCELG2)=0
            NBORG2(3,NCELG2)=0
            NBORG2(4,NCELG2)=0
         ENDIF
C
         IF(JCELL.EQ.NYLO-NYOFF    .AND. JCELL.EQ.NXUP+NXOFF+NFLP) THEN
            NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
            NBORG2(2,NCELG2)=0
         ENDIF
C
170      CONTINUE
C
C      SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THE GLOBAL FINE LEVEL
C
         ILVLG2(1,0)=ILVLG2(6,-1)+1
         ILVLG2(2,0)=NCELG2
         ILVLG2(3,0)=NCELG2+1
         ILVLG2(4,0)=NCELG2
         ILVLG2(5,0)=NCELG2+1
         ILVLG2(6,0)=NCELG2
C
C      INITIALIZE THE MULTIPLE-GRID-LEVEL ARRAY FOR ALL EMBEDDED MESHES
C
         DO 180 ILEVEL=1,MLVLG2+1
         ILVLG2(1,ILEVEL)=NCELG2+1
         ILVLG2(2,ILEVEL)=NCELG2
         ILVLG2(3,ILEVEL)=NCELG2+1
         ILVLG2(4,ILEVEL)=NCELG2
         ILVLG2(5,ILEVEL)=NCELG2+1
         ILVLG2(6,ILEVEL)=NCELG2
180      CONTINUE
C
C*****ADD BOUNDARY CONDITION INFORMATION TO POINTER SYSTEM
C
C      SET UP THE BOUNDARY CONDITION ARRAYS AND SET BOUNDARY POINTERS FOR
C         BOUNDARY NODES AT THE FAR-FIELD
C         ...FOR SOUTHWESTERN CORNER
C
         NBNDG2=NBNDG2+1
C
         IBNDG2(1,NBNDG2)=1
         IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+1
         IBNDG2(3,NBNDG2)=0
         IBNDG2(4,NBNDG2)=11
         IBNDG2(5,NBNDG2)=4
         IBNDG2(6,NBNDG2)=0
C
```

```
      BONDG2(1,NBNDG2)=BODYG2( 8,IWEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,IWEND)
      BONDG2(3,NBN  2)=BODYG2(10,ISBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,ISBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000B)
C
C       ...FOR SOUTHERN EDGE
C
      DO 190 IBOUND=2,NXNODE-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+IBOUND-1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+IBOUND
      IBNDG2(4,NBNDG2)=3
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(IBOUND-1)*NFACT+ISBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
190   CONTINUE
C
C       ...FOR SOUTHEASTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=7
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,ISEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,ISEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IEBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IEBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0007)
C
C       ...FOR EASTERN EDGE
```

```
C
      DO 200 IBOUND=2,NYNODE-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE* IBOUND
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL* IBOUND
      IBNDG2(4,NBNDG2)=6
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(IBOUND-1)*NFACT+IEBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0006)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0006)
C
200   CONTINUE
C
C       ...FOR NORTHEASTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*NYNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*NYCELL
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=14
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,IEEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,IEEND)
      BONDG2(3,NBNDG2)=BODYG2(10,INBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,INBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000E)
C
C       ...FOR NORTHERN EDGE
C
      DO 210 IBOUND=NXNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+IBOUND
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=4
```

```
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(NXNODE-IBOUND)*NFACT+INBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
210   CONTINUE
C
C      ...FOR NORTHWESTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+1
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=13
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,INEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,INEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IWBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IWBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000D)
C
C      ...FOR WESTERN EDGE
C
      DO 220 IBOUND=NYNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(IBOUND-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)+1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-2)+1
      IBNDG2(4,NBNDG2)=9
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(NYNODE-IBOUND)*NFACT+IWBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
```

```
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0009)
        ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0009)
C
220     CONTINUE
C
C        ...FOR LEADING EDGE OF AIRFOIL
C
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=NXUP+(NYLO-1)*NXNODE
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-2)*NXCELL+NXUP
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-1)*NXCELL+NXUP
        IBNDG2(4,NBNDG2)=5
        IBNDG2(5,NBNDG2)=3
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IALE)
        BONDG2(2,NBNDG2)=BODYG2( 9,IALE)
        BONDG2(3,NBNDG2)=BODYG2(10,IALE)
        BONDG2(4,NBNDG2)=BODYG2(11,IALE)
C
        ICELG2(10,IBNDG2(2,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)-1),HL0004)
        ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL000C)
        ICELG2(10,IBNDG2(3,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)-1),HL0002)
        ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL0003)
C
C        ...FOR TRAILING EDGE OF AIRFOIL
C
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=NXUP+NAIR+1+(NYLO-1)*NXNODE
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-1)*NXCELL+NXUP+NAIR
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-2)*NXCELL+NXUP+NAIR
        IBNDG2(4,NBNDG2)=10
        IBNDG2(5,NBNDG2)=1
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IAEND)
        BONDG2(2,NBNDG2)=BODYG2( 9,IAEND)
        BONDG2(3,NBNDG2)=BODYG2(10,IABEG)
        BONDG2(4,NBNDG2)=BODYG2(11,IABEG)
C
        ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL000C)
        ICELG2(10,IBNDG2(3,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)+1),HL0008)
        ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL0003)
```

```
      ICELG2(10,IBNDG2(2,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)+1),HL0001)
C
C       ...LOWER SURFACE OF AIRFOIL OF AIRFOIL
C
      DO 230 IBOUND=1,NAIR
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXUP+(NYLO-1)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-2)*NXCELL
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-2)*NXCELL-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IALE-IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
230   CONTINUE
C
C       ...UPPER SURFACE OF AIRFOIL
C
      DO 240 IBOUND=1,NAIR
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXNODE*NYNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-1)*NXCELL-1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-1)*NXCELL
      IBNDG2(4,NBNDG2)=3
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IALE+IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
240   CONTINUE
C
```

```
C         ...FOR LEADING EDGE OF FLAP
C
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=NXUP+NXOFF+(NYLO-NYOFF-1)*NXNODE
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-NYOFF-2)*NXCELL+NXUP+NXOFF
          IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-NYOFF-1)*NXCELL+NXUP+NXOFF
          IBNDG2(4,NBNDG2)=5
          IBNDG2(5,NBNDG2)=3
          IBNDG2(6,NBNDG2)=0
C
          BONDG2(1,NBNDG2)=BODYG2( 8,IFLE)
          BONDG2(2,NBNDG2)=BODYG2( 9,IFLE)
          BONDG2(3,NBNDG2)=BODYG2(10,IFLE)
          BONDG2(4,NBNDG2)=BODYG2(11,IFLE)
C
          ICELG2(10,IBNDG2(2,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)-1),HL0004)
          ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL000C)
          ICELG2(10,IBNDG2(3,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)-1),HL0002)
          ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL0003)
C
C         ...FOR TRAILING EDGE OF FLAP
C
          NBNDG2=NBNDG2+1
C
          IBNDG2(1,NBNDG2)=NXUP+NXOFF+NFLP+1+(NYLO-NYOFF-1)*NXNODE
          IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-NYOFF-1)*NXCELL
     &                                        +NXUP+NFLP+NXOFF
          IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-NYOFF-2)*NXCELL
     &                                        +NXUP+NFLP+NXOFF
          IBNDG2(4,NBNDG2)=10
          IBNDG2(5,NBNDG2)=1
          IBNDG2(6,NBNDG2)=0
C
          BONDG2(1,NBNDG2)=BODYG2( 8,IFEND)
          BONDG2(2,NBNDG2)=BODYG2( 9,IFEND)
          BONDG2(3,NBNDG2)=BODYG2(10,IFBEG)
          BONDG2(4,NBNDG2)=BODYG2(11,IFBEG)
C
          ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL000C)
          ICELG2(10,IBNDG2(3,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)+1),HL0008)
          ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL0003)
          ICELG2(10,IBNDG2(2,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)+1),HL0001)
C
C         ...LOWER SURFACE OF AIRFOIL OF FLAP
```

```
C
      DO 235 IBOUND=1,NFLP
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXUP+NXOFF+(NYLO-NYOFF-1)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+NXOFF+IBOUND
     &                                +(NYLO-NYOFF-2)*NXCELL
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+NXOFF+IBOUND
     &                                +(NYLO-NYOFF-2)*NXCELL-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IFLE-IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
235   CONTINUE
C
C      ...UPPER SURFACE OF FLAP
C
      DO 245 IBOUND=1,NFLP
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NAIR+NXNODE*NYNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+NXOFF+IBOUND
     &                                +(NYLO-NYOFF-1)*NXCELL-1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+NXOFF+IBOUND
     &                                +(NYLO-NYOFF-1)*NXCELL
      IBNDG2(4,NBNDG2)=3
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IFLE+IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
245   CONTINUE
C
```

```
C*****USE LAPLACE'S EQUATION FOR SMOOTH GRID VARIATION
C
        CALL G2LAPL(NITER,IOUT,1,IER)
C
C*****INITIALIZE THE DEPENDENT VARIABLES AT ALL NODES AND BOUNDARY
C        POINTERS
C
        DO 250 INODE=1,NNODG2
        DPENG2(1,INODE)=DPEN1
        DPENG2(2,INODE)=DPEN2
        DPENG2(3,INODE)=DPEN3
        DPENG2(4,INODE)=DPEN4
        DPENG2(5,INODE)=DPEN5
250     CONTINUE
C
        DO 260 IBODY=1,NBODG2
        BODYG2(3,IBODY)=DPEN1
        BODYG2(4,IBODY)=DPEN2
        BODYG2(5,IBODY)=DPEN3
        BODYG2(6,IBODY)=DPEN4
        BODYG2(7,IBODY)=DPEN5
260     CONTINUE
C
        DO 270 IBOUND=1,NBNDG2
        BONDG2(5,IBOUND)=DPEN1
        BONDG2(6,IBOUND)=DPEN2
        BONDG2(7,IBOUND)=DPEN3
        BONDG2(8,IBOUND)=DPEN4
        BONDG2(9,IBOUND)=DPEN5
270     CONTINUE
C
C     PRE-EMBED NEAR AIRFOIL AND FLAP SURFACES IF DESIRED
C
        DO 280 ISURF=1,NSURF
        CALL G2SURF(1,3,NCELL)
280     CONTINUE
C
C     GROW THE EMBEDDED REGION IF DESIRED
C
        IF(NGROW1.NE.0) CALL G2GROW(NGROW1,NGROW2,NCELL)
C
C*****SORT THE CELLS
C
        CALL G2SORT
C
        RETURN
        END
```

G2INP6

```
      SUBROUTINE G2INP6(IUNIT,IOUT
     &
     &                              )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 6 GEOMETRY DEFINITION, WHICH
C       IS A H-TYPE MESH AROUND A BI-PLANE.  THE AIRFOILS ARE THE SAME
C       AND ARE INPUT CLOCKWISE, STARTING AT THE TRAILING EDGE.  THE
C       NUMBER OF POINTS ON THE UPPER AND LOWER AIRFOIL SURFACES ARE
C       ASSUMED TO BE THE SAME
C
C*****************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C     STATEMENT FUNCTION FOR STRETCHING FUNCTION
C
      OMG(B,I,IMAX)=(EXP(B*REAL(I)/REAL(IMAX))-1.0)
     &              /(EXP(         B          )-1.0)
C
C*****************************************************************************
C
C     READ GLOBAL MESH SIZE AND MESH INDICATOR
C
      READ(IUNIT,10) NAIRFL,KDEFG2,NITER ,NCOARS,NSURF ,NGROW1,NGROW2,
     &               DPEN1 ,DPEN2 ,DPEN3 ,DPEN4 ,DPEN5 ,
     &               YGAP  ,NGAP  ,
     &               XUP   ,BXUP  ,NXUP  ,
     &               XDN   ,BXDN  ,NXDN  ,
     &               YLOUP ,YLODN ,BYLO  ,NYLO  ,
     &               YUPUP ,YUPDN ,BYUP  ,NYUP
10    FORMAT(7I5/
     &       5F10.0/
     &        F10.0,I5/
     &       2F10.0,I5/
     &       2F10.0,I5/
     &       3F10.0,I5/
     &       3F10.0,I5)
C
C     COMPUTE TOTAL NUMBER OF POINTS INPUT ON THE AIRFOIL
C
      NFACT =2**KDEFG2
      NABODY=1+(NAIRFL-1)*NFACT
C
C     FIND THE NUMBER OF POINTS ON THE UPPER AND LOWER AIRFOIL SURFACE
C       (EXCLUDING THE LEADING AND TRAILING EDGES)
```

```
C
        NAIR=(NABODY-1)/(2*NFACT)-1
C
C     COMPUTE NUMBER OF NODES IN EACH DIRECTION
C
        NXNODE=NXUP+NAIR+NXDN
        NYNODE=NYLO+NYUP+NGAP-2
C
C     COMPUTE NUMNER OF BODY POINTS IN EACH DIRECTION
C
        NXBODY=1+(NXNODE-1)*NFACT
        NYBODY=1+(NYNODE-1)*NFACT
C
C     COMPUTE TOTAL NUMBER OF GLOBAL NODES
C
        NNODG2=NXNODE*(NYLO+NYUP+NGAP)-2*(NXUP+NXDN)
C
C     COMPUTE NUMBER OF CELLS IN EACH DIRECTION ON THE GLOBAL MESH
C
        NXCELL=NXNODE-1
        NYCELL=NYNODE-1
C
C     INDICES FOR SURFACE FITTING
C
        IABEG=     +1
        IAEND=     +NABODY
        IBBEG=IAEND+1
        IBEND=IAEND+NABODY
        ISBEG=IBEND+1
        ISEND=IBEND+NXBODY
        IEBEG=ISEND+1
        IEEND=ISEND+NYBODY
        INBEG=IEEND+1
        INEND=IEEND+NXBODY
        IWBEG=INEND+1
        IWEND=INEND+NYBODY
C
        IF(IWEND.GT.MBODG2) THEN
           CALL UTEROR(+1,REAL(IWEND),REAL(MBODG2))
        ENDIF
C
C*****READ THE AIRFOIL SHAPE AND SPLINE FIT IT
C
        IALE=(IABEG+IAEND)/2
        IBLE=IALE+NABODY
C
        IF((IALE-IABEG)/NFACT*NFACT.NE.(IALE-IABEG)) THEN
           CALL UTEROR(+2,REAL(IALE),REAL(NFACT))
        ENDIF
C
        DO 30 IBODY=1,NABODY
        READ(IUNIT,20) X,Y
        BODYG2(1,IBODY+IABEG-1)=X
```

```
        BODYG2(2,IBODY+IABEG-1)=Y-0.5*YGAP
        BODYG2(1,IBODY+IBBEG-1)=X
        BODYG2(2,IBODY+IBBEG-1)=Y+0.5*YGAP
20      FORMAT(2F10.0)
30      CONTINUE
C
        CALL G2BOND(IABEG,IAEND,1)
        CALL G2BOND(IBBEG,IBEND,1)
C
C     STORE THE AIRFOIL LEADING AND TRAILING EDGE LOCATIONS
C
        XALE=BODYG2(1,IALE)
        YALE=BODYG2(2,IALE)
C
        XATE=BODYG2(1,IABEG)
        YATE=BODYG2(2,IABEG)
C
        XBLE=BODYG2(1,IBLE)
        YBLE=BODYG2(2,IBLE)
C
        XBTE=BODYG2(1,IBBEG)
        YBTE=BODYG2(2,IBBEG)
C
C*****GENERATE THE FAR-FIELD BOUNDARIES AND SPLINE FIT THEM
C
C     ...SOUTHERN BOUNDARY
C
        DO 40 IPNT=1,NXBODY
        IBODY=ISBEG-1+IPNT
        INODE=(IPNT-1)/NFACT+1
C
        OMGAI=REAL(IPNT-1)/REAL(NXBODY-1)
        BODYG2(1,IBODY)=XDN  *OMGAI+XUP  *(1.0-OMGAI)
        BODYG2(2,IBODY)=YLODN*OMGAI+YLOUP*(1.0-OMGAI)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=BODYG2(1,IBODY)
           DPENG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(3,INODE)=0.0
        ENDIF
C
40      CONTINUE
C
        CALL G2BOND(ISBEG,ISEND,1)
C
C     ...NORTHERN BOUNDARY
C
        DO 50 IPNT=1,NXBODY
        IBODY=INEND+1-IPNT
        INODE=(IPNT-1)/NFACT+1+NXNODE*(NYNODE-1)
C
```

```
        OMGAI=REAL(IPNT-1)/REAL(NXBODY-1)
        BODYG2(1,IBODY)=XDN  *OMGAI+XUP  *(1.0-OMGAI)
        BODYG2(2,IBODY)=YUPDN*OMGAI+YUPUP*(1.0-OMGAI)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=BODYG2(1,IBODY)
           DPENG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(3,INODE)=0.0
        ENDIF
C
50      CONTINUE
C
        CALL G2BOND(INBEG,INEND,1)
C
C     ...WESTERN BOUNDARY
C
        DO 60 IPNT=1,NYBODY
        IBODY=IWEND+1-IPNT
        INODE=((IPNT-1)/NFACT+1)*NXNODE+1-NXNODE
C
        OMGAJ=REAL(IPNT-1)/REAL(NYBODY-1)
        BODYG2(1,IBODY)=XUP
        BODYG2(2,IBODY)=YUPUP*OMGAJ+YLOUP*(1.0-OMGAJ)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=BODYG2(1,IBODY)
           DPENG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(3,INODE)=0.0
        ENDIF
C
        CALL G2BOND(IWBEG,IWEND,1)
C
60      CONTINUE
C
C     ...EASTERN BOUNDARY
C
        DO 70 IPNT=1,NYBODY
        IBODY=IEBEG-1+IPNT
        INODE=((IPNT-1)/NFACT+1)*NXNODE
C
        OMGAJ=REAL(IPNT-1)/REAL(NYBODY-1)
        BODYG2(1,IBODY)=XDN
        BODYG2(2,IBODY)=YUPDN*OMGAJ+YLODN*(1.0-OMGAJ)
C
        IF(((IPNT-1)/NFACT*NFACT).EQ.(IPNT-1)) THEN
           GEOMG2(1,INODE)=BODYG2(1,IBODY)
           GEOMG2(2,INODE)=BODYG2(2,IBODY)
           DPENG2(1,INODE)=BODYG2(1,IBODY)
           DPENG2(2,INODE)=BODYG2(2,IBODY)
```

```
            DPENG2(3,INODE)=0.0
          ENDIF
C
70        CONTINUE
C
          CALL G2BOND(IEBEG,IEEND,1)
C
          NBODG2=IWEND
C
C     SET UP THE PSEUDO-STAGNATION STREAMLINES
C
C     ...UPSTREAM OF AIRFOIL
C
          ISTAGA=(NYLO     -1)*NXNODE
          ISTAGB=(NYLO+NGAP-2)*NXNODE
          DO 80 INODE=2,NXUP
          OMGAI=OMG(BXUP,NXUP-INODE,NXUP-1)
C
          GEOMG2(1,ISTAGA+INODE)=XUP*OMGAI             +XALE*(1.0-OMGAI)
          GEOMG2(2,ISTAGA+INODE)=GEOMG2(2,ISTAGA+1)*OMGAI
     &                                                  +YALE*(1.0-OMGAI)
          DPENG2(1,ISTAGA+INODE)=XUP*OMGAI             +XALE*(1.0-OMGAI)
          DPENG2(2,ISTAGA+INODE)=GEOMG2(2,ISTAGA+1)*OMGAI
     &                                                  +YALE*(1.0-OMGAI)
          DPENG2(3,ISTAGA+INODE)=1.0
C
          GEOMG2(1,ISTAGB+INODE)=XUP*OMGAI             +XBLE*(1.0-OMGAI)
          GEOMG2(2,ISTAGB+INODE)=GEOMG2(2,ISTAGB+1)*OMGAI
     &                                                  +YBLE*(1.0-OMGAI)
          DPENG2(1,ISTAGB+INODE)=XUP*OMGAI             +XBLE*(1.0-OMGAI)
          DPENG2(2,ISTAGB+INODE)=GEOMG2(2,ISTAGB+1)*OMGAI
     &                                                  +YBLE*(1.0-OMGAI)
          DPENG2(3,ISTAGB+INODE)=1.0
80        CONTINUE
C
          DPENG2(3,ISTAGA+NXUP)=0.0
          DPENG2(3,ISTAGB+NXUP)=0.0
C
C     ...DOWNSTREAM OF AIRFOIL
C
          ISTAGA=(NYLO     -1)*NXNODE+NXUP+NAIR
          ISTAGB=(NYLO+NGAP-2)*NXNODE+NXUP+NAIR
          DO 90 INODE=1,NXDN-1
          OMGAI=OMG(BXDN,INODE-1,NXDN-1)
C
          GEOMG2(1,ISTAGA+INODE)=XDN*OMGAI             +XATE*(1.0-OMGAI)
          GEOMG2(2,ISTAGA+INODE)=GEOMG2(2,ISTAGA+NXDN)*OMGAI
     &                                                  +YATE*(1.0-OMGAI)
          DPENG2(1,ISTAGA+INODE)=XDN*OMGAI             +XATE*(1.0-OMGAI)
          DPENG2(2,ISTAGA+INODE)=GEOMG2(2,ISTAGA+NXDN)*OMGAI
     &                                                  +YATE*(1.0-OMGAI)
          DPENG2(3,ISTAGA+INODE)=1.0
C
```

```
      GEOMG2(1,ISTAGB+INODE)=XDN*OMGAI            +XBTE*(1.0-OMGAI)
      GEOMG2(2,ISTAGB+INODE)=GEOMG2(2,ISTAGB+NXDN)*OMGAI
     &                                             +YBTE*(1.0-OMGAI)
      DPENG2(1,ISTAGB+INODE)=XDN*OMGAI            +XBTE*(1.0-OMGAI)
      DPENG2(2,ISTAGB+INODE)=GEOMG2(2,ISTAGB+NXDN)*OMGAI
     &                                             +YBTE*(1.0-OMGAI)
      DPENG2(3,ISTAGB+INODE)=1.0
90    CONTINUE
C
      DPENG2(3,ISTAGA+1)=0.0
      DPENG2(3,ISTAGB+1)=0.0
C
C     SET UP THE AIRFOIL SURFACE
C
C     ...LOWER SURFACE
C
      DO 100 I=1,NAIR
      INODEA=I+NXUP+(NYLO     -1)*NXNODE
      IBODYA=(NAIR-I+1)*NFACT+1
      FRACA=(GEOMG2(1,INODEA)-XALE)/(XATE-XALE)
C
      GEOMG2(1,INODEA)=BODYG2(1,IBODYA)
      GEOMG2(2,INODEA)=BODYG2(2,IBODYA)
      DPENG2(1,INODEA)=BODYG2(1,IBODYA)
      DPENG2(2,INODEA)=YALE*(1.0-FRACA)+YATE*FRACA
      DPENG2(3,INODEA)=0.0
C
      INODEB=I+NXUP+(NYLO+NGAP-2)*NXNODE
      IBODYB=(NAIR-I+1)*NFACT+1+NABODY
      FRACB=(GEOMG2(1,INODEB)-XBLE)/(XBTE-XBLE)
C
      GEOMG2(1,INODEB)=BODYG2(1,IBODYB)
      GEOMG2(2,INODEB)=BODYG2(2,IBODYB)
      DPENG2(1,INODEB)=BODYG2(1,IBODYB)
      DPENG2(2,INODEB)=YBLE*(1.0-FRACB)+YBTE*FRACB
      DPENG2(3,INODEB)=0.0
C
100   CONTINUE
C
C     ...UPPER SURFACE
C
      DO 110 I=1,NAIR
      INODEA=I+NYNODE*NXNODE
      IBODYA=(NAIR+I+1)*NFACT+1
      FRACA=(GEOMG2(1,INODEA)-XALE)/(XATE-XALE)
C
      GEOMG2(1,INODEA)=BODYG2(1,IBODYA)
      GEOMG2(2,INODEA)=BODYG2(2,IBODYA)
      DPENG2(1,INODEA)=BODYG2(1,IBODYA)
      DPENG2(2,INODEA)=YALE*(1.0-FRACA)+YATE*FRACA
      DPENG2(3,INODEA)=0.0
C
      INODEB=I+NAIR+NYNODE*NXNODE
```

```
        IBODYB=(NAIR+I+1)*NFACT+1+NABODY
        FRACB=(GEOMG2(1,INODEB)-XBLE)/(XBTE-XBLE)
C
        GEOMG2(1,INODEB)=BODYG2(1,IBODYB)
        GEOMG2(2,INODEB)=BODYG2(2,IBODYB)
        DPENG2(1,INODEB)=BODYG2(1,IBODYB)
        DPENG2(2,INODEB)=YBLE*(1.0-FRACB)+YBTE*FRACB
        DPENG2(3,INODEB)=0.0
110     CONTINUE
C
C*****SET UP ALL INTERIOR NODES USING AN ALGEBRAIC TECHNIQUE
C
C     ABOVE THE AIRFOIL
C
        DO 120 J=2,NYUP-1
C
        OMGL=REAL(J-1)/REAL(NYUP-1)
        OMGC=OMG(BYUP,J-1,NYUP-1)
        OMGR=REAL(J-1)/REAL(NYUP-1)
C
        DO 115 I=2,NXNODE-1
C
        IBOT =NXNODE*(NYLO+NGAP     -2)+I
        ITOP =NXNODE*(NYLO+NGAP+NYUP-3)+I
        INODE=NXNODE*(NYLO+NGAP+J   -3)+I
        IF(I.GT.NXUP .AND. I.LT.(NXNODE-NXDN))
     &                              IBOT=NXNODE*NYNODE+I-NXUP+NAIR
C
        IF(I.LT.NXUP) THEN
           FRAC=(DPENG2(1,IBOT)-XUP)/(XALE-XUP)
           OMGAY=OMGC*FRAC+OMGL*(1.0-FRAC)
           OMGAX=OMGAY**2
        ELSEIF(I.GT.NXUP+NAIR+1) THEN
           FRAC=(DPENG2(1,IBOT)-XDN)/(XATE-XDN)
           OMGAY=OMGC*FRAC+OMGR*(1.0-FRAC)
           OMGAX=OMGAY**2
        ELSE
           OMGAY=OMGC
           OMGAX=OMGAY**2
        ENDIF
C
        GEOMG2(1,INODE)=OMGAX*GEOMG2(1,ITOP)+(1.0-OMGAX)*GEOMG2(1,IBOT)
        GEOMG2(2,INODE)=OMGAY*GEOMG2(2,ITOP)+(1.0-OMGAY)*GEOMG2(2,IBOT)
        DPENG2(1,INODE)=OMGAX*DPENG2(1,ITOP)+(1.0-OMGAX)*DPENG2(1,IBOT)
        DPENG2(2,INODE)=OMGAY*DPENG2(2,ITOP)+(1.0-OMGAY)*DPENG2(2,IBOT)
        DPENG2(3,INODE)=1.00
115     CONTINUE
C
120     CONTINUE
C
C     BELOW THE AIRFOIL
C
        DO 130 J=2,NYLO-1
```

```
C
      OMGL=REAL(NYLO-J)/REAL(NYLO-1)
      OMGC=OMG(BYLO,NYLO-J,NYLO-1)
      OMGR=REAL(NYLO-J)/REAL(NYLO-1)
C
      DO 125 I=2,NXNODE-1
C
      IBOT =                 I
      ITOP =NXNODE*(NYLO-1)+I
      INODE=NXNODE*(J   -1)+I
C
      IF(I.LT.NXUP) THEN
         FRAC=(DPENG2(1,ITOP)-XUP)/(XALE-XUP)
         OMGAY=OMGC*FRAC+OMGL*(1.0-FRAC)
         OMGAX=OMGAY**2
      ELSEIF(I.GT.NXUP+NAIR+1) THEN
         FRAC=(DPENG2(1,ITOP)-XDN)/(XATE-XDN)
         OMGAY=OMGC*FRAC+OMGR*(1.0-FRAC)
         OMGAX=OMGAY**2
      ELSE
         OMGAY=OMGC
         OMGAX=OMGAY**2
      ENDIF
C
      GEOMG2(1,INODE)=OMGAX*GEOMG2(1,IBOT)+(1.0-OMGAX)*GEOMG2(1,ITOP)
      GEOMG2(2,INODE)=OMGAY*GEOMG2(2,IBOT)+(1.0-OMGAY)*GEOMG2(2,ITOP)
      DPENG2(1,INODE)=OMGAX*DPENG2(1,IBOT)+(1.0-OMGAX)*DPENG2(1,ITOP)
      DPENG2(2,INODE)=OMGAY*DPENG2(2,IBOT)+(1.0-OMGAY)*DPENG2(2,ITOP)
      DPENG2(3,INODE)=1.00
125   CONTINUE
C
130   CONTINUE
C
C     BETWEEN THE AIRFOILS
C
      DO 132 J=2,NGAP-1
      OMGA=REAL(J-1)/REAL(NGAP-1)
C
      DO 131 I=2,NXNODE-1
C
      IBOT =NXNODE*(NYLO     -1)+I
      ITOP =NXNODE*(NYLO+NGAP-2)+I
      INODE=NXNODE*(NYLO+J   -2)+I
      IF(I.GT.NXUP .AND. I.LT.(NXNODE-NXDN))
     &                          IBOT=NXNODE*NYNODE+I-NXUP
C
      GEOMG2(1,INODE)=OMGA*GEOMG2(1,ITOP)+(1.0-OMGA)*GEOMG2(1,IBOT)
      GEOMG2(2,INODE)=OMGA*GEOMG2(2,ITOP)+(1.0-OMGA)*GEOMG2(2,IBOT)
      DPENG2(1,INODE)=OMGA*DPENG2(1,ITOP)+(1.0-OMGA)*DPENG2(1,IBOT)
      DPENG2(2,INODE)=OMGA*DPENG2(2,ITOP)+(1.0-OMGA)*DPENG2(2,IBOT)
      DPENG2(3,INODE)=1.00
131   CONTINUE
C
```

```
132     CONTINUE
C
C*****SET UP POINTER SYSTEM (DATA STRUCTURE) FOR CELLS ON ALL LEVELS
C
C     INITIALIZE THE NUMBER OF CELLS AND BOUNDARY CONDITION POINTERS
C
        NCELG2=0
        NBNDG2=0
C
C     COMPUTE INDICES OF FIRST AND LAST POINTS WHICH ARE ON THE LOWER
C       AIRFOIL SURFACE SO THAT THEY CAN BE TRANSFERRED TO POINT TO
C       UPPER SURFACE NODES (IF ALONG THE SOUTH SIDE OF A CELL)
C
        IBEGA=NXUP+   1+(NYLO     -1)*NXNODE
        IENDA=NXUP+NAIR+(NYLO     -1)*NXNODE
        IOFFA=(NYUP+NGAP-1)*NXNODE-NXUP
C
        IBEGB=NXUP+   1+(NYLO+NGAP-2)*NXNODE
        IENDB=NXUP+NAIR+(NYLO+NGAP-2)*NXNODE
        IOFFB=NYUP*NXNODE-NXUP+NAIR
C
C     INITIALIZE POINTERS FOR ALL LEVELS
C
        DO 140 ILEVEL=-MLVLG2,MLVLG2
        ILVLG2(1,ILEVEL)=1
        ILVLG2(2,ILEVEL)=0
        ILVLG2(3,ILEVEL)=1
        ILVLG2(4,ILEVEL)=0
        ILVLG2(5,ILEVEL)=1
        ILVLG2(6,ILEVEL)=0
140     CONTINUE
C
C     LOOP THROUGH ALL COARSER GRID LEVELS (IF ANY)
C
        DO 160 ICOARS=MIN(NCOARS,MLVLG2-1),1,-1
        ISIZE=2**ICOARS
C
C     LOOP THROUGH EACH CELL ON THIS LEVEL
C
        DO 150 JCELL=1,NYCELL,ISIZE
        DO 150 ICELL=1,NXCELL,ISIZE
C
        NCELG2=NCELG2+1
C
C     FIND THE CENTER OF THIS CELL
C
        ICELG2(1,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE/2-1)*NXNODE
C
C     COMPUTE INDICES OF ALL BOUNDING NODES
C
        ICELG2(2,NCELG2)=(ICELL        )+(JCELL        -1)*NXNODE
        ICELG2(3,NCELG2)=(ICELL+ISIZE/2)+(JCELL        -1)*NXNODE
        ICELG2(4,NCELG2)=(ICELL+ISIZE  )+(JCELL        -1)*NXNODE
```

```
         ICELG2(5,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE/2-1)*NXNODE
         ICELG2(6,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE  -1)*NXNODE
         ICELG2(7,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE  -1)*NXNODE
         ICELG2(8,NCELG2)=(ICELL        )+(JCELL+ISIZE  -1)*NXNODE
         ICELG2(9,NCELG2)=(ICELL        )+(JCELL+ISIZE/2-1)*NXNODE
C
C     ADJUST SOUTH NODES TO POINT TO UPPER SURFACE OF AIRFOIL INSTEAD
C
         IF(ICELG2(2,NCELG2).GE.IBEGA .AND. ICELG2(2,NCELG2).LE.IENDA)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFA
         IF(ICELG2(3,NCELG2).GE.IBEGA .AND. ICELG2(3,NCELG2).LE.IENDA)
     &                          ICELG2(3,NCELG2)=ICELG2(3,NCELG2)+IOFFA
         IF(ICELG2(4,NCELG2).GE.IBEGA .AND. ICELG2(4,NCELG2).LE.IENDA)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFA
C
         IF(ICELG2(2,NCELG2).GE.IBEGB .AND. ICELG2(2,NCELG2).LE.IENDB)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFB
         IF(ICELG2(3,NCELG2).GE.IBEGB .AND. ICELG2(3,NCELG2).LE.IENDB)
     &                          ICELG2(3,NCELG2)=ICELG2(3,NCELG2)+IOFFB
         IF(ICELG2(4,NCELG2).GE.IBEGB .AND. ICELG2(4,NCELG2).LE.IENDB)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFB
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
         ICELG2(10,NCELG2)=0
C
C      THE NEIGHBOR INFORMATION IS NOT USED ON COARSE GLOBAL LEVELS,
C        SO SET THEM ALL TO ZERO
C
         NBORG2(1,NCELG2)=0
         NBORG2(2,NCELG2)=0
         NBORG2(3,NCELG2)=0
         NBORG2(4,NCELG2)=0
         NBORG2(5,NCELG2)=0
         NBORG2(6,NCELG2)=0
         NBORG2(7,NCELG2)=0
         NBORG2(8,NCELG2)=0
         NBORG2(9,NCELG2)=0
C
150      CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THIS LEVEL
C
         ILEVEL=-ICOARS
         ILVLG2(1,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
         ILVLG2(2,ILEVEL)=ILVLG2(6,ILEVEL-1)
         ILVLG2(3,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
         ILVLG2(4,ILEVEL)=ILVLG2(6,ILEVEL-1)
         ILVLG2(5,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
         ILVLG2(6,ILEVEL)=NCELG2
C
C     GO BACK FOR A FINER COARSE LEVEL
C
```

```
160     CONTINUE
C
C     LOOP THROUGH EACH GLOBAL CELL
C
        IOFFN=NCELG2-NXCELL
C
        DO 170 JCELL=1,NYCELL
        DO 170 ICELL=1,NXCELL
C
        NCELG2=NCELG2+1
C
C     COMPUTE INDICES OF EACH CORNER OF CELL
C
        ICELG2(2,NCELG2)=ICELL  +(JCELL-1)*NXNODE
        ICELG2(4,NCELG2)=ICELL+1+(JCELL-1)*NXNODE
        ICELG2(6,NCELG2)=ICELL+1+(JCELL  )*NXNODE
        ICELG2(8,NCELG2)=ICELL  +(JCELL  )*NXNODE
C
C     ADJUST SOUTH NODES TO POINT TO UPPER SURFACE OF AIRFOIL INSTEAD
C
        IF(ICELG2(2,NCELG2).GE.IBEGA .AND. ICELG2(2,NCELG2).LE.IENDA)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFA
        IF(ICELG2(4,NCELG2).GE.IBEGA .AND. ICELG2(4,NCELG2).LE.IENDA)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFA
C
        IF(ICELG2(2,NCELG2).GE.IBEGB .AND. ICELG2(2,NCELG2).LE.IENDB)
     &                          ICELG2(2,NCELG2)=ICELG2(2,NCELG2)+IOFFB
        IF(ICELG2(4,NCELG2).GE.IBEGB .AND. ICELG2(4,NCELG2).LE.IENDB)
     &                          ICELG2(4,NCELG2)=ICELG2(4,NCELG2)+IOFFB
C
C     INITIALLY, THERE IS NO NODE IN THE CENTER OF A FINE CELL
C
        ICELG2(1,NCELG2)=0
C
C     THERE ARE NO NODES IN THE CENTER OF THE SIDES OF A FINE CELL
C
        ICELG2(3,NCELG2)=0
        ICELG2(5,NCELG2)=0
        ICELG2(7,NCELG2)=0
        ICELG2(9,NCELG2)=0
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C     INITIALIZE ALL NEIGHBOR INFORMATION FOR THIS LEVEL
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL-1)
        NBORG2(3,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL  )
        NBORG2(4,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL+1)
        NBORG2(5,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL+1)
        NBORG2(6,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL+1)
```

```
      NBORG2(7,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL  )
      NBORG2(8,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL-1)
      NBORG2(9,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL-1)
C
C     CORRECT NEIGHBOR INFORMATION FOR BOUNDARY CELLS
C
      IF(ICELL.EQ.1) THEN
         NBORG2(2,NCELG2)=0
         NBORG2(8,NCELG2)=0
         NBORG2(9,NCELG2)=0
      ENDIF
C
      IF(ICELL.EQ.NXCELL) THEN
         NBORG2(4,NCELG2)=0
         NBORG2(5,NCELG2)=0
         NBORG2(6,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.1) THEN
         NBORG2(2,NCELG2)=0
         NBORG2(3,NCELG2)=0
         NBORG2(4,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYCELL) THEN
         NBORG2(6,NCELG2)=0
         NBORG2(7,NCELG2)=0
         NBORG2(8,NCELG2)=0
      ENDIF
C
C     CORRECT NEIGHBOR INFORMATION FOR CELLS AROUND AIRFOIL
C
      IF(JCELL.EQ.NYLO-1 .AND. ICELL.EQ.NXUP) THEN
         NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
         NBORG2(6,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO-1 .AND. ICELL.GT.NXUP
     &                   .AND. ICELL.LT.NXUP+NAIR) THEN
         NBORG2(6,NCELG2)=0
         NBORG2(7,NCELG2)=0
         NBORG2(8,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO-1 .AND. ICELL.EQ.NXUP+NAIR) THEN
         NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
         NBORG2(8,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO   .AND. ICELL.EQ.NXUP) THEN
         NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
         NBORG2(4,NCELG2)=0
      ENDIF
```

```
C
      IF(JCELL.EQ.NYLO    .AND. ICELL.GT.NXUP
     &                    .AND. ICELL.LT.NXUP+NAIR) THEN
         NBORG2(2,NCELG2)=0
         NBORG2(3,NCELG2)=0
         NBORG2(4,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO    .AND. ICELL.EQ.NXUP+NAIR) THEN
         NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
         NBORG2(2,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO+NGAP-2 .AND. ICELL.EQ.NXUP) THEN
         NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
         NBORG2(6,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO+NGAP-2 .AND. ICELL.GT.NXUP
     &                        .AND. ICELL.LT.NXUP+NAIR) THEN
         NBORG2(6,NCELG2)=0
         NBORG2(7,NCELG2)=0
         NBORG2(8,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO+NGAP-2 .AND. ICELL.EQ.NXUP+NAIR) THEN
         NBORG2(7,NCELG2)=-NBORG2(7,NCELG2)
         NBORG2(8,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO+NGAP-1 .AND. ICELL.EQ.NXUP) THEN
         NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
         NBORG2(4,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO+NGAP-1 .AND. ICELL.GT.NXUP
     &                        .AND. ICELL.LT.NXUP+NAIR) THEN
         NBORG2(2,NCELG2)=0
         NBORG2(3,NCELG2)=0
         NBORG2(4,NCELG2)=0
      ENDIF
C
      IF(JCELL.EQ.NYLO+NGAP-1 .AND. ICELL.EQ.NXUP+NAIR) THEN
         NBORG2(3,NCELG2)=-NBORG2(3,NCELG2)
         NBORG2(2,NCELG2)=0
      ENDIF
C
170   CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THE GLOBAL FINE LEVEL
C
      ILVLG2(1,0)=ILVLG2(6,-1)+1
      ILVLG2(2,0)=NCELG2
```

```
        ILVLG2(3,0)=NCELG2+1
        ILVLG2(4,0)=NCELG2
        ILVLG2(5,0)=NCELG2+1
        ILVLG2(6,0)=NCELG2
C
C     INITIALIZE THE MULTIPLE-GRID-LEVEL ARRAY FOR ALL EMBEDDED MESHES
C
        DO 180 ILEVEL=1,MLVLG2+1
        ILVLG2(1,ILEVEL)=NCELG2+1
        ILVLG2(2,ILEVEL)=NCELG2
        ILVLG2(3,ILEVEL)=NCELG2+1
        ILVLG2(4,ILEVEL)=NCELG2
        ILVLG2(5,ILEVEL)=NCELG2+1
        ILVLG2(6,ILEVEL)=NCELG2
180     CONTINUE
C
C*****ADD BOUNDARY CONDITION INFORMATION TO POINTER SYSTEM
C
C     SET UP THE BOUNDARY CONDITION ARRAYS AND SET BOUNDARY POINTERS FOR
C        BOUNDARY NODES AT THE FAR-FIELD
C        ...FOR SOUTHWESTERN CORNER
C
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=1
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+1
        IBNDG2(3,NBNDG2)=0
        IBNDG2(4,NBNDG2)=11
        IBNDG2(5,NBNDG2)=4
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IWEND)
        BONDG2(2,NBNDG2)=BODYG2( 9,IWEND)
        BONDG2(3,NBNDG2)=BODYG2(10,ISBEG)
        BONDG2(4,NBNDG2)=BODYG2(11,ISBEG)
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HLOOOB)
C
C        ...FOR SOUTHERN EDGE
C
        DO 190 IBOUND=2,NXNODE-1
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=IBOUND
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+IBOUND-1
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+IBOUND
        IBNDG2(4,NBNDG2)=3
        IBNDG2(5,NBNDG2)=4
        IBNDG2(6,NBNDG2)=0
C
        IBODY=(IBOUND-1)*NFACT+ISBEG
        BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
```

```
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
190   CONTINUE
C
C       ...FOR SOUTHEASTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=7
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,ISEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,ISEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IEBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IEBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0007)
C
C       ...FOR EASTERN EDGE
C
      DO 200 IBOUND=2,NYNODE-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE* IBOUND
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL* IBOUND
      IBNDG2(4,NBNDG2)=6
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(IBOUND-1)*NFACT+IEBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0006)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0006)
C
```

```
200     CONTINUE
C
C        ...FOR NORTHEASTERN CORNER
C
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=NXNODE*NYNODE
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*NYCELL
        IBNDG2(3,NBNDG2)=0
        IBNDG2(4,NBNDG2)=14
        IBNDG2(5,NBNDG2)=4
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IEEND)
        BONDG2(2,NBNDG2)=BODYG2( 9,IEEND)
        BONDG2(3,NBNDG2)=BODYG2(10,INBEG)
        BONDG2(4,NBNDG2)=BODYG2(11,INBEG)
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HLOOOE)
C
C        ...FOR NORTHERN EDGE
C
        DO 210 IBOUND=NXNODE-1,2,-1
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+IBOUND
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND-1
        IBNDG2(4,NBNDG2)=12
        IBNDG2(5,NBNDG2)=4
        IBNDG2(6,NBNDG2)=0
C
        IBODY=(NXNODE-IBOUND)*NFACT+INBEG
        BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
        BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
        BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
        BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HLOOOC)
        ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HLOOOC)
C
210     CONTINUE
C
C        ...FOR NORTHWESTERN CORNER
C
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+1
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+1
        IBNDG2(3,NBNDG2)=0
```

```
      IBNDG2(4,NBNDG2)=13
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,INEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,INEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IWBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IWBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000D)
C
C       ...FOR WESTERN EDGE
C
      DO 220 IBOUND=NYNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(IBOUND-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)+1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-2)+1
      IBNDG2(4,NBNDG2)=9
      IBNDG2(5,NBNDG2)=4
      IBNDG2(6,NBNDG2)=0
C
      IBODY=(NYNODE-IBOUND)*NFACT+IWBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0009)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0009)
C
220   CONTINUE
C
C       ...FOR LEADING EDGE
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXUP+(NYLO-1)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-2)*NXCELL+NXUP
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-1)*NXCELL+NXUP
      IBNDG2(4,NBNDG2)=5
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 6,IALE)
      BONDG2(2,NBNDG2)=BODYG2( 9,IALE)
      BONDG2(3,NBNDG2)=BODYG2(10,IALE)
      BONDG2(4,NBNDG2)=BODYG2(11,IALE)
C
```

```
      ICELG2(10,IBNDG2(2,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)-1),HL0004)
      ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2)-1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)-1),HL0002)
      ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL0003)
C
C       ...FOR TRAILING EDGE
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXUP+NAIR+1+(NYLO-1)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO-1)*NXCELL+NXUP+NAIR
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO-2)*NXCELL+NXUP+NAIR
      IBNDG2(4,NBNDG2)=10
      IBNDG2(5,NBNDG2)=1
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,IAEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,IAEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IABEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IABEG)
C
      ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)+1),HL0008)
      ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL0003)
      ICELG2(10,IBNDG2(2,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)+1),HL0001)
C
C       ...LOWER SURFACE OF AIRFOIL
C
      DO 230 IBOUND=1,NAIR
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXUP+(NYLO-1)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-2)*NXCELL
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-2)*NXCELL-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IALE-IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
```

```
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
230   CONTINUE
C
C      ...UPPER SURFACE OF AIRFOIL
C
      DO 240 IBOUND=1,NAIR
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXNODE*NYNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-1)*NXCELL-1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO-1)*NXCELL
      IBNDG2(4,NBNDG2)=3
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IALE+IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
240   CONTINUE
C
C      ...FOR LEADING EDGE
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXUP+(NYLO+NGAP-2)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO+NGAP-3)*NXCELL+NXUP
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO+NGAP-2)*NXCELL+NXUP
      IBNDG2(4,NBNDG2)=5
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,IBLE)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBLE)
      BONDG2(3,NBNDG2)=BODYG2(10,IBLE)
      BONDG2(4,NBNDG2)=BODYG2(11,IBLE)
C
      ICELG2(10,IBNDG2(2,NBNDG2)-1)
     &                     =IOR(ICELG2(10,IBNDG2(2,NBNDG2)-1),HL0004)
      ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                     =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2)-1)
     &                     =IOR(ICELG2(10,IBNDG2(3,NBNDG2)-1),HL0002)
```

```
      ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL0003)
C
C       ...FOR TRAILING EDGE
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXUP+NAIR+1+(NYLO+NGAP-2)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+(NYLO+NGAP-2)*NXCELL+NXUP+NAIR
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+(NYLO+NGAP-3)*NXCELL+NXUP+NAIR
      IBNDG2(4,NBNDG2)=10
      IBNDG2(5,NBNDG2)=1
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,IBEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IBBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IBBEG)
C
      ICELG2(10,IBNDG2(3,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)  ),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)+1),HL0008)
      ICELG2(10,IBNDG2(2,NBNDG2)  )
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)  ),HL0003)
      ICELG2(10,IBNDG2(2,NBNDG2)+1)
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)+1),HL0001)
C
C       ...LOWER SURFACE OF AIRFOIL
C
      DO 235 IBOUND=1,NAIR
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=IBOUND+NXUP+(NYLO+NGAP-2)*NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO+NGAP-3)*NXCELL
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO+NGAP-3)*NXCELL-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=3
      IBNDG2(6,NBNDG2)=0
C
      IBODY=IBLE-IBOUND*NFACT
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
235   CONTINUE
C
```

```
C         ...UPPER SURFACE OF AIRFOIL
C
        DO 245 IBOUND=1,NAIR
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=IBOUND+NXNODE*NYNODE+NAIR
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO+NGAP-2)*NXCELL-1
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXUP+IBOUND+(NYLO+NGAP-2)*NXCELL
        IBNDG2(4,NBNDG2)=3
        IBNDG2(5,NBNDG2)=3
        IBNDG2(6,NBNDG2)=0
C
        IBODY=IBLE+IBOUND*NFACT
        BONDG2(1,NBNDG2)=BODYG2( 6,IBODY)
        BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
        BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
        BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
        ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
245     CONTINUE
C
C*****USE LAPLACE'S EQUATION FOR SMOOTH GRID VARIATION
C
        CALL G2LAPL(NITER,IOUT,1,IER)
C
C     UN-COMMENT OUT THE NEXT FOUR LINES TO LOOK AT THE XI-ETA
C        DISTRIBUTION USED IN THE GRID GENERATION
C
C       DO 246 INODE=1,NNODG2
C       GEOMG2(1,INODE)=DPENG2(1,INODE)
C       GEOMG2(2,INODE)=DPENG2(2,INODE)
C246    CONTINUE
C
C*****INITIALIZE THE DEPENDENT VARIABLES AT ALL NODES AND BOUNDARY
C        POINTERS
C
        DO 250 INODE=1,NNODG2
        DPENG2(1,INODE)=DPEN1
        DPENG2(2,INODE)=DPEN2
        DPENG2(3,INODE)=DPEN3
        DPENG2(4,INODE)=DPEN4
        DPENG2(5,INODE)=DPEN5
250     CONTINUE
C
        DO 260 IBODY=1,NBODG2
        BODYG2(3,IBODY)=DPEN1
        BODYG2(4,IBODY)=DPEN2
        BODYG2(5,IBODY)=DPEN3
        BODYG2(6,IBODY)=DPEN4
```

```
        BODYG2(7,IBODY)=DPEN5
260     CONTINUE
C
        DO 270 IBOUND=1,NBNDG2
        BONDG2(5,IBOUND)=DPEN1
        BONDG2(6,IBOUND)=DPEN2
        BONDG2(7,IBOUND)=DPEN3
        BONDG2(8,IBOUND)=DPEN4
        BONDG2(9,IBOUND)=DPEN5
270     CONTINUE
C
C     PRE-EMBED NEAR AIRFOIL SURFACE IF DESIRED
C
        DO 280 ISURF=1,NSURF
        CALL G2SURF(1,3,NCELL)
280     CONTINUE
C
C     GROW THE EMBEDDED REGION IF DESIRED
C
        IF(NGROW1.NE.0) CALL G2GROW(NGROW1,NGROW2,NCELL)
C
C*****SORT THE CELLS
C
        CALL G2SORT
C
        RETURN
        END
```

G2INP7

```
        SUBROUTINE G2INP7(IUNIT,IOUT
     &
     &                                )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A TYPE 7 GEOMETRY DEFINITION, WHICH
C       IS A SIMPLY-CONNECTED RECTANGLE.  THE RECTANGLE'S NODES
C       ARE READ IN ((FROM WEST TO EAST) FROM SOUTH TO NORTH).
C       THE DATA STRUCTURE IS THEN INITAILIZED.
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
C     READ GLOBAL MESH SIZE AND OTHER CONTROLLING PARAMETERS
```

```
C
      READ(IUNIT,10) NXNODE,NYNODE,              NCOARS,
     &               IBCSW ,IBCS  ,IBCSE ,IBCE  ,
     &               IBCNE ,IBCN  ,IBCNW ,IBCW
10    FORMAT(2I5,10X,9I5)
C
      KDEFG2=0
C
C     COMPUTE NUMBER OF POINTS INPUT IN EACH DIRECTION
C
      NFACT =2**KDEFG2
      NXBODY=1+(NXNODE-1)*NFACT
      NYBODY=1+(NYNODE-1)*NFACT
C
C     COMPUTE TOTAL NUMBER OF GLOBAL NODES
C
      NNODG2=NXNODE*NYNODE
C
C     COMPUTE NUMBER OF CELLS IN EACH DIRECTION ON THE GLOBAL MESH
C
      NXCELL=NXNODE-1
      NYCELL=NYNODE-1
C
C     READ THE NODAL INFORMATION
C
      READ(IUNIT,20) (GEOMG2(1,I),GEOMG2(2,I),
     &                DPENG2(1,I),DPENG2(2,I),
     &                DPENG2(3,I),DPENG2(4,I),
     &                DPENG2(5,I),            I=1,NNODG2)
20    FORMAT(7F10.0)
C
C     SET UP THE SPLINE COEFFICIENTS
C
      ISBEG=     +1
      ISEND=     +NXBODY
      INBEG=ISEND+1
      INEND=ISEND+NXBODY
      IEBEG=INEND+1
      IEEND=INEND+NYBODY
      IWBEG=IEEND+1
      IWEND=IEEND+NYBODY
C
C     ...SET UP THE SOUTHERN BOUNDARY AND SPLINE FIT IT
C
      DO 30 IPNT=1,NXBODY
      IBODY=ISBEG-1+IPNT
      INODE=(IPNT-1)/NFACT+1
      BODYG2(1,IBODY)=GEOMG2(1,INODE)
      BODYG2(2,IBODY)=GEOMG2(2,INODE)
      BODYG2(3,IBODY)=DPENG2(1,INODE)
      BODYG2(4,IBODY)=DPENG2(2,INODE)
      BODYG2(5,IBODY)=DPENG2(3,INODE)
      BODYG2(6,IBODY)=DPENG2(4,INODE)
```

```
      BODYG2(7,IBODY)=DPENG2(5,INODE)
30    CONTINUE
C
      CALL G2BOND(ISBEG,ISEND,1)
C
C     ...SET UP THE NORTHERN BOUNDARY AND SPLINE FIT IT
C
      DO 40 IPNT=1,NXBODY
      IBODY=INEND+1-IPNT
      INODE=(IPNT-1)/NFACT+1+NXNODE*(NYNODE-1)
      BODYG2(1,IBODY)=GEOMG2(1,INODE)
      BODYG2(2,IBODY)=GEOMG2(2,INODE)
      BODYG2(3,IBODY)=DPENG2(1,INODE)
      BODYG2(4,IBODY)=DPENG2(2,INODE)
      BODYG2(5,IBODY)=DPENG2(3,INODE)
      BODYG2(6,IBODY)=DPENG2(4,INODE)
      BODYG2(7,IBODY)=DPENG2(5,INODE)
40    CONTINUE
C
      CALL G2BOND(INBEG,INEND,1)
C
C     ...SET UP THE WESTERN BOUNDARY AND SPLINE FIT IT
C
      DO 50 IPNT=1,NYBODY
      IBODY=IWEND+1-IPNT
      INODE=((IPNT-1)/NFACT+1)*NXNODE+1-NXNODE
      BODYG2(1,IBODY)=GEOMG2(1,INODE)
      BODYG2(2,IBODY)=GEOMG2(2,INODE)
      BODYG2(3,IBODY)=DPENG2(1,INODE)
      BODYG2(4,IBODY)=DPENG2(2,INODE)
      BODYG2(5,IBODY)=DPENG2(3,INODE)
      BODYG2(6,IBODY)=DPENG2(4,INODE)
      BODYG2(7,IBODY)=DPENG2(5,INODE)
      CALL G2BOND(IWBEG,IWEND,1)
C
50    CONTINUE
C
C     ...SET UP THE EASTERN BOUNDARY AND SPLINE FIT IT
C
      DO 60 IPNT=1,NYBODY
      IBODY=IEBEG-1+IPNT
      INODE=((IPNT-1)/NFACT+1)*NXNODE
      BODYG2(1,IBODY)=GEOMG2(1,INODE)
      BODYG2(2,IBODY)=GEOMG2(2,INODE)
      BODYG2(3,IBODY)=DPENG2(1,INODE)
      BODYG2(4,IBODY)=DPENG2(2,INODE)
      BODYG2(5,IBODY)=DPENG2(3,INODE)
      BODYG2(6,IBODY)=DPENG2(4,INODE)
      BODYG2(7,IBODY)=DPENG2(5,INODE)
60    CONTINUE
C
      CALL G2BOND(IEBEG,IEEND,1)
C
```

```
        NBODG2=2*(NXBODY+NYBODY)
C
C*****SET UP POINTER SYSTEM (DATA STRUCTURE) FOR CELLS ON ALL LEVELS
C
C     INITIALIZE THE NUMBER OF CELLS AND BOUNDARY CONDITION POINTERS
C
        NCELG2=0
        NBNDG2=0
C
C     INITIALIZE POINTERS FOR ALL LEVELS
C
        DO 100 ILEVEL=-MLVLG2,MLVLG2
        ILVLG2(1,ILEVEL)=1
        ILVLG2(2,ILEVEL)=0
        ILVLG2(3,ILEVEL)=1
        ILVLG2(4,ILEVEL)=0
        ILVLG2(5,ILEVEL)=1
        ILVLG2(6,ILEVEL)=0
100     CONTINUE
C
C     LOOP THROUGH ALL COARSER GRID LEVELS (IF ANY)
C
        DO 120 ICOARS=MIN(NCOARS,MLVLG2-1),1,-1
        ISIZE=2**ICOARS
C
C     LOOP THROUGH EACH CELL ON THIS LEVEL
C
        DO 110 JCELL=1,NYCELL,ISIZE
        DO 110 ICELL=1,NXCELL,ISIZE
C
        NCELG2=NCELG2+1
C
C     FIND THE CENTER OF THIS CELL
C
        ICELG2(1,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE/2-1)*NXNODE
C
C     COMPUTE INDICES OF ALL BOUNDING NODES
C
        ICELG2(2,NCELG2)=(ICELL        )+(JCELL        -1)*NXNODE
        ICELG2(3,NCELG2)=(ICELL+ISIZE/2)+(JCELL        -1)*NXNODE
        ICELG2(4,NCELG2)=(ICELL+ISIZE  )+(JCELL        -1)*NXNODE
        ICELG2(5,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE/2-1)*NXNODE
        ICELG2(6,NCELG2)=(ICELL+ISIZE  )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(7,NCELG2)=(ICELL+ISIZE/2)+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(8,NCELG2)=(ICELL        )+(JCELL+ISIZE  -1)*NXNODE
        ICELG2(9,NCELG2)=(ICELL        )+(JCELL+ISIZE/2-1)*NXNODE
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C      THE NEIGHBOR INFORMATION IS NOT USED ON COARSE GLOBAL LEVELS,
C         SO SET THEM ALL TO ZERO
```

```
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=0
        NBORG2(3,NCELG2)=0
        NBORG2(4,NCELG2)=0
        NBORG2(5,NCELG2)=0
        NBORG2(6,NCELG2)=0
        NBORG2(7,NCELG2)=0
        NBORG2(8,NCELG2)=0
        NBORG2(9,NCELG2)=0
C
110     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THIS LEVEL
C
        ILEVEL=-ICOARS
        ILVLG2(1,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(2,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(3,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(4,ILEVEL)=ILVLG2(6,ILEVEL-1)
        ILVLG2(5,ILEVEL)=ILVLG2(6,ILEVEL-1)+1
        ILVLG2(6,ILEVEL)=NCELG2
C
C     GO BACK FOR A FINER COARSE LEVEL
C
120     CONTINUE
C
C     LOOP THROUGH EACH GLOBAL CELL
C
        IOFFN=NCELG2-NXCELL
C
        DO 130 JCELL=1,NYCELL
        DO 130 ICELL=1,NXCELL
C
        NCELG2=NCELG2+1
C
C     COMPUTE INDICES OF EACH CORNER OF CELL
C
        ICELG2(2,NCELG2)=ICELL  +(JCELL-1)*NXNODE
        ICELG2(4,NCELG2)=ICELL+1+(JCELL-1)*NXNODE
        ICELG2(6,NCELG2)=ICELL+1+(JCELL  )*NXNODE
        ICELG2(8,NCELG2)=ICELL  +(JCELL  )*NXNODE
C
C     INITIALLY, THERE IS NO NODE IN THE CENTER OF A FINE CELL
C
        ICELG2(1,NCELG2)=0
C
C     THERE ARE NO NODES IN THE CENTER OF THE SIDES OF A FINE CELL
C
        ICELG2(3,NCELG2)=0
        ICELG2(5,NCELG2)=0
        ICELG2(7,NCELG2)=0
        ICELG2(9,NCELG2)=0
```

```
C
C     INITIALIZE AUXILIARY CELL INFORMATION
C
        ICELG2(10,NCELG2)=0
C
C     INITIALIZE ALL NEIGHBOR INFORMATION FOR THIS LEVEL
C
        NBORG2(1,NCELG2)=0
        NBORG2(2,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL-1)
        NBORG2(3,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL  )
        NBORG2(4,NCELG2)=IOFFN+(JCELL-1)*NXCELL+(ICELL+1)
        NBORG2(5,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL+1)
        NBORG2(6,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL+1)
        NBORG2(7,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL  )
        NBORG2(8,NCELG2)=IOFFN+(JCELL+1)*NXCELL+(ICELL-1)
        NBORG2(9,NCELG2)=IOFFN+(JCELL  )*NXCELL+(ICELL-1)
C
C     CORRECT NEIGHBOR INFORMATION FOR BOUNDARY CELLS
C
        IF(ICELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(8,NCELG2)=0
           NBORG2(9,NCELG2)=0
        ENDIF
C
        IF(ICELL.EQ.NXCELL) THEN
           NBORG2(4,NCELG2)=0
           NBORG2(5,NCELG2)=0
           NBORG2(6,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.1) THEN
           NBORG2(2,NCELG2)=0
           NBORG2(3,NCELG2)=0
           NBORG2(4,NCELG2)=0
        ENDIF
C
        IF(JCELL.EQ.NYCELL) THEN
           NBORG2(6,NCELG2)=0
           NBORG2(7,NCELG2)=0
           NBORG2(8,NCELG2)=0
        ENDIF
C
130     CONTINUE
C
C     SET UP THE MULTIPLE-GRID-LEVEL ARRAY FOR THE GLOBAL FINE LEVEL
C
        ILVLG2(1,0)=ILVLG2(6,-1)+1
        ILVLG2(2,0)=NCELG2
        ILVLG2(3,0)=NCELG2+1
        ILVLG2(4,0)=NCELG2
        ILVLG2(5,0)=NCELG2+1
        ILVLG2(6,0)=NCELG2
```

```
C
C     INITIALIZE THE MULTIPLE-GRID-LEVEL ARRAY FOR ALL EMBEDDED MESHES
C
        DO 140 ILEVEL=1,MLVLG2+1
        ILVLG2(1,ILEVEL)=NCELG2+1
        ILVLG2(2,ILEVEL)=NCELG2
        ILVLG2(3,ILEVEL)=NCELG2+1
        ILVLG2(4,ILEVEL)=NCELG2
        ILVLG2(5,ILEVEL)=NCELG2+1
        ILVLG2(6,ILEVEL)=NCELG2
140     CONTINUE
C
C*****ADD BOUNDARY CONDITION INFORMATION TO POINTER SYSTEM
C
C     SET UP THE BOUNDARY CONDITION ARRAYS AND SET BOUNDARY POINTERS FOR
C        BOUNDARY NODES...
C        ...FOR SOUTHWESTERN CORNER
C
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=1
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+1
        IBNDG2(3,NBNDG2)=0
        IBNDG2(4,NBNDG2)=11
        IBNDG2(5,NBNDG2)=IBCSW
        IBNDG2(6,NBNDG2)=0
C
        BONDG2(1,NBNDG2)=BODYG2( 8,IWEND)
        BONDG2(2,NBNDG2)=BODYG2( 9,IWEND)
        BONDG2(3,NBNDG2)=BODYG2(10,ISBEG)
        BONDG2(4,NBNDG2)=BODYG2(11,ISBEG)
        BONDG2(5,NBNDG2)=BODYG2( 3,ISBEG)
        BONDG2(6,NBNDG2)=BODYG2( 4,ISBEG)
        BONDG2(7,NBNDG2)=BODYG2( 5,ISBEG)
        BONDG2(8,NBNDG2)=BODYG2( 6,ISBEG)
        BONDG2(9,NBNDG2)=BODYG2( 7,ISBEG)
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HLOOOB)
C
C        ...FOR SOUTHERN EDGE
C
        DO 150 IBOUND=2,NXNODE-1
        NBNDG2=NBNDG2+1
C
        IBNDG2(1,NBNDG2)=IBOUND
        IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+IBOUND-1
        IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+IBOUND
        IBNDG2(4,NBNDG2)=3
        IBNDG2(5,NBNDG2)=IBCS
        IBNDG2(6,NBNDG2)=0
C
        IBODY=(IBOUND-1)*NFACT+ISBEG
```

```
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
      BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
      BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
      BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
      BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
      BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0003)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0003)
C
150   CONTINUE
C
C       ...FOR SOUTHEASTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=7
      IBNDG2(5,NBNDG2)=IBCSE
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,ISEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,ISEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IEBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IEBEG)
      BONDG2(5,NBNDG2)=BODYG2( 3,IEBEG)
      BONDG2(6,NBNDG2)=BODYG2( 4,IEBEG)
      BONDG2(7,NBNDG2)=BODYG2( 5,IEBEG)
      BONDG2(8,NBNDG2)=BODYG2( 6,IEBEG)
      BONDG2(9,NBNDG2)=BODYG2( 7,IEBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0007)
C
C       ...FOR EASTERN EDGE
C
      DO 160 IBOUND=2,NYNODE-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE* IBOUND
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL* IBOUND
      IBNDG2(4,NBNDG2)=6
      IBNDG2(5,NBNDG2)=IBCE
      IBNDG2(6,NBNDG2)=0
C
```

```
      IBODY=(IBOUND-1)*NFACT+IEBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
      BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
      BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
      BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
      BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
      BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0006)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0006)
C
160   CONTINUE
C
C       ...FOR NORTHEASTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*NYNODE
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*NYCELL
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=14
      IBNDG2(5,NBNDG2)=IBCNE
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,IEEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,IEEND)
      BONDG2(3,NBNDG2)=BODYG2(10,INBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,INBEG)
      BONDG2(5,NBNDG2)=BODYG2( 3,INBEG)
      BONDG2(6,NBNDG2)=BODYG2( 4,INBEG)
      BONDG2(7,NBNDG2)=BODYG2( 5,INBEG)
      BONDG2(8,NBNDG2)=BODYG2( 6,INBEG)
      BONDG2(9,NBNDG2)=BODYG2( 7,INBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000E)
C
C       ...FOR NORTHERN EDGE
C
      DO 170 IBOUND=NXNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+IBOUND
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+IBOUND-1
      IBNDG2(4,NBNDG2)=12
      IBNDG2(5,NBNDG2)=IBCN
      IBNDG2(6,NBNDG2)=0
```

```
C
      IBODY=(NXNODE-IBOUND)*NFACT+INBEG
      BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
      BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
      BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
      BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
      BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
      BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
      BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
      BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
      BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000C)
      ICELG2(10,IBNDG2(3,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL000C)
C
170   CONTINUE
C
C       ...FOR NORTHWESTERN CORNER
C
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(NYNODE-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(NYCELL-1)+1
      IBNDG2(3,NBNDG2)=0
      IBNDG2(4,NBNDG2)=13
      IBNDG2(5,NBNDG2)=IBCNW
      IBNDG2(6,NBNDG2)=0
C
      BONDG2(1,NBNDG2)=BODYG2( 8,INEND)
      BONDG2(2,NBNDG2)=BODYG2( 9,INEND)
      BONDG2(3,NBNDG2)=BODYG2(10,IWBEG)
      BONDG2(4,NBNDG2)=BODYG2(11,IWBEG)
      BONDG2(5,NBNDG2)=BODYG2( 3,IWBEG)
      BONDG2(6,NBNDG2)=BODYG2( 4,IWBEG)
      BONDG2(7,NBNDG2)=BODYG2( 5,IWBEG)
      BONDG2(8,NBNDG2)=BODYG2( 6,IWBEG)
      BONDG2(9,NBNDG2)=BODYG2( 7,IWBEG)
C
      ICELG2(10,IBNDG2(2,NBNDG2))
     &                  =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL000D)
C
C       ...FOR WESTERN EDGE
C
      DO 180 IBOUND=NYNODE-1,2,-1
      NBNDG2=NBNDG2+1
C
      IBNDG2(1,NBNDG2)=NXNODE*(IBOUND-1)+1
      IBNDG2(2,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-1)+1
      IBNDG2(3,NBNDG2)=ILVLG2(6,-1)+NXCELL*(IBOUND-2)+1
      IBNDG2(4,NBNDG2)=9
      IBNDG2(5,NBNDG2)=IBCW
```

```
        IBNDG2(6,NBNDG2)=0
C
        IBODY=(MYNODE-IBOUND)*NFACT+IWBEG
        BONDG2(1,NBNDG2)=BODYG2( 8,IBODY)
        BONDG2(2,NBNDG2)=BODYG2( 9,IBODY)
        BONDG2(3,NBNDG2)=BODYG2(10,IBODY)
        BONDG2(4,NBNDG2)=BODYG2(11,IBODY)
        BONDG2(5,NBNDG2)=BODYG2( 3,IBODY)
        BONDG2(6,NBNDG2)=BODYG2( 4,IBODY)
        BONDG2(7,NBNDG2)=BODYG2( 5,IBODY)
        BONDG2(8,NBNDG2)=BODYG2( 6,IBODY)
        BONDG2(9,NBNDG2)=BODYG2( 7,IBODY)
C
        ICELG2(10,IBNDG2(2,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(2,NBNDG2)),HL0009)
        ICELG2(10,IBNDG2(3,NBNDG2))
     &                    =IOR(ICELG2(10,IBNDG2(3,NBNDG2)),HL0009)
C
180     CONTINUE
C
C*****SORT THE CELLS
C
        CALL G2SORT
C
        RETURN
        END
```

G2INTG

```
        SUBROUTINE G2INTG(ISTART,
     &
     &                    XINT,YINT)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE COMPUTES THE TWO INTEGRALS:
C
C              INT(WORK) DX                  INT(WORK) DY
C        XINT= ------------            YINT= ------------
C                 INT DX                        INT DY
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]UTCOMN.INC'
C
C*********************************************************************
C
C     GET BASED VARIABLES
```

```
C
        CALL UTBASE(+1,1000,IXPLOT)
        CALL UTBASE(+1,1000,IYPLOT)
        CALL UTBASE(+1,1000,IFPLOT)
C
C     GET X AND Y FOR THIS LINE
C
        CALL G2LINE( 1,ISTART,VARUT(IXPLOT+1),NPLOT)
        CALL G2LINE( 2,ISTART,VARUT(IYPLOT+1),NPLOT)
C
        CALL G2LINE(99,ISTART,VARUT(IFPLOT+1),NPLOT)
C
C     COMPUTE THE INTEGRALS BY TRAPEZOIDAL INTEGRATION
C
        XINT=0.0
        YINT=0.0
C
        DO 10 I=2,NPLOT
        XINT=XINT+0.50*(VARUT(IFPLOT+I)+VARUT(IFPLOT+I-1))
     &               *(VARUT(IXPLOT+I)-VARUT(IXPLOT+I-1))
        YINT=YINT+0.50*(VARUT(IFPLOT+I)+VARUT(IFPLOT+I-1))
     &               *(VARUT(IYPLOT+I)-VARUT(IYPLOT+I-1))
10      CONTINUE
C
C     RELEASE BASED VARIABLES
C
        CALL UTBASE(-1,1000,IXPLOT)
        CALL UTBASE(-1,1000,IYPLOT)
        CALL UTBASE(-1,1000,IFPLOT)
C
        RETURN
        END
```

G2LAPL

```
        SUBROUTINE G2LAPL(NITER,IUNIT,IEVEN,
     &
     &                    IER             )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES A SMOOTH GRID BY THE SOLUTION OF
C        LAPLACE'S EQUATION.  THIS IS DONE THROUGH THE POINTER SYSTEM
C        SO THAT ANY TOPOLOGY MAY BE HANDLED
C
C     IF IEVEN = 0 USE EVENLY SPACED XI AND ETA AND ALL INTERIOR NODES
C                  CAN FLOAT
C              = 1 THEN DPENG2(1)      CONTAINS XI
C                       DPENG2(2)      CONTAINS ETA
C                       DPENG2(3).EQ.0 IMPLIES THAT NODE IS PINNED
```

```
C                       DPENG2(3).NE.0 IMPLIES THAT NODE FLOATS
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
        DIMENSION ACYC(4),BCYC(-1:0)
        DATA      ACYC   /1.0, 0.5, 1.0, 8.0/
        DATA      BCYC   /2.0, 8.0          /
C
C*************************************************************************
C
        IF(NITER.LE.0) RETURN
C
C     ASSIGN STORAGE FOR TEMPORARY VARIABLES
C
        IF((9*NNODG2).GT.(2*MNODG2)) THEN
           CALL UTEROR(+1,REAL(NNODG2),REAL(MNODG2))
        ENDIF
C
C     SET UP OFFSETS FOR TEMPORARY ARRAYS
C
        IXXI=0
        IXET=NNODG2
        IYXI=NNODG2*2
        IYET=NNODG2*3
        IXIP=NNODG2*4
        IXIM=NNODG2*5
        IETP=NNODG2*6
        IETM=NNODG2*7
        IHIT=NNODG2*8
C
        IC1 =0
        IC2 =NNODG2
        IC3 =NNODG2*2
        IC4 =NNODG2*3
        IC5 =NNODG2*4
        IDX =NNODG2*5
        IDY =NNODG2*6
C
C     LOOP THROUGH FOR EACH MULTIUPLE GRID LEVEL
C
C       DO 80 ILEVEL=-1,0
        ILEVEL=0
        IF(ILVLG2(1,ILEVEL).GT.ILVLG2(6,ILEVEL)) GOTO 80
C
C     LOOP THROUGH FOR EACH ITERATION
C
        DO 60 ITER=1,NITER
C
C     SET THE RELAXATION PARAMETER BASED UPON EMPIRICALLY DETERMINED
```

```
C        'BEST' RELAXATION PARAMETERS.  SPECIAL THANX TO MIKE
C
         RLX=MIN(ACYC(1+MOD(ITER,4)) , BCYC(ILEVEL))
C
C     PASS 0...ZERO OUT ACCUMULATIONS FOR XXI, XET, YXI, YET
C
         DO 10 INODE=1,NNODG2
         WORKG2(IXXI+INODE)=0.0
         WORKG2(IXET+INODE)=0.0
         WORKG2(IYXI+INODE)=0.0
         WORKG2(IYET+INODE)=0.0
         WORKG2(IXIP+INODE)=0.0
         WORKG2(IXIM+INODE)=0.0
         WORKG2(IETP+INODE)=0.0
         WORKG2(IETM+INODE)=0.0
         WORKG2(IHIT+INODE)=0.0
10       CONTINUE
C
C     PASS 1...ACCUMULATE XXI, XET, YXI, YET, XI+, XI-, ET+, ET-
C
         DO 20 ICELL=ILVLG2(1,ILEVEL),ILVLG2(6,ILEVEL)
C
         ISW=ICELG2(2,ICELL)
         ISE=ICELG2(4,ICELL)
         INE=ICELG2(6,ICELL)
         INW=ICELG2(8,ICELL)
C
         IAUX=ICELG2(10,ICELL)
C
         WORKG2(IXXI+ISW)=WORKG2(IXXI+ISW)+(GEOMG2(1,ISE)-GEOMG2(1,ISW))
         WORKG2(IYXI+ISW)=WORKG2(IYXI+ISW)+(GEOMG2(2,ISE)-GEOMG2(2,ISW))
         WORKG2(IXET+ISW)=WORKG2(IXET+ISW)+(GEOMG2(1,INW)-GEOMG2(1,ISW))
         WORKG2(IYET+ISW)=WORKG2(IYET+ISW)+(GEOMG2(2,INW)-GEOMG2(2,ISW))
C
         IF(IAND(IAUX,HL0001).NE.HL0001)
     &                           WORKG2(IHIT+ISW)=WORKG2(IHIT+ISW)+1.0
C
         IF(IEVEN.EQ.0) THEN
          WORKG2(IXIP+ISW)=WORKG2(IXIP+ISW)+1.0
          WORKG2(IETP+ISW)=WORKG2(IETP+ISW)+1.0
         ELSE
          WORKG2(IXIP+ISW)=WORKG2(IXIP+ISW)+(DPENG2(1,ISE)-DPENG2(1,ISW))
          WORKG2(IETP+ISW)=WORKG2(IETP+ISW)+(DPENG2(2,INW)-DPENG2(2,ISW))
         ENDIF
C
         WORKG2(IXXI+ISE)=WORKG2(IXXI+ISE)-(GEOMG2(1,ISW)-GEOMG2(1,ISE))
         WORKG2(IYXI+ISE)=WORKG2(IYXI+ISE)-(GEOMG2(2,ISW)-GEOMG2(2,ISE))
         WORKG2(IXET+ISE)=WORKG2(IXET+ISE)+(GEOMG2(1,INE)-GEOMG2(1,ISE))
         WORKG2(IYET+ISE)=WORKG2(IYET+ISE)+(GEOMG2(2,INE)-GEOMG2(2,ISE))
C
         IF(IAND(IAUX,HL0002).NE.HL0002)
     &                           WORKG2(IHIT+ISE)=WORKG2(IHIT+ISE)+1.0
C
```

```
      IF(IEVEN.EQ.0) THEN
       WORKG2(IXIM+ISE)=WORKG2(IXIM+ISE)+1.0
       WORKG2(IETP+ISE)=WORKG2(IETP+ISE)+1.0
      ELSE
       WORKG2(IXIM+ISE)=WORKG2(IXIM+ISE)-(DPENG2(1,ISW)-DPENG2(1,ISE))
       WORKG2(IETP+ISE)=WORKG2(IETP+ISE)+(DPENG2(2,INE)-DPENG2(2,ISE))
      ENDIF
C
      WORKG2(IXXI+INE)=WORKG2(IXXI+INE)-(GEONG2(1,INW)-GEONG2(1,INE))
      WORKG2(IYXI+INE)=WORKG2(IYXI+INE)-(GEONG2(2,INW)-GEONG2(2,INE))
      WORKG2(IXET+INE)=WORKG2(IXET+INE)-(GEONG2(1,ISE)-GEONG2(1,INE))
      WORKG2(IYET+INE)=WORKG2(IYET+INE)-(GEONG2(2,ISE)-GEONG2(2,INE))
C
      IF(IAND(IAUX,HLOOO4).NE.HLOOO4)
     &                          WORKG2(IHIT+INE)=WORKG2(IHIT+INE)+1.0
C
      IF(IEVEN.EQ.0) THEN
       WORKG2(IXIM+INE)=WORKG2(IXIM+INE)+1.0
       WORKG2(IETM+INE)=WORKG2(IETM+INE)+1.0
      ELSE
       WORKG2(IXIM+INE)=WORKG2(IXIM+INE)-(DPENG2(1,INW)-DPENG2(1,INE))
       WORKG2(IETM+INE)=WORKG2(IETM+INE)-(DPENG2(2,ISE)-DPENG2(2,INE))
      ENDIF
C
      WORKG2(IXXI+INW)=WORKG2(IXXI+INW)+(GEONG2(1,INE)-GEONG2(1,INW))
      WORKG2(IYXI+INW)=WORKG2(IYXI+INW)+(GEONG2(2,INE)-GEONG2(2,INW))
      WORKG2(IXET+INW)=WORKG2(IXET+INW)-(GEONG2(1,ISW)-GEONG2(1,INW))
      WORKG2(IYET+INW)=WORKG2(IYET+INW)-(GEONG2(2,ISW)-GEONG2(2,INW))
C
      IF(IAND(IAUX,HLOOO8).NE.HLOOO8)
     &                          WORKG2(IHIT+INW)=WORKG2(IHIT+INW)+1.0
C
      IF(IEVEN.EQ.0) THEN
       WORKG2(IXIP+INW)=WORKG2(IXIP+INW)+1.0
       WORKG2(IETM+INW)=WORKG2(IETM+INW)+1.0
      ELSE
       WORKG2(IXIP+INW)=WORKG2(IXIP+INW)+(DPENG2(1,INE)-DPENG2(1,INW))
       WORKG2(IETM+INW)=WORKG2(IETM+INW)-(DPENG2(2,ISW)-DPENG2(2,INW))
      ENDIF
C
20    CONTINUE
C
C     PASS 2...CALCULATE PDE COEFFICIENTS AT EACH POINT AND INITIALIZE
C              CHANGES IN LOCATION
C
      DO 30 INODE=1,NNODG2
      IF(WORKG2(IHIT+INODE).NE.4.0) GOTO 30
C
      DXIAV=0.50*(WORKG2(IXIM+INODE)+WORKG2(IXIP+INODE))
      DETAV=0.50*(WORKG2(IETM+INODE)+WORKG2(IETP+INODE))
C
      XXI=WORKG2(IXXI+INODE)/DXIAV
      YXI=WORKG2(IYXI+INODE)/DXIAV
```

```
      XET=WORKG2(IXET+INODE)/DETAV
      YET=WORKG2(IYET+INODE)/DETAV
C
      ALFA=XET*XET+YET*YET
      BETA=XXI*XET+YXI*YET
      GAMA=XXI*XXI+YXI*YXI
C
      CXIM=1.0/(WORKG2(IXIM+INODE)*DXIAV)
      CXIP=1.0/(WORKG2(IXIP+INODE)*DXIAV)
      CETM=1.0/(WORKG2(IETM+INODE)*DETAV)
      CETP=1.0/(WORKG2(IETP+INODE)*DETAV)
      CCCC=1.0/(DXIAV*DETAV)
C
      DENOM=2.0*(ALFA*(CXIM+CXIP)+GAMA*(CETM+CETP))
C
      WORKG2(IC1+INODE)=ALFA*CXIM/DENOM
      WORKG2(IC2+INODE)=ALFA*CXIP/DENOM
      WORKG2(IC3+INODE)=BETA*CCCC/DENOM
      WORKG2(IC4+INODE)=GAMA*CETM/DENOM
      WORKG2(IC5+INODE)=GAMA*CETP/DENOM
C
      WORKG2(IDX+INODE)=0.0
      WORKG2(IDY+INODE)=0.0
C
30    CONTINUE
C
C     PASS 3...ACCUMULATE CHANGE IN POSITION FOR EACH INTERIOR NODE
C
      DO 40 ICELL=ILVLG2(1,ILEVEL),ILVLG2(6,ILEVEL)
C
      ISW=ICELG2(2,ICELL)
      ISE=ICELG2(4,ICELL)
      INE=ICELG2(6,ICELL)
      INW=ICELG2(8,ICELL)
C
      WORKG2(IDX+ISW)=WORKG2(IDX+ISW)
     &                +WORKG2(IC2+ISW)*(GEOMG2(1,ISE)-GEOMG2(1,ISW))
     &                -WORKG2(IC3+ISW)*(GEOMG2(1,INE))
     &                +WORKG2(IC5+ISW)*(GEOMG2(1,INW)-GEOMG2(1,ISW))
      WORKG2(IDY+ISW)=WORKG2(IDY+ISW)
     &                +WORKG2(IC2+ISW)*(GEOMG2(2,ISE)-GEOMG2(2,ISW))
     &                -WORKG2(IC3+ISW)*(GEOMG2(2,INE))
     &                +WORKG2(IC5+ISW)*(GEOMG2(2,INW)-GEOMG2(2,ISW))
C
      WORKG2(IDX+ISE)=WORKG2(IDX+ISE)
     &                +WORKG2(IC1+ISE)*(GEOMG2(1,ISW)-GEOMG2(1,ISE))
     &                +WORKG2(IC3+ISE)*(GEOMG2(1,INW))
     &                +WORKG2(IC5+ISE)*(GEOMG2(1,INE)-GEOMG2(1,ISE))
      WORKG2(IDY+ISE)=WORKG2(IDY+ISE)
     &                +WORKG2(IC1+ISE)*(GEOMG2(2,ISW)-GEOMG2(2,ISE))
     &                +WORKG2(IC3+ISE)*(GEOMG2(2,INW))
     &                +WORKG2(IC5+ISE)*(GEOMG2(2,INE)-GEOMG2(2,ISE))
C
```

```
      WORKG2(IDX+INE)=WORKG2(IDX+INE)
     &                +WORKG2(IC1+INE)*(GEOMG2(1,INW)-GEOMG2(1,INE))
     &                -WORKG2(IC3+INE)*(GEOMG2(1,ISW))
     &                +WORKG2(IC4+INE)*(GEOMG2(1,ISE)-GEOMG2(1,INE))
      WORKG2(IDY+INE)=WORKG2(IDY+INE)
     &                +WORKG2(IC1+INE)*(GEOMG2(2,INW)-GEOMG2(2,INE))
     &                -WORKG2(IC3+INE)*(GEOMG2(2,ISW))
     &                +WORKG2(IC4+INE)*(GEOMG2(2,ISE)-GEOMG2(2,INE))
C
      WORKG2(IDX+INW)=WORKG2(IDX+INW)
     &                +WORKG2(IC2+INW)*(GEOMG2(1,INE)-GEOMG2(1,INW))
     &                +WORKG2(IC3+INW)*(GEOMG2(1,ISE))
     &                +WORKG2(IC4+INW)*(GEOMG2(1,ISW)-GEOMG2(1,INW))
      WORKG2(IDY+INW)=WORKG2(IDY+INW)
     &                +WORKG2(IC2+INW)*(GEOMG2(2,INE)-GEOMG2(2,INW))
     &                +WORKG2(IC3+INW)*(GEOMG2(2,ISE))
     &                +WORKG2(IC4+INW)*(GEOMG2(2,ISW)-GEOMG2(2,INW))
C
40    CONTINUE
C
C     PASS 4...ADD CHANGES TO ALL INTERIOR NODES
C
      XYMAX=0.0
C
      DO 50 INODE=1,NNODG2
      IF(WORKG2(IHIT+INODE).NE.4.0) GOTO 50
C
C     PIN NODE LOACTIONS IF DPENG2(3)=0 AND IEVEN IS SET
C
      IF(IEVEN.EQ.1 .AND. DPENG2(3,INODE).EQ.0.0) GOTO 50
C
      XYMAX=MAX(XYMAX,ABS(WORKG2(IDX+INODE)),ABS(WORKG2(IDY+INODE)))
C
      GEOMG2(1,INODE)=GEOMG2(1,INODE)+RLX*WORKG2(IDX+INODE)
      GEOMG2(2,INODE)=GEOMG2(2,INODE)+RLX*WORKG2(IDY+INODE)
C
50    CONTINUE
C
      IF(IUNIT.NE.0) WRITE(IUNIT,55) ITER,RLX,XYMAX
55    FORMAT(' ITER=',I5,'  RLX=',F5.2,'  XYMAX=',F8.5)
C
C     CHECK CONVERGENCE AND STOP IF THE FINEST GLOBAL LEVEL
C
      IF(XYMAX.LT.0.001 .AND. ILEVEL.EQ.0) THEN
         IER=0
         RETURN
C
C     OTHERWISE, IF CONVERGED, GO TO NEXT FINER GRID
C
      ELSEIF(XYMAX.LT.0.001) THEN
         GOTO 65
C
C     OTHERWISE, GO FOR THE NEXT ITERATION
```

```
C
        ENDIF
C
60      CONTINUE
C
C     DID NOT CONVERGE
C
        CALL UTEROR(-2,XYMAX,REAL(NITER))
        IER=1
        RETURN
C
C     INTERPOLATE TO THE NEXT FINER GRID
C
65      DO 70 ICELL=ILVLG2(1,ILEVEL),ILVLG2(6,ILEVEL)
C
        IC =ICELG2( 1,ICELL)
        ISW=ICELG2( 2,ICELL)
        IS =ICELG2( 3,ICELL)
        ISE=ICELG2( 4,ICELL)
        IE =ICELG2( 5,ICELL)
        INE=ICELG2( 6,ICELL)
        IN =ICELG2( 7,ICELL)
        INW=ICELG2( 8,ICELL)
        IW =ICELG2( 9,ICELL)
C
C     INTERPOLATE TO THE SIDE NODES
C
        IF((IEVEN.EQ.0 .AND. (WORKG2(IHIT+ISW).EQ.4.0 .OR.
     &                        WORKG2(IHIT+ISE).EQ.4.0     )) .OR.
     &     (IEVEN.EQ.1 .AND.  DPENG2(3,IS).NE.0.0          )     ) THEN
           GEOMG2(1,IS)=0.50*(GEOMG2(1,ISW)+GEOMG2(1,ISE))
           GEOMG2(2,IS)=0.50*(GEOMG2(2,ISW)+GEOMG2(2,ISE))
        ENDIF
C
        IF((IEVEN.EQ.0 .AND. (WORKG2(IHIT+ISE).EQ.4.0 .OR.
     &                        WORKG2(IHIT+INE).EQ.4.0     )) .OR.
     &     (IEVEN.EQ.1 .AND.  DPENG2(3,IE).NE.0.0          )     ) THEN
           GEOMG2(1,IE)=0.50*(GEOMG2(1,ISE)+GEOMG2(1,INE))
           GEOMG2(2,IE)=0.50*(GEOMG2(2,ISE)+GEOMG2(2,INE))
        ENDIF
C
        IF((IEVEN.EQ.0 .AND. (WORKG2(IHIT+INE).EQ.4.0 .OR.
     &                        WORKG2(IHIT+INW).EQ.4.0     )) .OR.
     &     (IEVEN.EQ.1 .AND.  DPENG2(3,IN).NE.0.0          )     ) THEN
           GEOMG2(1,IN)=0.50*(GEOMG2(1,INE)+GEOMG2(1,INW))
           GEOMG2(2,IN)=0.50*(GEOMG2(2,INE)+GEOMG2(2,INW))
        ENDIF
C
        IF((IEVEN.EQ.0 .AND. (WORKG2(IHIT+INW).EQ.4.0 .OR.
     &                        WORKG2(IHIT+ISW).EQ.4.0     )) .OR.
     &     (IEVEN.EQ.1 .AND.  DPENG2(3,IW).NE.0.0          )     ) THEN
           GEOMG2(1,IW)=0.50*(GEOMG2(1,INW)+GEOMG2(1,ISW))
           GEOMG2(2,IW)=0.50*(GEOMG2(2,INW)+GEOMG2(2,ISW))
```

```
        ENDIF
C
C     INTERPOLATE TO THE CENTER NODES
C
        IF(IEVEN.EQ.1 .AND. DPENG2(3,IC).NE.0.0) THEN
           GEOMG2(1,IC)=0.25*(GEOMG2(1,ISW)+GEOMG2(1,ISE)
     &                       +GEOMG2(1,INE)+GEOMG2(1,INW))
           GEOMG2(2,IC)=0.25*(GEOMG2(2,ISW)+GEOMG2(2,ISE)
     &                       +GEOMG2(2,INE)+GEOMG2(2,INW))
        ENDIF
C
70      CONTINUE
C
C     NEXT FINER GRID
C
80      CONTINUE
C
        END
```

G2LINE

```
        SUBROUTINE G2LINE(IDATA,ISTART,
     &
     &                    DATA$,NDATA )
C
        INCLUDE '[.UTILITY]PROLCG.INC'
C
C     THIS SUBROUTINE GENERATES A STRING OF DATA BY MARCHING
C        THROUGH THE FIELD EITHER TO THE NORTH OR TO THE EAST,
C        STARTING AT IABS(ISTART)
C
C        IDATA   = 0   DATA$ CONTAINS NODE NUMBERS
C                = 1                  GEOM1
C                = 2                  GEOM2
C                =11                  DPEN1
C                =12                  DPEN2
C                =13                  DPEN3
C                =14                  DPEN4
C                =15                  DPEN5
C                =99                  WORK
C
C        IF(ISTART.GT.0) MARCH TO THE EAST
C        IF(ISTART.LT.0) MARCH TO THE NORTH
C
        DIMENSION DATA$(*)
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
```

```
C*********************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
        IF(INBRG2.EQ.0) THEN
           CALL UTEROR(+1,REAL(INBRG2),0.0)
        ENDIF
C
C     SET UP FOR FIRST DATA POINT
C
        INODE=ABS(ISTART)
        JCELL=0
        NDATA=0
C
C     SAVE DATA FOR THIS POINT
C
10      NDATA=NDATA+1
C
        IF(IDATA.EQ. 0) DATA$(NDATA)=INODE
        IF(IDATA.EQ. 1) DATA$(NDATA)=GEOMG2(1,INODE)
        IF(IDATA.EQ. 2) DATA$(NDATA)=GEOMG2(2,INODE)
        IF(IDATA.EQ.11) DATA$(NDATA)=DPENG2(1,INODE)
        IF(IDATA.EQ.12) DATA$(NDATA)=DPENG2(2,INODE)
        IF(IDATA.EQ.13) DATA$(NDATA)=DPENG2(3,INODE)
        IF(IDATA.EQ.14) DATA$(NDATA)=DPENG2(4,INODE)
        IF(IDATA.EQ.15) DATA$(NDATA)=DPENG2(5,INODE)
        IF(IDATA.EQ.99) DATA$(NDATA)=WORKG2(  INODE)
C
C     CHECK FOR O-GRID CYCLING
C
        IF(INODE.EQ.ABS(ISTART) .AND. NDATA.GT.1) GOTO 40
C
        JNODE=INODE
C
C     IF NEIGHBOR CELL DOESN'T EXIST, THEN TRY FULL SEARCH
C
        IF(JCELL.EQ.0) GOTO 20
C
C     IF NEIGHBOR CELL DOES EXIST BUT THERE IS A NODE ON THE
C        APPROPRIATE FACE, THEN GOTO FULL SEARCH, OTHERWISE USE
C        THE NEIGHBOR INFORMATION
C
        IF(IMODE.EQ.1) THEN
           IF(ICELG2(3,JCELL).NE.0) GOTO 20
           INODE=ICELG2(4,JCELL)
           JCELL=NBORG2(5,JCELL)
           GOTO 10
        ELSEIF(IMODE.EQ.2) THEN
           IF(ICELG2(7,JCELL).NE.0) GOTO 20
           INODE=ICELG2(6,JCELL)
           JCELL=NBORG2(5,JCELL)
           GOTO 10
        ELSEIF(IMODE.EQ.3) THEN
```

```
            IF(ICELG2(9,JCELL).NE.0) GOTO 20
            INODE=ICELG2(8,JCELL)
            JCELL=NBORG2(7,JCELL)
            GOTO 10
         ELSEIF(INODE.EQ.4) THEN
            IF(ICELG2(5,JCELL).NE.0) GOTO 20
            INODE=ICELG2(6,JCELL)
            JCELL=NBORG2(7,JCELL)
            GOTO 10
         ENDIF
C
C     FULL SEARCH ... SCAN THROUGH EACH CANDIDATE CELL (LAST TO FIRST
C        TO GET FINEST)
C
20       INODE=0
C
         DO 30 ICELL=NCELG2,1,-1
         IF(ICELG2(1,ICELL).NE.0) GOTO 30
C
         IF(ISTART.GT.0) THEN
C
            IF(JNODE.EQ.ICELG2(2,ICELL)) THEN
               INODE=ICELG2(4,ICELL)
               JCELL=NBORG2(5,ICELL)
               IMODE=1
            ELSEIF(JNODE.EQ.ICELG2(8,ICELL)) THEN
               INODE=ICELG2(6,ICELL)
               JCELL=NBORG2(5,ICELL)
               IMODE=2
            ENDIF
C
         ELSE
C
            IF(JNODE.EQ.ICELG2(2,ICELL)) THEN
               INODE=ICELG2(8,ICELL)
               JCELL=NBORG2(7,ICELL)
               IMODE=3
            ELSEIF(JNODE.EQ.ICELG2(4,ICELL)) THEN
               INODE=ICELG2(6,ICELL)
               JCELL=NBORG2(7,ICELL)
               IMODE=4
            ENDIF
C
         ENDIF
C
         IF(IMODE.NE.0) GOTO 10
C
30       CONTINUE
C
C     ALL CONNECTED CELLS HAVE BEEN EXHAUSTED
C
40       RETURN
         END
```

G2NBOR

```
      SUBROUTINE G2NBOR(IUNIT
     &
     &                        )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS AND WRITES THE NEIGHBOR ARRAYS.
C       IF IUNIT.GT.0    WRITE TO  UNIT  IUNIT
C       IF IUNIT.LT.0    READ FROM UNIT -IUNIT
C
C*****************************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      LOGICAL OPENF
C
C*****************************************************************************
C
      JUNIT=ABS(IUNIT)
C
C     IF UNIT IS NOT OPEN, THEN CREATE A NEW FILE (UNFORMATTED)
C
      INQUIRE(UNIT=JUNIT,OPENED=OPENF)
      IF(.NOT.OPENF) OPEN(UNIT=JUNIT,STATUS='NEW',FORM='UNFORMATTED')
C
      IF(IUNIT.GT.0) THEN
C
C.....WRITE OUT NEIGHBOR TABLE
C
         REWIND JUNIT
C
         WRITE(JUNIT) NCELG2
         WRITE(JUNIT) ((NBORG2(K,ICELL),K=1,9),ICELL=1,NCELG2)
C
         INBRG2=0
      ELSE
C
C.....READ IN NEIGHBOR TABLE
C
         REWIND JUNIT
C
         READ(JUNIT,END=10) NCELL
         IF(NCELL.NE.NCELG2) CALL UTEROR(1,REAL(NCELL),REAL(NCELG2))
         READ(JUNIT,END=10) ((NBORG2(K,ICELL),K=1,9),ICELL=1,NCELG2)
C
         INBRG2=1
      ENDIF
C
      RETURN
```

```
C
C     END OF FILE DETECTED DURING READ
C
10      CALL UTEROR(2,REAL(NCELG2),0.)
        RETURN
C
        END
```

G2PLOT

```
        SUBROUTINE G2PLOT(ITYPE,IAUX,IOPTN
     &
     &                                  )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES EITHER A LINE, GRID, VECTOR, OR CONTOUR
C       PLOT OF THE DATA CONTAINED IN /G2COMN/
C
C       FOR ITYPE =1 GRID PLOT
C                 =2 VECTOR PLOT
C                 =3 LINE PLOT          (IAUX IS STARTING NODE)
C                 =4 CONTOUR PLOT       (IAUX IN NUMBER OF CONTOURS)
C                 =5 PERSPECTIVE PLOT   (IAUX IS UNIT NUMBER)
C                 =6 DATA-CONTOUR PLOT
C                 =7 SURFACE PLOTS      (IAUX IS BOUNDARY CONDITION
C                                                          POINTER)
C                 =8 DIMPLED CONTOUR    (IAUX IS NUMBER OF CONTOURS)
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]UTCOMN.INC'
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
        EXTERNAL G2PLTC,G2PLTD,G2PLTG,G2PLTV,G2PLTS
C
        DIMENSION GRDUMY(8),XSCALE(2),YSCALE(2),DATAIN(8)
C
        CHARACTER*80 PLTITL
        CHARACTER* 8 TITL,TITLIN(8)
C
C*************************************************************************
C
C     BRANCH BASED ON TYPE OF PLOT
C
        GOTO (100,200,300,400,500,600,700,800), ITYPE
                                                RETURN
```

```
C
C*****GRID PLOT
C
C     FIND MINIMUM AND MAXIMUM COORDINATE LOCATIONS (AND DEPENDENT
C        VARIABLES IF CONTOUR) FOR THIS PLOT
C
100     XMIN= 1.E+20
        XMAX=-1.E+20
        YMIN= 1.E+20
        YMAX=-1.E+20
C
C      LOOP THROUGH ALL NODES
C
        DO 110 INODE=1,NNODG2
C
        XNODE=GEONG2(1    ,INODE)
        YNODE=GEONG2(2    ,INODE)
C
        IF(XNODE.LT.XMIN) XMIN=XNODE
        IF(XNODE.GT.XMAX) XMAX=XNODE
        IF(YNODE.LT.YMIN) YMIN=YNODE
        IF(YNODE.GT.YMAX) YMAX=YNODE
C
110     CONTINUE
C
C     STORE THESE VALUES FOR SCALING BY GRAPHC
C
        XSCALE(1)=XMIN
        XSCALE(2)=XMAX
        YSCALE(1)=YMIN
        YSCALE(2)=YMAX
C
C     SET UP GRAPHICS INDICATOR, TITLE, AND THEN PLOT
C
        INDGR=23+IOPTN
C
        WRITE(PLTITL,120) NNODG2
120     FORMAT('  GEOM1   GEOM2 COMPUTATIONAL GRID',27X,'NODES=',I6)
C
        CALL GRAPHC(G2PLTG,INDGR,PLTITL,XSCALE,YSCALE,2,GRDUMY)
C
        RETURN
C
C*****VECTOR PLOT
C
C     FIND MINIMUM AND MAXIMUM COORDINATE LOCATIONS (AND DEPENDENT
C        VARIABLES IF CONTOUR) FOR THIS PLOT
C
200     XMIN= 1.E+20
        XMAX=-1.E+20
        YMIN= 1.E+20
        YMAX=-1.E+20
C
```

```
C      LOOP THROUGH ALL NODES
C
       DO 210 INODE=1,NNODG2
C
       XNODE=GEOMG2(1    ,INODE)
       YNODE=GEOMG2(2    ,INODE)
C
       IF(XNODE.LT.XMIN) XMIN=XNODE
       IF(XNODE.GT.XMAX) XMAX=XNODE
       IF(YNODE.LT.YMIN) YMIN=YNODE
       IF(YNODE.GT.YMAX) YMAX=YNODE
C
210    CONTINUE
C
C     STORE THESE VALUES FOR SCALING BY GRAPHC
C
       XSCALE(1)=XMIN
       XSCALE(2)=XMAX
       YSCALE(1)=YMIN
       YSCALE(2)=YMAX
C
C     SET UP GRAPHICS INDICATOR, TITLE, AND THEN PLOT
C
       INDGR=19+IOPTN
C
       WRITE(PLTITL,220) NNODG2
220    FORMAT('  GEOM1   GEOM2 VELOCITY VECTORS  ',27X,'NODES=',I6)
C
       CALL GRAPHC(G2PLTV,INDGR,PLTITL,XSCALE,YSCALE,2,GRDUMY)
C
       RETURN
C
C*****LINE PLOT
C
C     GET BASED VARIABLES
C
300    CALL UTBASE(+1,1000,IXPLOT)
       CALL UTBASE(+1,1000,IYPLOT)
C
       IF(IAUX.NE.0) THEN
          ISTART=IAUX
       ELSE
          ISTART=1
       ENDIF
C
C     GET X FOR THIS LINE
C
       CALL G2LINE(1,ISTART,VARUT(IXPLOT+1),NPLOT)
C
C     IF PLOT IS VERSUS ARC-LENGTH, GET Y AND CALCULATE ARC-LENGTH
C
       IF(IAUX.EQ.0) GOTO 315
C
```

```
        CALL G2LINE(2,ISTART,VARUT(IYPLOT+1),NPLOT)
C
C     CALCULATE ARC-LENGTH ALONG THIS LINE
C
        XOLD=VARUT(IXPLOT+1)
        YOLD=VARUT(IYPLOT+1)
        SOLD=0.0
C
        VARUT(IXPLOT+1)=SOLD
C
        DO 310 I=2,NPLOT
C
        XNEW=VARUT(IXPLOT+I)
        YNEW=VARUT(IYPLOT+I)
C
        SNEW=SOLD+SQRT((XOLD-XNEW)**2+(YOLD-YNEW)**2)
C
        XOLD=XNEW
        YOLD=YNEW
        SOLD=SNEW
C
        VARUT(IXPLOT+I)=SNEW
C
310     CONTINUE
C
C     GET DATA FOR THIS PLOT
C
315     CALL G2LINE(99,ISTART,VARUT(IYPLOT+1),NPLOT)
C
C     SET UP GRAPHICS INDICATOR, TITLE, AND THEN PLOT
C
        INDGR=29+IOPTN
C
        IF(IAUX.GT.0) WRITE(PLTITL, 320) TITLG2,ABS(IAUX),NNODG2
320     FORMAT(' ARCLEN ',A8,' WEST->EAST TRAVERSE STARTING AT',I4,
     &                                                  5X,'NODES=',I6)
        IF(IAUX.LT.0) WRITE(PLTITL, 330) TITLG2,ABS(IAUX),NNODG2
330     FORMAT(' ARCLEN ',A8,' SOUTH->NORTH TRAVERSE STARTING AT',I4,
     &                                                  5X,'NODES=',I6)
        IF(IAUX.EQ.0) WRITE(PLTITL, 340) TITLG2,NNODG2
340     FORMAT('    X   ',A8,' TRAVERSE STARTING AT NODE 1',
     &                                                  5X,'NODES=',I6)
C
        CALL GRLINE(1,1,1,PLTITL,INDGR,
     &                          VARUT(IXPLOT+1),VARUT(IYPLOT+1),NPLOT)
C
C     RELEASE BASED VARIABLES
C
        CALL UTBASE(-1,1000,IXPLOT)
        CALL UTBASE(-1,1000,IYPLOT)
C
        RETURN
C
```

```
C*****CONTOUR PLOT
C
C     FIND MINIMUM AND MAXIMUM COORDINATE LOCATIONS (AND DEPENDENT
C        VARIABLES IF CONTOUR) FOR THIS PLOT
C
400      XMIN= 1.E+20
         XMAX=-1.E+20
         YMIN= 1.E+20
         YMAX=-1.E+20
         FMIN= 1.E+20
         FMAX=-1.E+20
C
C      LOOP THROUGH ALL NODES
C
         DO 410 INODE=1,NNODG2
C
         XNODE=GEOMG2(1     ,INODE)
         YNODE=GEOMG2(2     ,INODE)
         FNODE=WORKG2(      INODE)
C
         IF(XNODE.LT.XMIN) XMIN=XNODE
         IF(XNODE.GT.XMAX) XMAX=XNODE
         IF(YNODE.LT.YMIN) YMIN=YNODE
         IF(YNODE.GT.YMAX) YMAX=YNODE
         IF(FNODE.LT.FMIN) FMIN=FNODE
         IF(FNODE.GT.FMAX) FMAX=FNODE
C
410      CONTINUE
C
C     STORE THESE VALUES FOR SCALING BY GRAPHC
C
         XSCALE(1)=XMIN
         XSCALE(2)=XMAX
         YSCALE(1)=YMIN
         YSCALE(2)=YMAX
C
C     SET UP CONTOUR LEVELS
C
         IF(IAUX.GT.0) THEN
            NCONT=IAUX
            CALL GRSCAL(FMIN,FMAX,REAL(NCONT-1),CBASE,CSTEP)
         ELSE
            CALL UTINPI('NCONT',NCONT)
            CALL UTINPF('CBASE',CBASE)
            CALL UTINPF('CSTEP',CSTEP)
         ENDIF
C
         GRDUMY(1)=NCONT
         GRDUMY(2)=CBASE
         GRDUMY(3)=CSTEP
         GRDUMY(4)=0.0
C
C     SET UP GRAPHICS INDICATOR, TITLE, AND THEN PLOT
```

```
C
      INDGR=23+IOPTN
C
      WRITE(PLTITL,420) TITLG2,CSTEP,NNODG2
420   FORMAT('  GEOM1   GEOM2 CONTOURS OF ',A8,'  INCREMENT=',F8.4,
     &                                          5X,'NODES=',I6)
C
      CALL GRAPHC(G2PLTC,INDGR,PLTITL,XSCALE,YSCALE,2,GRDUMY)
C
      RETURN
C
C*****PERSPECTIVE PLOT
C
C     READ THE FILE HEADER
C
500   REWIND IAUX
      READ(IAUX,510) NPERLN,PLTITL
510   FORMAT(I5,A)
C
C     GET BASED VARIABLES
C
      CALL UTBASE(+1,NPERLN     ,IXPLOT)
      CALL UTBASE(+1,NPERLN*6000,IYPLOT)
C
C     READ THE ABSCISSA
C
      READ(IAUX,520) (VARUT(IXPLOT+KK),KK=1,NPERLN)
520   FORMAT(13F10.5)
C
C     READ THE DATA FILE
C
      NLINES=1
530   IOFF=IYPLOT+(NLINES-1)*NPERLN
      READ(IAUX,520,END=540) (VARUT(IOFF+KK),KK=1,NPERLN)
      NLINES=NLINES+1
      GOTO 530
C
540   NLINES=NLINES-1
C
C     SET UP OTHER PLOTTING CONTROLS AND THEN PLOT
C
      INDGR=21+IOPTN
C
      GRDUMY(1)=0.0
C
      CALL GRPERS(VARUT(IXPLOT+1),VARUT(IYPLOT+1),NPERLN,
     &                                 NLINES,GRDUMY,PLTITL,INDGR)
C
C     RELEASE BASED VARIABLES
C
      CALL UTBASE(-1,NPERLN     ,IXPLOT)
      CALL UTBASE(-1,NPERLN*6000,IYPLOT)
C
```

```
      RETURN
C
C*****DATA-CONTOUR PLOT
C
C     FIND MINIMUM AND MAXIMUM COORDINATE LOCATIONS (AND DEPENDENT
C        VARIABLES IF CONTOUR) FOR THIS PLOT
C
600     XMIN= 1.E+20
        XMAX=-1.E+20
        YMIN= 1.E+20
        YMAX=-1.E+20
        FMIN= 1.E+20
        FMAX=-1.E+20
C
C      LOOP THROUGH ALL NODES
C
        DO 610 INODE=1,NNODG2
C
        XNODE=GEOMG2(1    ,INODE)
        YNODE=GEOMG2(2    ,INODE)
        FNODE=WORKG2(      INODE)
C
        IF(XNODE.LT.XMIN) XMIN=XNODE
        IF(XNODE.GT.XMAX) XMAX=XNODE
        IF(YNODE.LT.YMIN) YMIN=YNODE
        IF(YNODE.GT.YMAX) YMAX=YNODE
        IF(FNODE.LT.FMIN) FMIN=FNODE
        IF(FNODE.GT.FMAX) FMAX=FNODE
C
610     CONTINUE
C
C     STORE THESE VALUES FOR SCALING BY GRAPHC
C
        XSCALE(1)=XMIN
        XSCALE(2)=XMAX
        YSCALE(1)=YMIN
        YSCALE(2)=YMAX
C
C     SET UP CONTOUR LEVELS
C
        NCONT=14
C
C     FIND CONTOUR LEVELS
C
        CALL GRSCAL(FMIN,FMAX,REAL(NCONT-1),CBASE,CSTEP)
C
        GRDUMY(1)=NCONT
        GRDUMY(2)=CBASE
        GRDUMY(3)=CSTEP
C
C     SET UP GRAPHICS INDICATOR, TITLE, AND THEN PLOT
C
        INDGR=23+IOPTN
```

```
C
        WRITE(PLTITL,620) TITLG2,WNCDG2
620     FORMAT('  GEOM1   GEOM2 DISCRETE CONTOURS OF ',A8,
     &                                                  5X,'NODES=',I6)
C
        CALL GRAPHC(G2PLTD,INDGR,PLTITL,XSCALE,YSCALE,2,GRDUMY)
C
        RETURN
C
C*****SURFACE PLOT
C
C     GET BASED VARIABLES
C
700     CALL UTBASE(+1,1000,IXPLOT)
        CALL UTBASE(+1,1000,IYPLOT)
C
C     FIND MINIMUM AND MAXIMUM DATA VALUES TO PLOT
C
        XMIN=+1.0E+20
        XMAX=-1.0E+20
        FMIN=+1.0E+20
        FMAX=-1.0E+20
C
        JAUX=ABS(IAUX)
C
        DO 710 IBOUND=1,NBNDG2
        IF(IBNDG2(5,IBOUND).NE.JAUX) GOTO 710
C
        INODE=IBNDG2(1,IBOUND)
        XMIN=MIN(XMIN,GEOMG2(1,INODE))
        XMAX=MAX(XMAX,GEOMG2(1,INODE))
        FMIN=MIN(FMIN,WORKG2(  INODE))
        FMAX=MAX(FMAX,WORKG2(  INODE))
C
710     CONTINUE
C
C     STORE THESE VALUES FOR SCALING BY GRAPHC
C
        VARUT(IXPLOT+1)=XMIN
        VARUT(IXPLOT+2)=XMAX
        VARUT(IYPLOT+1)=FMIN
        VARUT(IYPLOT+2)=FMAX
        NPLOT=2
C
C     IF IAUX.LT.0 THEN LOOK FOR COMPARISON DATA
C
        IF(IAUX.GE.0) GOTO 750
C
C     READ DATA FILE HEADER IF IT EXISTS TO FIND OUT WHAT TYPES
C        OF DATA THE FILE CONTAINS
C
        REWIND JCMPAR
        READ(JCMPAR,720,END=750) TITLIN
```

```
720     FORMAT(/8(A8,2X))
C
C     SEE IF A DATA TYPE MATCH THAT BEING PLOTTED
C
        DO 730 ICOL=1,8
        IF(TITLIN(ICOL).EQ.TITLG2) THEN
           JCOL=ICOL
           GOTO 740
        ENDIF
730     CONTINUE
C
        GOTO 750
C
C     READ THE FILE, PICKING UP THE APPROPRIATE DATA
C
740     READ(JCMPAR,745,END=750) DATAIN
745     FORMAT(8F10.0)
C
C     PUT THIS DATA IN THE PLOTTING ARRAY
C
        NPLOT=NPLOT+1
        VARUT(IXPLOT+NPLOT)=DATAIN(   1)
        VARUT(IYPLOT+NPLOT)=DATAIN(JCOL)
        IF(NPLOT.LT.1000) GOTO 740
C
C     SET UP GRAPHICS INDICATOR, TITLE, AND THEN PLOT
C
750     INDGR=29+IOPTN
C
        WRITE(PLTITL,760) TITLG2,NNODG2
760     FORMAT('    X   ',A8,'PLOT ALONG SOLID SURFACE(S)',
     &                                                  5X,'NODES=',I6)
C
        GRDUMY(1)=JAUX
C
        CALL GRAPHC(G2PLTS,INDGR,PLTITL,
     &                  VARUT(IXPLOT+1),VARUT(IYPLOT+1),NPLOT,GRDUMY)
C
C     RELEASE BASED VARIABLES
C
        CALL UTBASE(-1,1000,IXPLOT)
        CALL UTBASE(-1,1000,IYPLOT)
C
        RETURN
C
C*****CONTOUR PLOT
C
C     FIND MINIMUM AND MAXIMUM COORDINATE LOCATIONS (AND DEPENDENT
C        VARIABLES IF CONTOUR) FOR THIS PLOT
C
800     XMIN= 1.E+20
        XMAX=-1.E+20
        YMIN= 1.E+20
```

```
        YMAX=-1.E+20
        FMIN= 1.E+20
        FMAX=-1.E+20
C
C      LOOP THROUGH ALL NODES
C
        DO 810 INODE=1,NNODG2
C
        XNODE=GEOMG2(1    ,INODE)
        YNODE=GEOMG2(2    ,INODE)
        FNODE=WORKG2(      INODE)
C
        IF(XNODE.LT.XMIN) XMIN=XNODE
        IF(XNODE.GT.XMAX) XMAX=XNODE
        IF(YNODE.LT.YMIN) YMIN=YNODE
        IF(YNODE.GT.YMAX) YMAX=YNODE
        IF(FNODE.LT.FMIN) FMIN=FNODE
        IF(FNODE.GT.FMAX) FMAX=FNODE
C
810     CONTINUE
C
C     STORE THESE VALUES FOR SCALING BY GRAPHC
C
        XSCALE(1)=XMIN
        XSCALE(2)=XMAX
        YSCALE(1)=YMIN
        YSCALE(2)=YMAX
C
C     SET UP CONTOUR LEVELS
C
        IF(IAUX.GT.0) THEN
           NCONT=IAUX
           CALL GRSCAL(FMIN,FMAX,REAL(NCONT-1),CBASE,CSTEP)
        ELSE
           CALL UTINPI('NCONT',NCONT)
           CALL UTINPF('CBASE',CBASE)
           CALL UTINPF('CSTEP',CSTEP)
        ENDIF
C
        GRDUMY(1)=NCONT
        GRDUMY(2)=CBASE
        GRDUMY(3)=CSTEP
        GRDUMY(4)=1.0
C
C     GET THE DIMPLING LEVEL
C
        CALL UTINPF('DIMPLE LEVEL',GRDUMY(5))
C
C     SET UP GRAPHICS INDICATOR, TITLE, AND THEN PLOT
C
        INDGR=23+IOPTN
C
        WRITE(PLTITL,820) TITLG2,CSTEP,NNODG2
```

```
820     FORMAT('  GEOM1   GEOM2 CONTOURS OF ',A8,'  INCREMENT=',F8.4,
     &                                          5X,'NODES=',I6)
C
        CALL GRAPHC(G2PLTC,INDGR,PLTITL,XSCALE,YSCALE,2,GRDUMY)
C
        RETURN
C
        END
```

G2PLTC

```
        SUBROUTINE G2PLTC(XDUM$,YDUM$,NDUM,GRDUMY
     &
     &                                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES A CONTOUR PLOT FOR THE GRID
C       CONTAINED IN /G2COMN/
C
        DIMENSION XDUM$(*),YDUM$(*),GRDUMY(*)
C
C*********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
        DIMENSION INODE(9)
C
C*********************************************************************
C
        NCNT=NINT(GRDUMY(1))
C
C     LOOP THROUGH EACH CANDIDATE CELL
C
        DO 80 ICELL=1,NCELG2
        IF(ICELG2(1,ICELL).NE.0) GOTO 80
C
C     MAKE SURE CELL IS NOT OUTSIDE CLIPPING BOUNDARIES
C
        XMIN=MIN(GEOMG2(1,ICELG2(2,ICELL)),
     &           GEOMG2(1,ICELG2(4,ICELL)),
     &           GEOMG2(1,ICELG2(6,ICELL)),
     &           GEOMG2(1,ICELG2(8,ICELL)))
        IF(XMIN.GT.XMXCGR) GOTO 80
C
        XMAX=MAX(GEOMG2(1,ICELG2(2,ICELL)),
     &           GEOMG2(1,ICELG2(4,ICELL)),
     &           GEOMG2(1,ICELG2(6,ICELL)),
```

```
     &           GEOMG2(1,ICELG2(8,ICELL)))
         IF(XMAX.LT.XMNCGR) GOTO 80
C
         YMIN=MIN(GEOMG2(2,ICELG2(2,ICELL)),
     &           GEOMG2(2,ICELG2(4,ICELL)),
     &           GEOMG2(2,ICELG2(6,ICELL)),
     &           GEOMG2(2,ICELG2(8,ICELL)))
         IF(YMIN.GT.YMXCGR) GOTO 80
C
         YMAX=MAX(GEOMG2(2,ICELG2(2,ICELL)),
     &           GEOMG2(2,ICELG2(4,ICELL)),
     &           GEOMG2(2,ICELG2(6,ICELL)),
     &           GEOMG2(2,ICELG2(8,ICELL)))
         IF(YMAX.LT.YMNCGR) GOTO 80
C
C     STORE NODES FOR THIS CELL
C
         NFACES=1
         INODE(NFACES)=ICELG2(2,ICELL)
C
         IF(ICELG2(9,ICELL).NE.0) THEN
            NFACES=NFACES+1
            INODE(NFACES)=ICELG2(9,ICELL)
         ENDIF
C
         NFACES=NFACES+1
         INODE(NFACES)=ICELG2(8,ICELL)
C
         IF(ICELG2(7,ICELL).NE.0) THEN
            NFACES=NFACES+1
            INODE(NFACES)=ICELG2(7,ICELL)
         ENDIF
C
         NFACES=NFACES+1
         INODE(NFACES)=ICELG2(6,ICELL)
C
         IF(ICELG2(5,ICELL).NE.0) THEN
            NFACES=NFACES+1
            INODE(NFACES)=ICELG2(5,ICELL)
         ENDIF
C
         NFACES=NFACES+1
         INODE(NFACES)=ICELG2(4,ICELL)
C
         IF(ICELG2(3,ICELL).NE.0) THEN
            NFACES=NFACES+1
            INODE(NFACES)=ICELG2(3,ICELL)
         ENDIF
C
C     FIND LARGEST AND SMALLEST DEPENDENT VARIABLE VALUES FOR THIS CELL
C
         UMIN=WORKG2(INODE(1))
         UMAX=WORKG2(INODE(1))
```

```
C
        DO 10 IFACE=2,NFACES
        UMIN=MIN(UMIN,WORKG2(INODE(IFACE)))
        UMAX=MAX(UMAX,WORKG2(INODE(IFACE)))
10      CONTINUE
C
C     DETERMINE IF THIS CELL IS CROSSED BY ANY CONTOUR...IF NOT, GO TO
C       NEXT CELL
C
        DO 20 ICNT=1,NCNT
        CNT=GRDUMY(2)+(ICNT-1)*GRDUMY(3)
        IF(CNT.GE.UMIN .AND. CNT.LE.UMAX) GOTO 30
20      CONTINUE
C
        GOTO 80
C
C     FIND CENTROID
C
30      XMOMNT=0.
        YMOMNT=0.
        TAREA =0.
C
        DO 40 ITRI=3,NFACES
C
        INODEA=INODE(     1)
        INODEB=INODE(ITRI-1)
        INODEC=INODE(ITRI  )
C
        XCENT=(GEOMG2(1,INODEA)+GEOMG2(1,INODEB)+GEOMG2(1,INODEC))/3.0
        YCENT=(GEOMG2(2,INODEA)+GEOMG2(2,INODEB)+GEOMG2(2,INODEC))/3.0
C
        AREA =(GEOMG2(2,INODEB)-GEOMG2(2,INODEA))
     &       *(GEOMG2(1,INODEC)-GEOMG2(1,INODEA))
     &       -(GEOMG2(2,INODEC)-GEOMG2(2,INODEA))
     &       *(GEOMG2(1,INODEB)-GEOMG2(1,INODEA))
C
        XMOMNT=XMOMNT+XCENT*AREA
        YMOMNT=YMOMNT+YCENT*AREA
        TAREA =TAREA +      AREA
C
40      CONTINUE
C
        XCENT=XMOMNT/TAREA
        YCENT=YMOMNT/TAREA
C
        INODE(NFACES+1)=INODE(1)
C
C     COMPUTE THE CENTROID VALUES BASED UPON AREA WEIGHTING
C       OF SUB-TRIANGLES
C
        UCENT=0.
C
        DO 50 JTRI=1,NFACES
```

```
        JLEFT=INODE(JTRI  )
        JRITE=INODE(JTRI+1)
C
        AREA=(GEOMG2(2,JLEFT)-YCENT)*(GEOMG2(1,JRITE)-XCENT)
     &      -(GEOMG2(2,JRITE)-YCENT)*(GEOMG2(1,JLEFT)-XCENT)
        FACT=0.5*AREA/TAREA
C
        UCENT=UCENT+FACT*(WORKG2(JLEFT)+WORKG2(JRITE))
C
50      CONTINUE
C
C     STEP THROUGH EACH SUB-TRIANGLE, DRAWING THE APPROPRIATE CONTOURS
C
        DO 70 ITRI=1,NFACES
C
        XLEFT=GEOMG2(1,INODE(ITRI  ))
        YLEFT=GEOMG2(2,INODE(ITRI  ))
        ULEFT=WORKG2(  INODE(ITRI  ))
C
        XRITE=GEOMG2(1,INODE(ITRI+1))
        YRITE=GEOMG2(2,INODE(ITRI+1))
        URITE=WORKG2(  INODE(ITRI+1))
C
        UMIN=MIN(UCENT,ULEFT,URITE)
        UMAX=MAX(UCENT,ULEFT,URITE)
C
C     STEP THROUGH EACH CONTOUR
C
        DO 60 ICNT=1,NCNT
        CNT=GRDUMY(2)+(ICNT-1)*GRDUMY(3)
C
        IF(CNT.LT.UMIN .OR. CNT.GT.UMAX) GOTO 60
C
        CALL GRCTRI(XLEFT,YLEFT,ULEFT,
     &              XRITE,YRITE,URITE,
     &              XCENT,YCENT,UCENT,CNT)
C
60      CONTINUE
C
C     GO ON TO NEXT SUB-TRIANGLE
C
70      CONTINUE
C
C     NEXT CELL
C
80      CONTINUE
C
C*****DRAW THE DOMAIN BOUNDARIES
C
        DO 160 IBOUND=1,NBNDG2
C
        KNODE=IBNDG2(1,IBOUND)
        ILEFT=IBNDG2(2,IBOUND)
```

```
        IEDGE=IBNDG2(4,IBOUND)
        ITYPE=IBNDG2(5,IBOUND)
C
        IF(ITYPE.EQ.0) GOTO 160
C
C     BRANCH DEPENDING ON WHICH EDGE THE BOUNDARY CONDITION REFERS TO
C
        GOTO (160,110,120,110,130,130,120,110,
     &        140,140,110,140,130,130,120,110), (IEDGE+1)
C
C     ...SOUTH SIDE, SOUTHEAST CORNER (OUTSIDE), SOUTHWEST
C        CORNER (INSIDE), OR TRAILING EDGE SLIT
C
110     JNODE=ICELG2(2,ILEFT)
        GOTO 150
C
C     ...EAST SIDE, NORTHEAST CORNER (OUTSIDE), OR SOUTHEAST
C        CORNER (INSIDE)
C
120     JNODE=ICELG2(4,ILEFT)
        GOTO 150
C
C     ...NORTH SIDE, NORTHWEST CORNER (OUTSIDE), NORTHEAST
C        CORNER (INSIDE), OR LEADING EDGE SLIT
C
130     JNODE=ICELG2(6,ILEFT)
        GOTO 150
C
C     ...WEST SIDE, SOUTHWEST CORNER (OUTSIDE), OR NORTHWEST
C        CORNER (INSIDE)
C
140     JNODE=ICELG2(8,ILEFT)
C
C     DRAW THIS SEGMENT OF THE BOUNDARY
C
150     CALL GRMOVE(GEOMG2(1,KNODE),GEOMG2(2,KNODE),0)
        CALL GRDRAW(GEOMG2(1,JNODE),GEOMG2(2,JNODE),0)
C
160     CONTINUE
C
C     DIMPLE IF REQUIRED
C
        IF(NINT(GRDUMY(4)).NE.1) GOTO 180
C
        VDIMP=GRDUMY(5)
C
        DO 170 JNODE=1,NNODG2
        IF(WORKG2(JNODE).GE.VDIMP) THEN
           CALL GRMOVE(GEOMG2(1,JNODE),GEOMG2(2,JNODE),0)
           CALL GRDRAW(GEOMG2(1,JNODE),GEOMG2(2,JNODE),0)
        ENDIF
170     CONTINUE
C
```

```
180     RETURN
        END
```

G2PLTD

```
        SUBROUTINE G2PLTD(XDUM$,YDUM$,NDUM,GRDUMY
     &
     &                                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES A DATA-CONTOUR PLOT FOR THE GRID
C       CONTAINED IN /G2COMN/
C
        DIMENSION XDUM$(*),YDUM$(*),GRDUMY(*)
C
C*****************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************
C
        NSYM=NINT(GRDUMY(1))
        FMIN=     GRDUMY(2)
        FINC=     GRDUMY(3)
C
C     SET UP AN AREA FOR THE KEY
C
        CALL GRKEY(NSYM,FMIN,FINC,1)
C
C     LOOP THROUGH EACH NODE
C
        DO 10 INODE=1,NNODG2
C
        ISYM=1+(WORKG2(INODE)-FMIN)/FINC
        IF(ISYM.LT.   1) ISYM=1
        IF(ISYM.GT.NSYM) ISYM=NSYM
C
        CALL GRMOVE(GEOMG2(1,INODE),GEOMG2(2,INODE),ISYM)
C
10      CONTINUE
C
C     DRAW THE DOMAIN BOUNDARIES
C
        DO 70 IBOUND=1,NBNDG2
C
        KNODE=IBNDG2(1,IBOUND)
        ILEFT=IBNDG2(2,IBOUND)
```

```
        IEDGE=IBNDG2(4,IBOUND)
        ITYPE=IBNDG2(5,IBOUND)
C
        IF(ITYPE.EQ.0) GOTO 70
C
C     BRANCH DEPENDING ON WHICH EDGE THE BOUNDARY CONDITION REFERS TO
C
        GOTO (70,20,30,20,40,40,30,20,
     &        50,50,20,50,40,40,30,20), (IEDGE+1)
C
C     ...SOUTH SIDE, SOUTHEAST CORNER (OUTSIDE), SOUTHWEST
C        CORNER (INSIDE), OR TRAILING EDGE SLIT
C
20      JNODE=ICELG2(2,ILEFT)
        GOTO 60
C
C     ...EAST SIDE, NORTHEAST CORNER (OUTSIDE), OR SOUTHEAST
C        CORNER (INSIDE)
C
30      JNODE=ICELG2(4,ILEFT)
        GOTO 60
C
C     ...NORTH SIDE, NORTHWEST CORNER (OUTSIDE), NORTHEAST
C        CORNER (INSIDE), OR LEADING EDGE SLIT
C
40      JNODE=ICELG2(6,ILEFT)
        GOTO 60
C
C     ...WEST SIDE, SOUTHWEST CORNER (OUTSIDE), OR NORTHWEST
C        CORNER (INSIDE)
C
50      JNODE=ICELG2(8,ILEFT)
C
C     DRAW THIS SEGMENT OF THE BOUNDARY
C
60      CALL GRMOVE(GEOMG2(1,KNODE),GEOMG2(2,KNODE),0)
        CALL GRDRAW(GEOMG2(1,JNODE),GEOMG2(2,JNODE),0)
C
70      CONTINUE
C
C     RESET CLIPPING PARAMETER AND WRITE KEY
C
        CALL GRKEY(NSYM,FMIN,FINC,2)
C
        RETURN
        END
```

G2PLTG

```
        SUBROUTINE G2PLTG(XDUM$,YDUM$,NDUM,GRDUMY
```

```
     &
     &                                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES A GRID PLOT FOR THE GRID
C        CONTAINED IN /G2COMN/
C
        DIMENSION XDUM$(*),YDUM$(*),GRDUMY(*)
C
C*************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*************************************************************************
C
C
C     LOOP THROUGH EACH CANDIDATE CELL
C
        DO 20 ICELL=1,NCELG2
        IF(ICELG2(1,ICELL).NE.0) GOTO 20
C
C     MOVE TO THE NORTHWEST CORNER OF THIS CELL
C
        INODE=ICELG2(8,ICELL)
        XNODE=GEOMG2(1,INODE)
        YNODE=GEOMG2(2,INODE)
        CALL GRMOVE(XNODE,YNODE,0)
C
C     DRAW TO ALL FOUR CORNERS OF THIS CELL
C
        DO 10 ICORN=2,8,2
        IF(ICORN.EQ.6 .AND.
     &                IAND(ICELG2(10,ICELL),HL000F).EQ.0) GOTO 20
C
        INODE=ICELG2(ICORN,ICELL)
        XNODE=GEOMG2(1    ,INODE)
        YNODE=GEOMG2(2    ,INODE)
        CALL GRDRAW(XNODE,YNODE,0)
C
10      CONTINUE
20      CONTINUE
C
        RETURN
        END
```

G2PLTS

```
      SUBROUTINE G2PLTS(XDUM$,YDUM$,NDUM,GRDUMY
     &
     &                                          )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES A SURFACE PLOT FOR THE GRID
C       CONTAINED IN /G2COMN/
C
      DIMENSION XDUM$(*),YDUM$(*),GRDUMY(*)
C
C*****************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
C*****************************************************************
C
      IBCTYP=NINT(GRDUMY(1))
C
C     VISIT ALL BOUNDARY CELLS
C
      DO 80 IBOUND=1,NBNDG2
C
      KNODE=IBNDG2(1,IBOUND)
      ILEFT=IBNDG2(2,IBOUND)
      IEDGE=IBNDG2(4,IBOUND)
      ITYPE=IBNDG2(5,IBOUND)
C
      IF(ITYPE.EQ.0) GOTO 80
C
C     BRANCH DEPENDING ON WHICH EDGE THE BOUNDARY CONDITION REFERS TO
C
      GOTO (80,10,20,10,30,30,20,10,
     &      40,40,10,40,30,30,20,10), (IEDGE+1)
C
C     ...SOUTH SIDE, SOUTHEAST CORNER (OUTSIDE), SOUTHWEST
C       CORNER (INSIDE), OR TRAILING EDGE SLIT
C
10    JNODE=ICELG2(2,ILEFT)
      GOTO 50
C
C     ...EAST SIDE, NORTHEAST CORNER (OUTSIDE), OR SOUTHEAST
C       CORNER (INSIDE)
C
20    JNODE=ICELG2(4,ILEFT)
      GOTO 50
C
C     ...NORTH SIDE, NORTHWEST CORNER (OUTSIDE), NORTHEAST
C       CORNER (INSIDE), OR LEADING EDGE SLIT
```

```
C
30      JNODE=ICELG2(6,ILEFT)
        GOTO 50
C
C     ...WEST SIDE, SOUTHWEST CORNER (OUTSIDE), OR NORTHWEST
C        CORNER (INSIDE)
C
40      JNODE=ICELG2(8,ILEFT)
C
C     LOOP FOR BOUNDARY CONDITION TYPE FOR JNODE
C
50      DO 60 JBOUND=1,NBNDG2
        IF(IBNDG2(1,JBOUND).EQ.JNODE) THEN
           JTYPE=IBNDG2(5,JBOUND)
           GOTO 70
        ENDIF
60      CONTINUE
C
        CALL UTEROR(+1,REAL(KNODE),REAL(JNODE))
C
C     GO TO NEXT PAIR OF NODES IF NEITHER OF THESE IS A SOLID BOUNDARY
C
70      IF(ITYPE.NE.IBCTYP .AND. JTYPE.NE.IBCTYP) GOTO 80
C
C     DRAW THIS SEGMENT OF THE DISTRIBUTION
C
        IF(NDUM.GT.2) THEN
           CALL GRMOVE(GEOMG2(1,KNODE),WORKG2(KNODE),0)
           CALL GRDRAW(GEOMG2(1,JNODE),WORKG2(JNODE),0)
        ELSE
           CALL GRMOVE(GEOMG2(1,KNODE),WORKG2(KNODE),0)
           CALL GRDRAW(GEOMG2(1,JNODE),WORKG2(JNODE),1)
        ENDIF
C
80      CONTINUE
C
C     PLOT THE COMPARISON DATA IF IT EXISTS
C
        DO 90 I=3,NDUM
        CALL GRMOVE(XDUM$(I),YDUM$(I),2)
90      CONTINUE
C
        RETURN
        END
```

G2PLTV

```
        SUBROUTINE G2PLTV(XDUM$,YDUM$,NDUM,GRDUMY
     &
     &                                          )
```

```
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES A VELOCITY VECTOR PLOT FOR THE GRID
C        CONTAINED IN /G2COMM/.  NOTE: THIS SUBROUTINE ASSUMES THAT
C        THE VELOCITY VECTORS ARE DPEN(2)/DPEN(1) AND DPEN(3)/DPEN(1)
C
        DIMENSION XDUM$(*),YDUM$(*),GRDUMY(*)
C
C*******************************************************************************
C
        INCLUDE '[.GRID2D]G2COMM.INC'
C
C*******************************************************************************
C
C     GET THE REQUIRED SYMBOL SIZE
C
        CALL GRINPF('SYMBOL SIZE',SYMSIZ)
        SIZE=ABS(SYMSIZ)
C
C     FOR EACH REGULAR POINT, CALCULATE FLOW ANGLE AND MAGNITUDE
C        AND PLOT BODY OF ARROW.
C
        DO 10 INODE=1,NNODG2
C
        UU=DPENG2(2,INODE)/DPENG2(1,INODE)
        VV=DPENG2(3,INODE)/DPENG2(1,INODE)
C
        CALL GRMOVE(GEOMG2(1,INODE),GEOMG2(2,INODE),0)
C
        XX=SIZE*UU+GEOMG2(1,INODE)
        YY=SIZE*VV+GEOMG2(2,INODE)
        CALL GRDRAW(XX,YY,0)
C
C     DRAW THE HEAD OF THE ARROW
C
        IF(SYMSIZ.LT.0.0) GOTO 10
C
        XX1=XX+SIZE*(-.25*UU-.15*VV)
        YY1=YY+SIZE*(-.25*VV+.15*UU)
        CALL GRDRAW(XX1,YY1,0)
C
        XX2=XX+SIZE*(-.25*UU+.15*VV)
        YY2=YY+SIZE*(-.25*VV-.15*UU)
        CALL GRMOVE(XX2,YY2,0)
C
        CALL GRDRAW(XX,YY,0)
C
10      CONTINUE
C
        RETURN
        END
```

G2PRNT

```
      SUBROUTINE G2PRNT(IOPT
     &
     &                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PRINTS ALL ARRAY VARIABLES.
C       IOPT SELECTS WHICH TABLES ARE TO BE PRINTED:
C
C              IOPT   NODE  CELL  BNDY   AUX
C                0      X     X     X     X
C                1            X     X     X
C                2      X           X     X
C                3                  X     X
C                4      X     X           X
C                5            X           X
C                6      X                 X
C                7                        X
C                8      X     X     X
C                9            X     X
C               10      X           X
C               11                  X
C               12      X     X
C               13            X
C               14      X
C
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
      IF(INBRG2.EQ.0) THEN
        CALL UTEROR(+1,REAL(INBRG2),0.0)
      ENDIF
C
C     WRITE OUT NODE ARRAYS
C
      IF(IAND(IOPT,HL0001).NE.0) GOTO 40
C
      CALL UTHEAD('NODE VARIABLES')
C
      WRITE(JPRINT,10)
```

```
10      FORMAT(   '    INODE        GEOM1          GEOM2      ',
     &          '     DPEN1          DPEN2          DPEN3      ',
     &          '     DPEN4          DPEN5      '               /)
C
        DO 30 INODE=1,NNGDG2
        WRITE(JPRINT,20) INODE,(GEOMG2(K,INODE),K=1,2),
     &                         (DPENG2(K,INODE),K=1,5)
20      FORMAT(1X,(I7,4X),7G15.8)
30      CONTINUE
C
C     WRITE OUT CELL ARRAYS
C
40      IF(IAND(IOPT,HL0002).NE.0) GOTO 80
C
        CALL UTHEAD('CELL VARIABLES')
C
        WRITE(JPRINT,50)
50      FORMAT('  ICELL  CENT SWEST SOUTH SEAST  EAST NEAST NORTH',
     &          ' NWEST  WEST              LFSW   LSW    LS   LSE',
     &          '    LE   LNE    LN   LNW    LW'                  /)
C
        DO 70 ICELL=1,NCELG2
        WRITE(JPRINT,60) ICELL,(ICELG2(K,ICELL),K=1,10),
     &                         (NBORG2(K,ICELL),K=1, 9)
60      FORMAT(1X,10I6,Z10,2X,9I6)
70      CONTINUE
C
C     WRITE OUT BOUNDARY CONDITION ARRAYS
C
80      IF(IAND(IOPT,HL0004).NE.0) GOTO 120
C
        CALL UTHEAD('BOUNDARY CONDITION INFORMATION')
C
        WRITE(JPRINT,90)
90      FORMAT('   IBOUND     NODE     ICLEFT    ICRITE  ',
     &          '    IEDGE     ITYPE    ILEVEL '          /)
C
        DO 110 IBOUND=1,NBNDG2
        WRITE(JPRINT,100) IBOUND,(IBNDG2(K,IBOUND),K=1,6),
     &                           (BONDG2(K,IBOUND),K=1,4)
100     FORMAT(1X,7(I7,3X),4G15.7)
110     CONTINUE
C
C     WRITE OUT MULTIPLE-GRID-LEVEL ARRAY
C
120     IF(IAND(IOPT,HL0008).NE.0) GOTO 220
C
        CALL UTHEAD('AUXILIARY ARRAY INFORMATION')
C
C     ...MULTIPLE GRID-LEVEL ARRAY
C
        WRITE(JPRINT,190)
190     FORMAT(//' MULTIPLE-GRID-LEVEL INFORMATION:'        /
```

```
     &           '              --FINE INTER--      --FINE ',
     &            ' EDGE--      ----COARSE----   '         /
     &           '    ILVL      IBEG      IEND     IBEG   ',
     &            '   IEND      IBEG      IEND   '         )
C
        DO 210 IMGL=-MLVLG2,MLVLG2
        WRITE(JPRINT,200) IMGL,(ILVLG2(K,IMGL),K=1,6)
200     FORMAT(1X,7(I7,3X))
210     CONTINUE
C
220     RETURN
        END
```

G2READ

```
        SUBROUTINE G2READ(IUNIT,IOUT
     &
     &                              )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS GEOMETRY INFORMATION FROM UNIT 'IUNIT'
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        LOGICAL OPENF
C
C*****************************************************************************
C
C     SET INDICATOR INDICATING THAT NEIGHBOR ARRAY IS IN MEMORY (OR
C        AT LEAST WILL BE)
C
        INBRG2=1
C
        INQUIRE(UNIT=IUNIT,OPENED=OPENF)
C
        IF(OPENF) THEN
           READ(IUNIT,10) IMESH
10         FORMAT(I5)
           ITYPE=0
        ELSE
           OPEN(UNIT=IUNIT,STATUS='UNKNOWN',FORM='UNFORMATTED')
           READ(IUNIT) IMESH
           ITYPE=1
        ENDIF
C
C     READ DIFFERENT KINDS OF INPUT AND SET UP TE POINTER SYSTEM
C        FOR DIFFEREENT TYPES OF MESHES
```

```
C
        IF    (IMESH.EQ.0) THEN
           CALL G2INP0(IUNIT,ITYPE)
        ELSEIF(IMESH.EQ.1) THEN
           CALL G2INP1(IUNIT,IOUT)
        ELSEIF(IMESH.EQ.2) THEN
           CALL G2INP2(IUNIT,IOUT)
        ELSEIF(IMESH.EQ.3) THEN
           CALL G2INP3(IUNIT,IOUT)
        ELSEIF(IMESH.EQ.4) THEN
           CALL G2INP4(IUNIT,IOUT)
        ELSEIF(IMESH.EQ.5) THEN
           CALL G2INP5(IUNIT,IOUT)
        ELSEIF(IMESH.EQ.6) THEN
           CALL G2INP6(IUNIT,IOUT)
        ELSEIF(IMESH.EQ.7) THEN
           CALL G2INP7(IUNIT,IOUT)
        ENDIF
C
C     CLOSE THE UNIT
C
        CLOSE(UNIT=IUNIT)
C
        RETURN
        END
```

G2SORT

```
        SUBROUTINE G2SORT
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE SORTS THE CELLS ON ALL FINE MULTIPLE GRID LEVELS
C        INTO THREE LISTS
C        1- FINE NOT JUST OUTSIDE AND NOT AT BOUNDARY
C        2- ALL OTHER FINE
C        3- COARSE
C
C**********************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]UTCOMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C**********************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
```

```
        IF(INBRG2.EQ.0) THEN
           CALL UTEROR(+1,REAL(INBRG2),0.0)
        ENDIF
C
C     CREATE TEMPORARY ARRAY FOR CELLS
C
        CALL UTBASE(+1,NCELG2+1,IBAS1)
        CALL UTBASE(+1,NCELG2+1,IBAS2)
C
        NBAS1=IBAS1+1
        NBAS2=IBAS2+1
C
        VARUT(NBAS1)=0.
        VARUT(NBAS2)=0.
C
C     SET UP THE WORK ARRAY WITH THE CELL NUMBERS
C
        DO 10 ICELL=1,NCELG2
        VARUT(NBAS1+ICELL)=ICELL
10      CONTINUE
C
C     LOOP THROUGH ALL FINE MULTIPLE-GRID LEVELS
C
        DO 120 IMGL=0,MLVLG2
        IF(ILVLG2(1,IMGL).GT.ILVLG2(6,IMGL)) GOTO 120
C
C.....FIRST SORT -- SEPARATE FINE AND COARSE CELLS
C
        ITOP=ILVLG2(1,IMGL)
        IBOT=ILVLG2(6,IMGL)
C
C     IF ENTRY AT ITOP BELONGS ON TOP LIST, THEN INCREMENT ITOP
C
20      IF(ICELG2(1,ITOP).EQ.0) THEN
           ITOP=ITOP+1
           IF(ITOP.LE.IBOT) GOTO 20
                            GOTO 60
        ENDIF
C
C     IF ENTRY AT IBOT BELONGS ON BOTTOM LIST, THEN DECREMENT IBOT
C
30      IF(ICELG2(1,IBOT).NE.0) THEN
           IBOT=IBOT-1
           IF(ITOP.LE.IBOT) GOTO 30
                            GOTO 60
        ENDIF
C
C     SWITCH ENTRIES AT ITOP AND IBOT
C
        TEMP              =VARUT(NBAS1+ITOP)
        VARUT(NBAS1+ITOP)=VARUT(NBAS1+IBOT)
        VARUT(NBAS1+IBOT)=TEMP
C
```

```
        DO 40 K=1,10
        ITEMP          =ICELG2(K,ITOP)
        ICELG2(K,ITOP)=ICELG2(K,IBOT)
        ICELG2(K,IBOT)=ITEMP
40      CONTINUE
C
        DO 50 K=1,9
        ITEMP          =NBORG2(K,ITOP)
        NBORG2(K,ITOP)=NBORG2(K,IBOT)
        NBORG2(K,IBOT)=ITEMP
50      CONTINUE
C
C     CONTINUE THE SORTING PROCESS
C
        GOTO 20
C
C     FIRST SORT HAS ENDED, SO SAVE POINTERS TO BEGINNING AND END
C        OF LISTS
C
60      ILVLG2(4,IMGL)=ITOP-1
        ILVLG2(5,IMGL)=ITOP
C
C.....SECOND SORT -- SEPARATE SIMPLE CELLS FROM INTERFACE AND
C        BOUNDARY CELLS
C
        ITOP=ILVLG2(1,IMGL)
        IBOT=ILVLG2(4,IMGL)
C
C     IF NO FINE CELLS, SET POINTERS APPROPRIATELY
C
        IF(ITOP.GT.IBOT) GOTO 110
C
C     IF ENTRY AT ITOP BELONGS ON TOP LIST, THEN INCREMENT ITOP
C
70      IF(IAND(ICELG2(10,ITOP),HLOOOF).EQ.0 .AND.
     &     IAND(ICELG2(10,ITOP),HLOOFO).EQ.0      ) THEN
           ITOP=ITOP+1
           IF(ITOP.LE.IBOT) GOTO  70
                            GOTO 110
        ENDIF
C
C     IF ENTRY AT IBOT BELONGS ON BOTTOM LIST, THEN DECREMENT IBOT
C
80      IF(IAND(ICELG2(10,IBOT),HLOOOF).NE.0 .OR.
     &     IAND(ICELG2(10,IBOT),HLOOFO).NE.0      ) THEN
           IBOT=IBOT-1
           IF(ITOP.LE.IBOT) GOTO  80
                            GOTO 110
        ENDIF
C
C     SWITCH ENTRIES AT ITOP AND IBOT
C
        TEMP             =VARUT(NBAS1+ITOP)
```

```
        VARUT(NBAS1+ITOP)=VARUT(NBAS1+IBOT)
        VARUT(NBAS1+IBOT)=TEMP
C
        DO 90 K=1,10
        ITEMP          =ICELG2(K,ITOP)
        ICELG2(K,ITOP)=ICELG2(K,IBOT)
        ICELG2(K,IBOT)=ITEMP
90      CONTINUE
C
        DO 100 K=1,9
        ITEMP          =NBORG2(K,ITOP)
        NBORG2(K,ITOP)=NBORG2(K,IBOT)
        NBORG2(K,IBOT)=ITEMP
100     CONTINUE
C
C     CONTINUE THE SORTING PROCESS
C
        GOTO 70
C
C     SECOND SORT HAS ENDED, SO SAVE POINTERS TO BEGINNING AND END
C        OF LISTS
C
110     ILVLG2(2,IMGL)=ITOP-1
        ILVLG2(3,IMGL)=ITOP
C
C     NEXT MULTIPLE-GRID LEVEL
C
120     CONTINUE
C
C     INVERT THE CORRESPONDENCE TABLE
C
        DO 125 I=1,NCELG2
        VARUT(NBAS2+NINT(VARUT(NBAS1+I)))=I
125     CONTINUE
C
C     STEP THROUGH ALL NEIGHBOR POINTERS, REALIGNING TO NEW CELL
C        NUMBERS
C
        DO 130 ICELL=ILVLG2(1,0),ILVLG2(6,MLVLG2)
        NBORG2(1,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(1,ICELL)))),
     &                                              NBORG2(1,ICELL)   )
        NBORG2(2,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(2,ICELL)))),
     &                                              NBORG2(2,ICELL)   )
        NBORG2(3,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(3,ICELL)))),
     &                                              NBORG2(3,ICELL)   )
        NBORG2(4,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(4,ICELL)))),
     &                                              NBORG2(4,ICELL)   )
        NBORG2(5,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(5,ICELL)))),
     &                                              NBORG2(5,ICELL)   )
        NBORG2(6,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(6,ICELL)))),
     &                                              NBORG2(6,ICELL)   )
        NBORG2(7,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(7,ICELL)))),
     &                                              NBORG2(7,ICELL)   )
```

```
        NBORG2(8,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(8,ICELL)))),
     &                                              NBORG2(8,ICELL)   )
        NBORG2(9,ICELL)=SIGN(NINT(VARUT(NBAS2+ABS(NBORG2(9,ICELL)))),
     &                                              NBORG2(9,ICELL)   )
130     CONTINUE
C
C     STEP THROUGH ALL BOUNDARY CONDITION POINTERS, REALIGNING TO
C        NEW CELL NUMBERS
C
        DO 140 IBOUND=1,NBNDG2
        IBNDG2(2,IBOUND)=NINT(VARUT(NBAS2+IBNDG2(2,IBOUND)))
        IBNDG2(3,IBOUND)=NINT(VARUT(NBAS2+IBNDG2(3,IBOUND)))
140     CONTINUE
C
C     RELEASE BASED VARIABLES FOR NODE MOVEMENT
C
        CALL UTBASE(-1,0,IBAS1)
        CALL UTBASE(-1,0,IBAS2)
C
        RETURN
        END
```

G2SUMY

```
        SUBROUTINE G2SUMY(IUNIT
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE WRITE A SUMMARY OF THE POINTER SYSTEM ON UNIT
C        IUNIT
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*****************************************************************************
C
        WRITE(IUNIT,10) TITLGR(1:79)
10      FORMAT(1X,A)
C
        WRITE(IUNIT,20) NNODG2,NBNDG2
20      FORMAT(' NNODG2=',I5,7X,
     &         ' NBNDG2=',I5   )
C
        WRITE(IUNIT,30)
30      FORMAT('              -INTERIOR FINE-     ---EDGE',
```

```
     &          ' FINE---     ----COARSE-----  '          /
     &        '    ILVL      IBEG      IEND      IBEG    ',
     &         '   IEND      IBEG      IEND    '          )
C
        DO 50 IMGL=-MLVLG2,MLVLG2
        IF(ILVLG2(1,IMGL).LE.ILVLG2(6,IMGL))
     &                  WRITE(IUNIT,40) IMGL,(ILVLG2(K,IMGL),K=1,6)
40      FORMAT(1X,7(I7,3X))
50      CONTINUE
C
        RETURN
        END
```

G2SURF

```
        SUBROUTINE G2SURF(IBCTY1,IBCTY2,
     &
     &                    NCHNG        )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE DIVIDES ALL CELLS ADJACENT TO BOUNDARIES WITH A
C        BOUNDARY CONDITION TYPE IBCTY1 OR IBCTY2
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMN.INC'
C
        INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*****************************************************************************
C
        NCHNG=0
C
C     VISIT ALL CELLS ON ALL FINE LEVELS
C
        DO 90 ICELL=NCELG2,ILVLG2(1,0),-1
        IF(ICELG2(2,ICELL).EQ.0) GOTO 90
C
C     DO NOT CONSIDER THIS CALL IF NOT NEAR A BOUNADRY
C
        IF(IAND(ICELG2(10,ICELL),HL000F).EQ.HL0000) GOTO 90
C
C     CHECK EACH OF THE NODES, AND IF A BOUNDARY NODE OF THE
C        APPROPRIATE TYPE, THEN DIVIDE IT
C
C     ...SOUTHWEST
C
        IF(IAND(ICELG2(10,ICELL),HL0001).EQ.HL0000) GOTO 20
        INODE=ICELG2(2,ICELL)
```

```
C
      DO 10 IBOUND=1,NBNDG2
      IF(IBNDG2(1,IBOUND).EQ.INODE) THEN
         IF(IBNDG2(5,IBOUND).EQ.IBCTY1 .OR.
     &      IBNDG2(5,IBOUND).EQ.IBCTY2     ) THEN
            GOTO 80
         ELSE
            GOTO 20
         ENDIF
      ENDIF
10    CONTINUE
C
C     BOUNDARY POINTER FOR THIS NODE NOT FOUND
C
      CALL UTEROR(+1,REAL(INODE),REAL(ICELL))
C
C     ...SOUTHEAST
C
20    IF(IAND(ICELG2(10,ICELL),HL0002).EQ.HL0000) GOTO 40
      INODE=ICELG2(4,ICELL)
C
      DO 30 IBOUND=1,NBNDG2
      IF(IBNDG2(1,IBOUND).EQ.INODE) THEN
         IF(IBNDG2(5,IBOUND).EQ.IBCTY1 .OR.
     &      IBNDG2(5,IBOUND).EQ.IBCTY2     ) THEN
            GOTO 80
         ELSE
            GOTO 40
         ENDIF
      ENDIF
30    CONTINUE
C
C     BOUNDARY POINTER FOR THIS NODE NOT FOUND
C
      CALL UTEROR(+2,REAL(INODE),REAL(ICELL))
C
C     ...NORTHEAST
C
40    IF(IAND(ICELG2(10,ICELL),HL0004).EQ.HL0000) GOTO 60
      INODE=ICELG2(6,ICELL)
C
      DO 50 IBOUND=1,NBNDG2
      IF(IBNDG2(1,IBOUND).EQ.INODE) THEN
         IF(IBNDG2(5,IBOUND).EQ.IBCTY1 .OR.
     &      IBNDG2(5,IBOUND).EQ.IBCTY2     ) THEN
            GOTO 80
         ELSE
            GOTO 60
         ENDIF
      ENDIF
50    CONTINUE
C
C     BOUNDARY POINTER FOR THIS NODE NOT FOUND
```

```
C
        CALL UTEROR(+3,REAL(INODE),REAL(ICELL))
C
C     ...NORTHWEST
C
60      IF(IAND(ICELG2(10,ICELL),HL0008).EQ.HL0000) GOTO 90
        INODE=ICELG2(8,ICELL)
C
        DO 70 IBOUND=1,NBNDG2
        IF(IBNDG2(1,IBOUND).EQ.INODE) THEN
           IF(IBNDG2(5,IBOUND).EQ.IBCTY1 .OR.
     &        IBNDG2(5,IBOUND).EQ.IBCTY2     ) THEN
              GOTO 80
           ELSE
              GOTO 90
           ENDIF
        ENDIF
70      CONTINUE
C
C     BOUNDARY POINTER FOR THIS NODE NOT FOUND
C
        CALL UTEROR(+4,REAL(INODE),REAL(ICELL))
C
C     DIVIDE THIS CELL
C
80      CALL G2DIVD(ICELL,ICHNG)
        NCHNG=NCHNG+ICHNG
C
90      CONTINUE
C
C     GARBAGE COLLECT
C
        CALL G2GRBG
C
        RETURN
        END
```

G2VOID

```
        SUBROUTINE G2VOID(
     &
     &                    NCHNG)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE ENSURES THAT THE CURRENT GRID DOESN'T HAVE ANY
C        VOIDS.  A VOID IS DEFINED AS AN UNDIVIDED CELL WHICH IS
C        BOUNDED AT ALL FOUR NODES BY EITHER ADAPTATION OR A BOUNDARY.
C        IT ALSO GETS RID OF ISLANDS BY COLLAPSING
C
```

```
C*********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.UTILITY]HEXCOD.INC'
C
C*********************************************************************
C
C     INITIALIZE NUMBER OF DIVIDED AND COLLAPSED CELLS
C
        NDIVD=0
        NFUSE=0
C
C     LOOP THROUGH ALL FINE CELLS TO CHECK FOR VOIDS
C
        DO 10 ICELL=ILVLG2(6,MLVLG2-1),ILVLG2(1,0),-1
        IF(ICELG2(1,ICELL).NE.0 .OR. ICELG2(2,ICELL).EQ.0) GOTO 10
C
C     COUNT THE NUMBER OF SIDES WHICH ARE EITHER ON A BOUNDARY OR AN
C       INTERFACE
C
        IAUX  =ICELG2(10,ICELL)
        NNODES=0
C
        IF(IAND(IAUX,HL0001).EQ.HL0001 .OR.
     &     IAND(IAUX,HL0010).EQ.HL0010    ) NNODES=NNODES+1
        IF(IAND(IAUX,HL0002).EQ.HL0002 .OR.
     &     IAND(IAUX,HL0020).EQ.HL0020    ) NNODES=NNODES+1
        IF(IAND(IAUX,HL0004).EQ.HL0004 .OR.
     &     IAND(IAUX,HL0040).EQ.HL0040    ) NNODES=NNODES+1
        IF(IAND(IAUX,HL0008).EQ.HL0008 .OR.
     &     IAND(IAUX,HL0080).EQ.HL0080    ) NNODES=NNODES+1
C
C     IF ALL FOUR NODES ARE FLAGGED, THEN THIS IS A VOID
C
        IF(NNODES.EQ.4) THEN
          CALL G2DIVD(ICELL,ICHANG)
          NDIVD=NDIVD+ICHANG
        ENDIF
C
10      CONTINUE
C
C     LOOP THROUGH ALL COARSE CELLS TO CHECK FOR ISLANDS
C
        DO 20 ICELL=ILVLG2(6,MLVLG2-1),ILVLG2(1,0),-1
        IF(ICELG2(1,ICELL).EQ.0 .OR. ICELG2(2,ICELL).EQ.0) GOTO 20
C
C     COUNT THE NUMBER OF SIDES WHICH ARE EITHER ON A BOUNDARY OR AN
C       INTERFACE
C
        IAUX  =ICELG2(10,ICELL)
        NNODES=0
C
```

```
      IF(IAND(IAUX,HL0001).EQ.HL0001 .OR.
     &   IAND(IAUX,HL0100).EQ.HL0100    ) NNODES=NNODES+1
      IF(IAND(IAUX,HL0002).EQ.HL0002 .OR.
     &   IAND(IAUX,HL0200).EQ.HL0200    ) NNODES=NNODES+1
      IF(IAND(IAUX,HL0004).EQ.HL0004 .OR.
     &   IAND(IAUX,HL0400).EQ.HL0400    ) NNODES=NNODES+1
      IF(IAND(IAUX,HL0008).EQ.HL0008 .OR.
     &   IAND(IAUX,HL0800).EQ.HL0800    ) NNODES=NNODES+1
C
C     IF ALL FOUR NODES ARE FLAGGED, THEN THIS IS AN ISLAND
C
      IF(NNODES.EQ.4) THEN
         CALL G2FUSE(ICELL,ICHANG)
         NFUSE=NFUSE+ICHANG
      ENDIF
C
20    CONTINUE
C
      NCHNG=NDIVD+NFUSE
C
      RETURN
      END
```

G2WRIT

```
      SUBROUTINE G2WRIT(IUNIT
     &
     &                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE WRITES GEOMETRY INFORMATION TO IUNIT.
C
C**********************************************************************
C
      INCLUDE '[.GRID2D]G2COMN.INC'
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C**********************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
      IF(INBRG2.EQ.0) THEN
         CALL UTEROR(+1,REAL(INBRG2),0.0)
      ENDIF
C
      OPEN(UNIT=IUNIT,STATUS='NEW',FORM='UNFORMATTED')
C
C     WRITE MESH INDICATOR
```

```
C
        IMESH=0
        WRITE(IUNIT) IMESH
C
        WRITE(IUNIT) NNODG2,NCELG2,NBNDG2,NBODG2,KDEFG2,
     &               DATEGR,TIMEGR
C
        WRITE(IUNIT) ((GEOMG2(K,INODE),K=1,2),
     &                (DPENG2(K,INODE),K=1,6),INODE=1,NNODG2)
C
        WRITE(IUNIT) ((ICELG2(K,ICELL),K=1,10),
     &                (NBORG2(K,ICELL),K=1, 9),ICELL=1,NCELG2)
C
        WRITE(IUNIT) ((ILVLG2(K,ILEVEL),K=1,6),ILEVEL=-NLVLG2,MLVLG2)
C
        WRITE(IUNIT) ((IBNDG2(K,IBOUND),K=1,6),
     &                (BONDG2(K,IBOUND),K=1,9),IBOUND=1,NBNDG2)
C
        WRITE(IUNIT) ((BODYG2(K,IBODY),K=1,11),IBODY=1,NBODG2)
C
        CLOSE(UNIT=IUNIT)
C
        RETURN
        END
```

G2XPND

```
        SUBROUTINE G2XPND(IMGL,NADD
     &
     &                               )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE EXPANDS THE LEVEL IMGL CELL ARRAYS BY NADD CELLS.
C        THE NEW BLANK CELLS ARE INITIALLY MARKED FOR DELETE
C
C*****************************************************************************
C
        INCLUDE '[.GRID2D]G2COMM.INC'
C
C*****************************************************************************
C
C     MAKE SURE NEIGHBOR TABLE IS IN MEMORY
C
        IF(INBRG2.EQ.0) THEN
           CALL UTEROR(+1,REAL(INBRG2),0.0)
        ENDIF
C
C     MAKE SURE A VALID LEVEL FOR EXPANSION WAS CHOSEN
C
```

```
        IF(IMGL.LE.0 .OR. IMGL.GT.MLVLG2) THEN
           CALL UTEROR(+2,REAL(IMGL),REAL(MLVLG2))
        ENDIF
C
C     CHECK IF THERE IS ENOUGH ROOM FOR THE EXPANSION
C
        IF((ILVLG2(6,MLVLG2+1)+NADD).GT.MCELG2) THEN
           CALL UTEROR(+3,REAL(ILVLG2(6,MLVLG2+1)),REAL(NADD))
        ENDIF
C
C     IF NO CELL VARIABLES ACTUALLY HAVE TO BE MOVED, SKIP THE
C        NEXT SECTION
C
        IFIRST=ILVLG2(1,IMGL+1)
        ILAST =ILVLG2(6,MLVLG2)
        IF(IFIRST.GT.ILAST) GOTO 40
C
C     MOVE THE CELL PLOINTER AND NEIGHBOR INFORMATION
C
        DO 10 ICELL=ILAST,IFIRST,-1
C
        ICELG2( 1,ICELL+NADD)=ICELG2( 1,ICELL)
        ICELG2( 2,ICELL+NADD)=ICELG2( 2,ICELL)
        ICELG2( 3,ICELL+NADD)=ICELG2( 3,ICELL)
        ICELG2( 4,ICELL+NADD)=ICELG2( 4,ICELL)
        ICELG2( 5,ICELL+NADD)=ICELG2( 5,ICELL)
        ICELG2( 6,ICELL+NADD)=ICELG2( 6,ICELL)
        ICELG2( 7,ICELL+NADD)=ICELG2( 7,ICELL)
        ICELG2( 8,ICELL+NADD)=ICELG2( 8,ICELL)
        ICELG2( 9,ICELL+NADD)=ICELG2( 9,ICELL)
        ICELG2(10,ICELL+NADD)=ICELG2(10,ICELL)
C
        NBORG2( 1,ICELL+NADD)=NBORG2( 1,ICELL)
        NBORG2( 2,ICELL+NADD)=NBORG2( 2,ICELL)
        NBORG2( 3,ICELL+NADD)=NBORG2( 3,ICELL)
        NBORG2( 4,ICELL+NADD)=NBORG2( 4,ICELL)
        NBORG2( 5,ICELL+NADD)=NBORG2( 5,ICELL)
        NBORG2( 6,ICELL+NADD)=NBORG2( 6,ICELL)
        NBORG2( 7,ICELL+NADD)=NBORG2( 7,ICELL)
        NBORG2( 8,ICELL+NADD)=NBORG2( 8,ICELL)
        NBORG2( 9,ICELL+NADD)=NBORG2( 9,ICELL)
C
10      CONTINUE
C
C     ADJUST ANY NEIGHBOR POINTERS WHICH REFER TO CELLS JUST MOVED
C
        DO 20 ICEL=ILVLG2(1,IMGL),(ILAST+NADD)
        IF(NBORG2(1,ICEL).GE.IFIRST) NBORG2(1,ICEL)=NBORG2(1,ICEL)+NADD
        IF(NBORG2(2,ICEL).GE.IFIRST) NBORG2(2,ICEL)=NBORG2(2,ICEL)+NADD
        IF(NBORG2(3,ICEL).GE.IFIRST) NBORG2(3,ICEL)=NBORG2(3,ICEL)+NADD
        IF(NBORG2(4,ICEL).GE.IFIRST) NBORG2(4,ICEL)=NBORG2(4,ICEL)+NADD
        IF(NBORG2(5,ICEL).GE.IFIRST) NBORG2(5,ICEL)=NBORG2(5,ICEL)+NADD
        IF(NBORG2(6,ICEL).GE.IFIRST) NBORG2(6,ICEL)=NBORG2(6,ICEL)+NADD
```

```
        IF(NBORG2(7,ICEL).GE.IFIRST) NBORG2(7,ICEL)=NBORG2(7,ICEL)+NADD
        IF(NBORG2(8,ICEL).GE.IFIRST) NBORG2(8,ICEL)=NBORG2(8,ICEL)+NADD
        IF(NBORG2(9,ICEL).GE.IFIRST) NBORG2(9,ICEL)=NBORG2(9,ICEL)+NADD
20      CONTINUE
C
C     ADJUST ANY BOUNDARY CONDITION POINTERS WHICH REFER TO CELLS JUST
C       MOVED
C
        DO 30 IBND=1,NBNDG2
        IF(IBNDG2(2,IBND).GE.IFIRST) IBNDG2(2,IBND)=IBNDG2(2,IBND)+NADD
        IF(IBNDG2(3,IBND).GE.IFIRST) IBNDG2(3,IBND)=IBNDG2(3,IBND)+NADD
30      CONTINUE
C
C     ADJUST THE MULTIPLE-GRID-LEVEL ARRAY
C
40      DO 50 ILEV=IMGL+1,MLVLG2+1
        ILVLG2(1,ILEV)=ILVLG2(1,ILEV)+NADD
        ILVLG2(2,ILEV)=ILVLG2(2,ILEV)+NADD
        ILVLG2(3,ILEV)=ILVLG2(3,ILEV)+NADD
        ILVLG2(4,ILEV)=ILVLG2(4,ILEV)+NADD
        ILVLG2(5,ILEV)=ILVLG2(5,ILEV)+NADD
        ILVLG2(6,ILEV)=ILVLG2(6,ILEV)+NADD
50      CONTINUE
C
C     INITIALIZE ALL ADDED CELLS BY MARKING THEM FOR DELETE
C
        DO 60 ICELL=(ILVLG2(6,IMGL)+1),(ILVLG2(1,IMGL+1)-1)
        ICELG2(2,ICELL)=0
60      CONTINUE
C
        RETURN
        END
```

B.6 UTILTY — Utility Subroutines

HEXCOD

```
C-----HEXADECIMAL CODES USED FOR ENCODING/DECODING BIT INFORMATION
C
        INTEGER HL0000 /Z00000000/, HL0001 /Z00000001/,
     &          HL0002 /Z00000002/, HL0003 /Z00000003/,
     &          HL0004 /Z00000004/, HL0005 /Z00000005/,
     &          HL0006 /Z00000006/, HL0007 /Z00000007/,
     &          HL0008 /Z00000008/, HL0009 /Z00000009/,
     &          HL000A /Z0000000A/, HL000B /Z0000000B/,
     &          HL000C /Z0000000C/, HL000D /Z0000000D/,
     &          HL000E /Z0000000E/, HL000F /Z0000000F/
```

```
      INTEGER                    HL0010 /Z00000010/,
     &        HL0020 /Z00000020/, HL0030 /Z00000030/,
     &        HL0040 /Z00000040/, HL0050 /Z00000050/,
     &        HL0060 /Z00000060/, HL0070 /Z00000070/,
     &        HL0080 /Z00000080/, HL0090 /Z00000090/,
     &        HL00A0 /Z000000A0/, HL00B0 /Z000000B0/,
     &        HL00C0 /Z000000C0/, HL00D0 /Z000000D0/,
     &        HL00E0 /Z000000E0/, HL00F0 /Z000000F0/
      INTEGER                    HL0100 /Z00000100/,
     &        HL0200 /Z00000200/, HL0300 /Z00000300/,
     &        HL0400 /Z00000400/, HL0500 /Z00000500/,
     &        HL0600 /Z00000600/, HL0700 /Z00000700/,
     &        HL0800 /Z00000800/, HL0900 /Z00000900/,
     &        HL0A00 /Z00000A00/, HL0B00 /Z00000B00/,
     &        HL0C00 /Z00000C00/, HL0D00 /Z00000D00/,
     &        HL0E00 /Z00000E00/, HL0F00 /Z00000F00/
      INTEGER                    HL1000 /Z00001000/,
     &        HL2000 /Z00002000/, HL3000 /Z00003000/,
     &        HL4000 /Z00004000/, HL5000 /Z00005000/,
     &        HL6000 /Z00006000/, HL7000 /Z00007000/,
     &        HL8000 /Z00008000/, HL9000 /Z00009000/,
     &        HLA000 /Z0000A000/, HLB000 /Z0000B000/,
     &        HLC000 /Z0000C000/, HLD000 /Z0000D000/,
     &        HLE000 /Z0000E000/, HLF000 /Z0000F000/
      INTEGER HU0000 /Z00000000/, HU0001 /Z00010000/,
     &        HU0002 /Z00020000/, HU0003 /Z00030000/,
     &        HU0004 /Z00040000/, HU0005 /Z00050000/,
     &        HU0006 /Z00060000/, HU0007 /Z00070000/,
     &        HU0008 /Z00080000/, HU0009 /Z00090000/,
     &        HU000A /Z000A0000/, HU000B /Z000B0000/,
     &        HU000C /Z000C0000/, HU000D /Z000D0000/,
     &        HU000E /Z000E0000/, HU000F /Z000F0000/
      INTEGER                    HU0010 /Z00100000/,
     &        HU0020 /Z00200000/, HU0030 /Z00300000/,
     &        HU0040 /Z00400000/, HU0050 /Z00500000/,
     &        HU0060 /Z00600000/, HU0070 /Z00700000/,
     &        HU0080 /Z00800000/, HU0090 /Z00900000/,
     &        HU00A0 /Z00A00000/, HU00B0 /Z00B00000/,
     &        HU00C0 /Z00C00000/, HU00D0 /Z00D00000/,
     &        HU00E0 /Z00E00000/, HU00F0 /Z00F00000/
      INTEGER                    HU0100 /Z01000000/,
     &        HU0200 /Z02000000/, HU0300 /Z03000000/,
     &        HU0400 /Z04000000/, HU0500 /Z05000000/,
     &        HU0600 /Z06000000/, HU0700 /Z07000000/,
     &        HU0800 /Z08000000/, HU0900 /Z09000000/,
     &        HU0A00 /Z0A000000/, HU0B00 /Z0B000000/,
     &        HU0C00 /Z0C000000/, HU0D00 /Z0D000000/,
     &        HU0E00 /Z0E000000/, HU0F00 /Z0F000000/
      INTEGER                    HU1000 /Z10000000/,
     &        HU2000 /Z20000000/, HU3000 /Z30000000/,
     &        HU4000 /Z40000000/, HU5000 /Z50000000/,
     &        HU6000 /Z60000000/, HU7000 /Z70000000/,
     &        HU8000 /Z80000000/, HU9000 /Z90000000/,
```

```
     &          HUA000 /ZA0000000/, HUB000 /ZB0000000/,
     &          HUC000 /ZC0000000/, HUD000 /ZD0000000/,
     &          HUE000 /ZE0000000/, HUF000 /ZF0000000/
        INTEGER HLFFF0 /ZFFFFFFF0/, HLFFF1 /ZFFFFFFF1/,
     &          HLFFF2 /ZFFFFFFF2/, HLFFF3 /ZFFFFFFF3/,
     &          HLFFF4 /ZFFFFFFF4/, HLFFF5 /ZFFFFFFF5/,
     &          HLFFF6 /ZFFFFFFF6/, HLFFF7 /ZFFFFFFF7/,
     &          HLFFF8 /ZFFFFFFF8/, HLFFF9 /ZFFFFFFF9/,
     &          HLFFFA /ZFFFFFFFA/, HLFFFB /ZFFFFFFFB/,
     &          HLFFFC /ZFFFFFFFC/, HLFFFD /ZFFFFFFFD/,
     &          HLFFFE /ZFFFFFFFE/, HLFFFF /ZFFFFFFFF/
        INTEGER HLFF0F /ZFFFFFF0F/, HLFF1F /ZFFFFFF1F/,
     &          HLFF2F /ZFFFFFF2F/, HLFF3F /ZFFFFFF3F/,
     &          HLFF4F /ZFFFFFF4F/, HLFF5F /ZFFFFFF5F/,
     &          HLFF6F /ZFFFFFF6F/, HLFF7F /ZFFFFFF7F/,
     &          HLFF8F /ZFFFFFF8F/, HLFF9F /ZFFFFFF9F/,
     &          HLFFAF /ZFFFFFFAF/, HLFFBF /ZFFFFFFBF/,
     &          HLFFCF /ZFFFFFFCF/, HLFFDF /ZFFFFFFDF/,
     &          HLFFEF /ZFFFFFFEF/
        INTEGER HLF0FF /ZFFFFF0FF/, HLF1FF /ZFFFFF1FF/,
     &          HLF2FF /ZFFFFF2FF/, HLF3FF /ZFFFFF3FF/,
     &          HLF4FF /ZFFFFF4FF/, HLF5FF /ZFFFFF5FF/,
     &          HLF6FF /ZFFFFF6FF/, HLF7FF /ZFFFFF7FF/,
     &          HLF8FF /ZFFFFF8FF/, HLF9FF /ZFFFFF9FF/,
     &          HLFAFF /ZFFFFFAFF/, HLFBFF /ZFFFFFBFF/,
     &          HLFCFF /ZFFFFFCFF/, HLFDFF /ZFFFFFDFF/,
     &          HLFEFF /ZFFFFFEFF/
        INTEGER HL0FFF /ZFFFF0FFF/, HL1FFF /ZFFFF1FFF/,
     &          HL2FFF /ZFFFF2FFF/, HL3FFF /ZFFFF3FFF/,
     &          HL4FFF /ZFFFF4FFF/, HL5FFF /ZFFFF5FFF/,
     &          HL6FFF /ZFFFF6FFF/, HL7FFF /ZFFFF7FFF/,
     &          HL8FFF /ZFFFF8FFF/, HL9FFF /ZFFFF9FFF/,
     &          HLAFFF /ZFFFFAFFF/, HLBFFF /ZFFFFBFFF/,
     &          HLCFFF /ZFFFFCFFF/, HLDFFF /ZFFFFDFFF/,
     &          HLEFFF /ZFFFFEFFF/
        INTEGER HUFFF0 /ZFFF0FFFF/, HUFFF1 /ZFFF1FFFF/,
     &          HUFFF2 /ZFFF2FFFF/, HUFFF3 /ZFFF3FFFF/,
     &          HUFFF4 /ZFFF4FFFF/, HUFFF5 /ZFFF5FFFF/,
     &          HUFFF6 /ZFFF6FFFF/, HUFFF7 /ZFFF7FFFF/,
     &          HUFFF8 /ZFFF8FFFF/, HUFFF9 /ZFFF9FFFF/,
     &          HUFFFA /ZFFFAFFFF/, HUFFFB /ZFFFBFFFF/,
     &          HUFFFC /ZFFFCFFFF/, HUFFFD /ZFFFDFFFF/,
     &          HUFFFE /ZFFFEFFFF/, HUFFFF /ZFFFFFFFF/
        INTEGER HUFF0F /ZFF0FFFFF/, HUFF1F /ZFF1FFFFF/,
     &          HUFF2F /ZFF2FFFFF/, HUFF3F /ZFF3FFFFF/,
     &          HUFF4F /ZFF4FFFFF/, HUFF5F /ZFF5FFFFF/,
     &          HUFF6F /ZFF6FFFFF/, HUFF7F /ZFF7FFFFF/,
     &          HUFF8F /ZFF8FFFFF/, HUFF9F /ZFF9FFFFF/,
     &          HUFFAF /ZFFAFFFFF/, HUFFBF /ZFFBFFFFF/,
     &          HUFFCF /ZFFCFFFFF/, HUFFDF /ZFFDFFFFF/,
     &          HUFFEF /ZFFEFFFFF/
        INTEGER HUF0FF /ZF0FFFFFF/, HUF1FF /ZF1FFFFFF/,
     &          HUF2FF /ZF2FFFFFF/, HUF3FF /ZF3FFFFFF/,
```

```
     &          HUF4FF /ZF4FFFFFF/, HUF5FF /ZF5FFFFFF/,
     &          HUF6FF /ZF6FFFFFF/, HUF7FF /ZF7FFFFFF/,
     &          HUF8FF /ZF8FFFFFF/, HUF9FF /ZF9FFFFFF/,
     &          HUFAFF /ZFAFFFFFF/, HUFBFF /ZFBFFFFFF/,
     &          HUFCFF /ZFCFFFFFF/, HUFDFF /ZFDFFFFFF/,
     &          HUFEFF /ZFEFFFFFF/
        INTEGER HU0FFF /Z0FFFFFFF/, HU1FFF /Z1FFFFFFF/,
     &          HU2FFF /Z2FFFFFFF/, HU3FFF /Z3FFFFFFF/,
     &          HU4FFF /Z4FFFFFFF/, HU5FFF /Z5FFFFFFF/,
     &          HU6FFF /Z6FFFFFFF/, HU7FFF /Z7FFFFFFF/,
     &          HU8FFF /Z8FFFFFFF/, HU9FFF /Z9FFFFFFF/,
     &          HUAFFF /ZAFFFFFFF/, HUBFFF /ZBFFFFFFF/,
     &          HUCFFF /ZCFFFFFFF/, HUDFFF /ZDFFFFFFF/,
     &          HUEFFF /ZEFFFFFFF/
```

IOUNIT

```
C-----INPUT/OUTPUT UNIT DEFINITIONS
C     JTERMI       TERMINAL INPUT
C     JTERMO       TERMINAL OUTPUT
C     JDUMMY       DUMMY UNIT
C     JPRINT       LISTING (PRINT) FILE
C     JCARDS       INPUT (DATA) FILE
C     JCMPAR       COMPARISON DATA FILE
C     JSTOPR       STOP-FLAG FILE (USED TO INTERACTIVELY STOP
C                  A JOB IN PROGRESS)
C     JFIL11-JFIL20
C                  GENERAL INPUT/OUTPUT FILES
C
       COMMON /IOUNIT/ JTERMI,JTERMO,JDUMMY,JPRINT,JCARDS,
     &                 JCMPAR,JSTOPR,
     &                 JFIL11,JFIL12,JFIL13,JFIL14,JFIL15,
     &                 JFIL16,JFIL17,JFIL18,JFIL19,JFIL20
```

PROLOG

```
C*************************************************************************C
C*************************************************************************C
C**                                                                     **C
C**      MM    MM  IIIIIII  TTTTTT   0000    SSSS   IIIIIII   SSSS     **C
C**      MM    MM     II       TT    00  00  SS  SS     II    SS  SS    **C
C**      MMM  MMM     II       TT    00  00  SSS        II    SSS       **C
C**      MM M MM      II       TT    00  00     SSS     II       SSS    **C
C**      MM    MM     II       TT    00  00  SS  SS     II    SS  SS    **C
C**      MM    MM  IIIIIII     TT     0000    SSSS   IIIIIII   SSSS     **C
C**                                                                     **C
C**              Massachusetts Institute of Technology                  **C
```

```
C**               Optimal Solver for Inviscid Systems                **C
C**                                                                  **C
C**                         Version 1.0                              **C
C**                                                                  **C
C**                by John Francis Dannenhoffer, III                 **C
C**                                                                  **C
C**                                                                  **C
C**    Copyright 1987 by the Massachusetts Institute of Technology   **C
C**                        All rights reserved.                      **C
C**                                                                  **C
C**********************************************************************C
C**********************************************************************C
```

UTCOMN

```
C-----BASED VARIABLE TABLE PARAMETERS
C     IOFTUT(I)    OFFSET OF THE ITH ALLOCATED VARIABLE FROM THE POOL
C     VARUT        BASED VARIABLE POOL
C
        PARAMETER (NENTUT=20,
     &             NVARUT=200000)
C
        COMMON /UTCOMN/ IOFTUT(NENTUT+1),VARUT(NVARUT)
```

UTBASE

```
        SUBROUTINE UTBASE(MODE,LENGTH,
     &                    IOFSET
     &                                 )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE SIMULATES PL/1 BASED VARIABLES BY ASSIGNING
C        TEMPORARY WORK SPACE FORM A CENTRAL POOL.  FOR EASE OF
C        IMPLEMENTATION,  THE WORK SPACE IS ONLY CONTRACTED IF THE LAST
C        WORK SPACE HAS BEEN RELEASED OR ALL ARRAYS HAVE BEEN RELEASED.
C
C        MODE=-1 RELEASE A WORK SPACE ENTRY (IOFSET IS GIVEN)
C            = 0 INITIALIZE WORK SPACE
C            =+1 ALLOCATE A WORK SPACE ENTRY (IOFSET IS GENERATED)
C
C**********************************************************************
C
        INCLUDE '[.UTILITY]UTCOMN.INC'
C
C**********************************************************************
C
```

```
      IF(MODE) 300,100,200
C
C*****INITIALIZE
C
100   NENTRY=0
      IOFTUT(1)=1
C
      RETURN
C
C*****ALLOCATE
C
C     CHECK IF ANY MORE ENTRIES ARE ALLOWED
C
200   IF(NENTRY.LT.NENTUT) GOTO 210
C
C     ERROR EXIT 1
C
      CALL UTEROR(+1,REAL(NENTRY),REAL(NENTUT))
C
C     CHECK IF THERE IS ENOUGH ROOM IN BASED VARIABLE WORK VECTOR
C
210   IF((ABS(IOFTUT(NENTRY+1))+LENGTH).LE.NVARUT) GOTO 220
C
C     ERROR EXIT 2
C
      CALL UTEROR(+2,REAL(LENGTH),REAL(NVARUT))
C
C     ALLOCATE STORAGE FOR THIS CALL
C
220   NENTRY=NENTRY+1
      IOFTUT(NENTRY)=ABS(IOFTUT(NENTRY))
      IOFSET=IOFTUT(NENTRY)
      IOFTUT(NENTRY+1)=IOFSET+LENGTH
C
      RETURN
C
C*****RELEASE
C
C     FIND IOFSET IN IOFTUT
C
300   DO 310 IENTRY=1,NENTRY
      MENTRY=IENTRY
      IF(IOFSET.EQ.IOFTUT(MENTRY)) GOTO 320
310   CONTINUE
C
C     ERROR EXIT 3
C
      CALL UTEROR(+3,REAL(IOFSET),REAL(MENTRY))
C
C     FLAG CORRESPONDING ENTRY IN OFFSET TABLE
C
320   IOFTUT(MENTRY)=-IOFTUT(MENTRY)
C
```

```
C     CONTRACT IF LAST VARIABLE HAS BEEN RELEASED
C
        MENTRY=NENTRY
        DO 330 IENTRY=MENTRY,1,-1
        IF(IOFTUT(IENTRY).GT.0) RETURN
        NENTRY=NENTRY-1
330     CONTINUE
C
C     REINITIALIZE SINCE ALL VARIABLES HAVE BEEN RELEASED
C
        GOTO 100
C
        END
```

UTBINS

```
        SUBROUTINE UTBINS(X$,NPTS,ARG,
     &
     &                    INT        )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE FINDS THE INTERVAL IN X$ INTO
C        WHICH ARG FALLS.  EXCEPT WHEN EXTRAPOLATING,
C             X$(INT).LE.ARG.LT.X$(INT+1)
C
        DIMENSION X$(NPTS)
C
C*********************************************************************
C
C     CHECK IF IN FIRST INTERVAL OR EXTRAPLOTAING TO LEFT
C
        IF(ARG.LT.X$(2)) THEN
           INT=1
           RETURN
        ENDIF
C
C     CHECK IF IN LAST INTERVAL OR EXTRAPOLATING TO RIGHT
C
        IF(ARG.GE.X$(NPTS-1)) THEN
           INT=NPTS-1
           RETURN
        ENDIF
C
C     SEARCH THROUGH INTERIOR INTERVALS
C
        ILEFT=2
        IRIGHT=NPTS-1
C
10      IF(ABS(IRIGHT-ILEFT).LE.1) THEN
```

```
          INT=ILEFT
          RETURN
        ENDIF
C
C     DIVIDE SPAN OF SEARCH BY 2
C
        IMIDL=(ILEFT+IRIGHT)/2
        IF(ARG-X$(IMIDL)) 20,30,40
C
C     USE LEFT SIDE INTERVAL
C
20      IRIGHT=IMIDL
        GOTO 10
C
C     INTERVAL HAS BEEN FOUND
C
30      INT=IMIDL
        RETURN
C
C     USE RIGHT SIDE INTERVAL
C
40      ILEFT=IMIDL
        GOTO 10
C
        END
```

UTEROR

```
        SUBROUTINE UTEROR(IER,ERR1,ERR2
     &
     &                                  )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS ROUTINE HANDLES TWO TYPES OF ERRORS:
C        IF IER <0  **WARNING**  ERROR NUMBER IS LOGGED
C               =0  **NORMAL**   NO ERROR
C               >0  **ERROR**    SIGNALS AN ERROR TO THE OPERATING
C                                SYSTEM AND THEN STOPS
C
C*********************************************************************
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
        IF(IER.GT.0) THEN
C
C*****FATAL ERROR
C
```

```
            WRITE(JTERMO,10) IER,ERR1,ERR2
10          FORMAT(' ERROR',I3,' DETECTED (',G16.7,',',G15.7,')')
C
            WRITE(JPRINT,20) IER,ERR1,ERR2
20          FORMAT(//' ERROR NUMBER',I3,6X,2G13.7)
C
            CALL LIB$SIGNAL(%VAL(2))
            STOP
C
         ELSEIF(IER.LT.0) THEN
C
C*****WARNING
C
            WRITE(JTERMO,10) IER,ERR1,ERR2
            WRITE(JPRINT,20) IER,ERR1,ERR2
C
         ENDIF
C
         RETURN
         END
```

UTHEAD

```
         SUBROUTINE UTHEAD(ITEXT
     &
     &                          )
C
         INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE WRITES A HEADER ON UNIT JPRINT
C
         CHARACTER*(*) ITEXT
C
C*************************************************************************
C
         INCLUDE '[.UTILITY]IOUNIT.INC'
C
         INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*************************************************************************
C
         WRITE(JPRINT,10) TITLGR,DATEGR,TIMEGR,
     &                    ITEXT
10       FORMAT('1',A80,T100,'ON: ',A9,' AT: ',A8/
     &          1X,A                              //)
C
         RETURN
         END
```

UTINIT

```
      SUBROUTINE UTINIT(IOPT
     &
     &                  )
C
      INCLUDE '[.UTILITY]PRGLOG.INC'
C
C     THIS SUBROUTINE DOES ALL THE INPUT/OUTPUT INITIALIZATIONS.  IF
C       IOPT.EQ.1, THEN THE INPUT FILE IS ALSO INITIALIZED
C
C**********************************************************************
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C**********************************************************************
C
C     SET THE INPUT/OUTPUT UNIT NUMBERS
C
        JSTOPR=4
        JTERMI=5
        JTERMO=6
        JPRINT=7
        JCARDS=8
        JDUMMY=9
        JCMPAR=10
C
        JFIL11=11
        JFIL12=12
        JFIL13=13
        JFIL14=14
        JFIL15=15
        JFIL16=16
        JFIL17=17
        JFIL18=18
        JFIL19=19
        JFIL20=20
C
C     INITIALIZE GRAPHICS PACKAGE
C
        CALL GRINIT(JTERMI,JTERMO,TITLGR)
C
C     READ TITLE AND PRINT INPUT FILE
C
        IF(IOPT.EQ.1) CALL UTPAG1
C
C     INITIALIZE TIMER
C
        CALL UTWTIM(' ')
```

```
C
C     INITIALIZE BASED VARIABLE TABLE
C
        CALL UTBASE(0,0,0)
C
C     INITIALIZE THE STOP FLAG
C
        ISTOP=0
C
10      OPEN(UNIT=JSTOPR,ERR=10,STATUS='UNKNOWN')
C
        WRITE(JSTOPR,20) ISTOP
20      FORMAT(I10)
C
        CLOSE(UNIT=JSTOPR)
C
        RETURN
        END
```

UTINPC

```
        SUBROUTINE UTINPC(ITITLE,
     &
     &                    VALUE )
C
        INCLUDE '[ UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A CHARACTER VARIABLE
C
        CHARACTER*(*) ITITLE,VALUE
C
C*********************************************************************
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
        NCHAR=LEN(VALUE)
C
10      WRITE(JTERMO,20)ITITLE,NCHAR
20      FORMAT(' ENTER ',A,'  (',I2,' CHARACTERS MAX) >'$)
C
        READ(JTERMI,30,END=10) VALUE
30      FORMAT(A)
C
        RETURN
        END
```

UTINPF

```
      SUBROUTINE UTINPF(ITITLE,
     &
     &                  VALUE )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A REAL NUMBER, VALUE
C
      CHARACTER*(*) ITITLE
C
C*********************************************************************
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
10    WRITE(JTERMO,20)ITITLE
20    FORMAT(' ENTER ',A,5X,'(F20.0) >'$)
C
      READ(JTERMI,*,END=10,ERR=10) VALUE
C
      RETURN
      END
```

UTINPI

```
      SUBROUTINE UTINPI(ITITLE,
     &
     &                  IVALUE)
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN AN INTEGER NUMBER, IVALUE
C
      CHARACTER*(*) ITITLE
C
C*********************************************************************
C
      INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
10    WRITE(JTERMO,20)ITITLE
20    FORMAT(' ENTER ',A,' >'$)
C
      READ(JTERMI,*,END=10,ERR=10) IVALUE
C
```

```
      RETURN
      END
```

UTINTP

```
      SUBROUTINE UTINTP(X$,Y$,NPTS,ARG,
     &                  INTRVL,
     &                  ANS,DYDX      )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE LINEARLY INTERPOLATES ARG ONTO Y$=Y$(X$).
C
      DIMENSION X$(NPTS),Y$(NPTS)
C
C*****************************************************************************
C
C     DO NOT SEARCH IF INTERVAL IS GIVEN
C
      INT=INTRVL
      IF(INT.GT.0) GOTO 10
C
C     BINARY SEARCH FOR PROPER INTERVAL
C
      CALL UTBINS(X$,NPTS,ARG,INT)
C
C     CALCULATE VARIABLES FOR INTERVAL
C
10    XSTEP=X$(INT+1)-X$(INT)
      YSTEP=Y$(INT+1)-Y$(INT)
C
      DYDX=YSTEP/XSTEP
      ANS=Y$(INT)+(ARG-X$(INT))*DYDX
C
C     RETURN INTERVAL IF INTRVL.EQ.0
C
      IF(INTRVL.EQ.0) INTRVL=INT
      RETURN
      END
```

UTPAG1

```
      SUBROUTINE UTPAG1
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS THE INPUT FILE AND PRINTS
```

```
C        AN IMAGE OF IT
C
C*************************************************************************
C
         INCLUDE '[.UTILITY]IOUNIT.INC'
C
         INCLUDE '[.GRAFIC]GRCOMN.INC'
C
         CHARACTER*80 ICARD
C
         CHARACTER* 1 ISTAR
         DATA ISTAR /'*'/
C
C*************************************************************************
C
C     READ THE TITLE
C
         REWIND JCARDS
C
         READ(JCARDS,10,END=80) TITLGR
10       FORMAT(A)
C
C     REWIND AND PRINT IMAGE OF THE FILE
C
         CALL UTHEAD('IMAGE OF INPUT FILE')
C
         WRITE(JPRINT,20)
20       FORMAT(11X,'         1         2         3         4',
     &             '         5         6         7         8'/
     &        11X,'12345678901234567890123456789012345678901234567890',
     &            '12345678901234567890123456789012345678901234567890'/)
C
         REWIND JCARDS
         NCARD=1
C
30       READ(JCARDS,40,END=60) ICARD
40       FORMAT(A)
C
         WRITE(JPRINT,50) NCARD,ICARD
50       FORMAT(' CARD',I5,':',A)
         NCARD=NCARD+1
         GOTO 30
C
C     WRITE OUT COLUMN HEADINGS AGAIN
C
60       WRITE(JPRINT,20)
C
C     REWIND AND REPOSITION FILE AFTER TITLE AND COMMENTS
C
         REWIND JCARDS
         READ(JCARDS,40) ICARD
70       READ(JCARDS,40) ICARD
         IF(ICARD(1:1).EQ.ISTAR) GOTO 70
```

```
        BACKSPACE JCARDS
C
        RETURN
C
C     NO INPUT FILE EXISTS
C
80      WRITE(JTERMO,90)
90      FORMAT(' INPUT FILE DOES NOT EXIST')
C
        RETURN
        END
```

UTSMTH

```
        SUBROUTINE UTSMTH(NSMTH,NPTS,X$,
     &                    Y$
     &                                 )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PERFORMS NSMTH SIMPLE SMOOTHINGS OF THE
C        POINTS X$(NPTS),Y$(NPTS).  THE SMOOTHING FUNCTION IS JUST
C        A SIMPLE WEIGHTED AVERAGE.
C
        DIMENSION X$(NPTS),Y$(NPTS)
C
C*****************************************************************
C
        INCLUDE '[.UTILITY]UTCOMN.INC'
C
C*****************************************************************
C
C     GET BASED VARIABLES
C
        CALL UTBASE(+1,NPTS,IYNEW)
C
C     LOOP FOR MULTIPLE SMOOTHINGS
C
        DO 30 ISMTH=1,NSMTH
C
C     LOOP THROUGH ALL INTERIOR POINTS
C
        DO 10 I=2,NPTS-1
        VARUT(IYNEW+I)=0.5*((X$(I+1)-X$(I))*(Y$(I+1)+Y$(I  ))
     &                     -(X$(I-1)-X$(I))*(Y$(I-1)+Y$(I  )))
     &                                      /(X$(I+1)-X$(I-1))
10      CONTINUE
C
C     SAVE SMOOTHED VALUES (INTERIOR ONLY)
C
```

```
        DO 20 I=2,NPTS-1
        Y$(I)=VARUT(IYNEW+I)
20      CONTINUE
C
C     GO BACK FOR NEXT SMOOTHING
C
30      CONTINUE
C
C     RELEASE BASED VARIABLE
C
        CALL UTBASE(-1,0,IYNEW)
C
        RETURN
        END
```

UTSPLN

```
        SUBROUTINE UTSPLN(Y$,NPTS,ITYPE,
     &
     &                    DY$          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CALCULATE A CUBIC SPLINE WHOSE ORDINATE IS Y$
C        AND WHOSE ASCSISSA IS I (THE INDEX).  THIS ROUTINE STORES
C        THE FIRST DERIVATIVES IN DY$
C
C     IF ITYPE=0 PERIODIC CUBIC
C             =1 VANISHIND SECOND DERIVATIVE AT ENDS
C
        DIMENSION Y$(NPTS),DY$(NPTS)
C
C*************************************************************************
C
        INCLUDE '[.UTILITY]UTCOMN.INC'
C
C*************************************************************************
C
        IF(ITYPE.EQ.1) GOTO 40
C
C*****PERIODIC CUBIC SPLINE
C
C     GET BASED VARIABLES FOR TRIDIAGONAL MATRIX AND TRIDIAGONAL SOLVER
C
        CALL UTBASE(+1,NPTS,IP)
        CALL UTBASE(+1,NPTS,IQ)
        CALL UTBASE(+1,NPTS,IS)
        CALL UTBASE(+1,NPTS,IT)
        CALL UTBASE(+1,NPTS,IU)
        CALL UTBASE(+1,NPTS,IV)
```

```
C
C     FORWARD TRIDIAGONAL SWEEP
C
        VARUT(IP+1)= 4.00
        VARUT(IQ+1)=-0.25
        VARUT(IU+1)= 0.75*(Y$(2)-Y$(NPTS))
        VARUT(IS+1)=-0.25
C
        DO 10 I=2,NPTS-1
        VARUT(IP+I)=VARUT(IQ+I-1)+4.00
        VARUT(IQ+I)=-1.00/VARUT(IP+I)
        VARUT(IU+I)=(3.00*(Y$(I+1)-Y$(I-1))-VARUT(IU+I-1))/VARUT(IP+I)
        VARUT(IS+I)=-VARUT(IS+I-1)/VARUT(IP+I)
10      CONTINUE
C
        VARUT(IP+NPTS)=VARUT(IQ+NPTS-1)+4.00
        VARUT(IQ+NPTS)=-1.00/VARUT(IP+NPTS)
        VARUT(IU+NPTS)=(3.00*(Y$(1)-Y$(NPTS-1))-VARUT(IU+NPTS-1))
     &                                              /VARUT(IP+NPTS)
        VARUT(IS+NPTS)=-VARUT(IS+NPTS-1)/VARUT(IP+NPTS)
C
C     BACKWARD TRIDIAGONAL SWEEP
C
        VARUT(IT+NPTS)=1.00
        VARUT(IV+NPTS)=0.00
C
        DO 20 I=NPTS-1,1,-1
        VARUT(IT+I)=VARUT(IQ+I)*VARUT(IT+I+1)+VARUT(IS+I)
        VARUT(IV+I)=VARUT(IQ+I)*VARUT(IV+I+1)+VARUT(IU+I)
20      CONTINUE
C
C     CALCULATE THE LAST SLOPE
C
        DY$(NPTS)=(3.00*(Y$(1)-Y$(NPTS-1))-VARUT(IV+NPTS-1)-VARUT(IV+1))
     &          /(       4.00           +VARUT(IT+NPTS-1)+VARUT(IT+1))
C
C     CALCULATE THE REMAINING SLOPES
C
        DO 30 I=1,NPTS-1
        DY$(I)=VARUT(IT+I)*DY$(NPTS)+VARUT(IV+I)
30      CONTINUE
C
C     RELEASE BASED VARIABLES
C
        CALL UTBASE(-1,NPTS,IP)
        CALL UTBASE(-1,NPTS,IQ)
        CALL UTBASE(-1,NPTS,I3)
        CALL UTBASF(-1,NPTS,IT)
        CALL UTBASE(-1,NPTS,IU)
        CALL UTBASE(-1,NPTS,IV)
C
        RETURN
C
```

```
C*****VANISHIND SECOND DERIVATIVES AT ENDS
C
C     GET BASED VARIABLES FOR TRIDIAGONAL MATRIX AND TRIDIAGONAL SOLVER
C
40      CALL UTBASE(+1,NPTS,IP)
        CALL UTBASE(+1,NPTS,IQ)
        CALL UTBASE(+1,NPTS,IU)
C
C     FORWARD TRIDIAGONAL SWEEP
C
        VARUT(IP+1)= 2.00
        VARUT(IQ+1)=-0.50
        VARUT(IU+1)= 1.50*(Y$(2)-Y$(1))
C
        DO 50 I=2,NPTS-1
        VARUT(IP+I)=VARUT(IQ+I-1)+4.00
        VARUT(IQ+I)=-1.00/VARUT(IP+I)
        VARUT(IU+I)=(3.00*(Y$(I+1)-Y$(I-1))-VARUT(IU+I-1))/VARUT(IP+I)
50      CONTINUE
C
        VARUT(IP+NPTS)=VARUT(IQ+NPTS-1)+2.00
        VARUT(IQ+NPTS)=-1.00/VARUT(IP+NPTS)
        VARUT(IU+NPTS)=(3.00*(Y$(NPTS)-Y$(NPTS-1))-VARUT(IU+NPTS-1))
     &                                                /VARUT(IP+NPTS)
C
C     SOLVE FOR LAST SLOPE
C
        DY$(NPTS)=VARUT(IU+NPTS)
C
C     BACK-SUBSTITUTE FOR REMAINING SLOPES
C
        DO 60 I=NPTS-1,1,-1
        DY$(I)=VARUT(IQ+I)*DY$(I+1)+VARUT(IU+I)
60      CONTINUE
C
C     RELEASE BASED VARIABLES
C
        CALL UTBASE(-1,NPTS,IP)
        CALL UTBASE(-1,NPTS,IQ)
        CALL UTBASE(-1,NPTS,IU)
C
        RETURN
C
        END
```

UTSTOP

```
        FUNCTION UTSTOP(IDUM
     &
     &                  )
```

```
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE RETURNS THE STOP FLAG CONTAINED IN FILE JSTOPR
C
C*********************************************************************
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*********************************************************************
C
C     OPEN JSTOPR, READ THE STOP FLAG, AND CLOSE THE FILE
C
10      OPEN(UNIT=JSTOPR,ERR=10,STATUS='UNKNOWN')
C
        READ(JSTOPR,20) ISTOP
20      FORMAT(I10)
C
        IF(ISTOP.NE.0) THEN
           REWIND JSTOPR
           JSTOP=0
           WRITE(JSTOPR,20) JSTOP
        ENDIF
C
        CLOSE(UNIT=JSTOPR)
C
        UTSTOP=ISTOP
C
        RETURN
        END
```

UTTIME

```
        SUBROUTINE UTTIME(
     &
     &                    TIME)
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE LOOKS UP THE CURREENT CPU TIME IN SECONDS
C
C*********************************************************************
C
        INTEGER*4 ITMLST(4),IRETLN,ICPUTM,IDELTM(2)
        INTEGER*2 IWORD(2),ITIMBF(7)
        EQUIVALENCE (IWORD(1),ITMLST(1))
C
        DATA INIT /0/
C
C*********************************************************************
```

```
C
C     SET UP ITEM LISTS FOR CALL TO JOB/PROCESS INFO ROUTINES
C
        IWORD(1)=4
        IWORD(2)='0407'X
        ITMLST(2)=%LOC(ICPUTM)
        ITMLST(3)=%LOC(IRETLN)
        ITMLST(4)=0
C
C     CALL SYSTEM SERVICE ROUTINES
C
        CALL SYS$GETJPI(,,,ITMLST,,,)
        CALL LIB$EMUL(ICPUTM,-100000,0,IDELTM)
        CALL SYS$NUMTIM(ITIMBF,IDELTM)
C
C     COMPUTE TIME
C
        TIME=86400.00*ITIMBF(3)
     &       + 3600.00*ITIMBF(4)
     &       +   60.00*ITIMBF(5)
     &       +    1.00*ITIMBF(6)
     &       +    0.01*ITIMBF(7)
C
C
C     SUBTRACT OFF THE INITIAL TIME
C
        IF(INIT.EQ.0) THEN
           TINIT=TIME
           INIT =1
        ENDIF
C
        TIME=TIME-TINIT
C
        RETURN
        END
```

UTWTIM

```
        SUBROUTINE UTWTIM(ITEXT
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PRINTS A MESSAGE GIVEN BY ITEXT
C        AS WELL AS INCREMENTAL AND TOTAL CPU TIMES
C
        CHARACTER ITEXT*(*)
C
C*************************************************************************
```

```
C
        INCLUDE '[.UTILITY]IOUNIT.INC'
C
C*****************************************************************************
C
C     GET THE TIME
C
        CALL UTTIME(TIME)
C
C     CHECK IF INITIALIZATION
C
        IF(LEN(ITEXT).LE.1) THEN
           TSTART=TIME
           TSAVE=0.
           RETURN
        ENDIF
C
C     NORMAL PROCESSING
C
        TCUM =TIME-TSTART
        TINC =TCUM-TSAVE
        TSAVE=TCUM
C
        IF(ITEXT.EQ.'.RETURN.') THEN
           WRITE(ITEXT,10) TCUM
10         FORMAT(F8.2)
        ELSE
           WRITE(JPRINT,20) TINC,TCUM,ITEXT
20         FORMAT(' **TIMER** INCREMENTAL CPU TIME =',F8.2,' SEC      ',
     &             'TOTAL CPU TIME =',F9.2,' SEC',5X,A                  )
        ENDIF
C
        RETURN
        END
```

B.7 GRAFIC — Graphics Package

GRCOMN

```
C-----GRAFIC CONTROL PARAMETERS
C     XMINGR       VALUE OF FIRST ABSCISSA AXIS LABEL
C     XMAXGR       VALUE OF LAST  ABSCISSA AXIS LABEL
C     YMINGR       VALUE OF FIRST ORDINATE AXIS LABEL
C     YMAXGR       VALUE OF LAST  ORDINATE AXIS LABEL
C     XMNCGR       ABSCISSA VALUE FOR LEFT   EDGE OF PLOTTING AREA (CLIPPING)
C     XMXCGR       ABSCISSA VALUE FOR RIGHT  EDGE OF PLOTTING AREA (CLIPPING)
C     YMNCGR       ORDINATE VALUE FOR BOTTOM EDGE OF PLOTTING AREA (CLIPPING)
```

```
C     YMXCGR       ORDINATE VALUE FOR TOP    EDGE OF PLOTTING AREA (CLIPPING)
C     INO1GR       =0  USE PLOT SCALES IN COMMON
C                  =1  COMPUTE SCALES
C     INO2GR       =0  CALCULATED SCALES ARE INDEPENDENT
C                  =1  INCREMENT OF CALCULATED SCALES ARE EQUAL
C     INO3GR       =0  DO NOT DRAW AXES
C                  =1  DRAW AXES
C     INO4GR       =0  DO NOT DRAW BACKGROUND GRID
C                  =1  DRAW BACKGROUND GRID
C     INO5GR       =0  GENERATE HARDCOPY AND RETURN
C                  =1  DRAW INTERACTIVE IMAGE AND DISPLAY MENU
C     INO6GR       =0  INCH PLOT (8*6)
C                  =1  CENTIMETER PLOT (20*15)
C     INO7GR       NOT USED
C     INO8GR       NOT USED
C     INO9GR       NOT USED
C     IN10GR       NOT USED
C     IN11GR       NOT USED
C     IN12GR       NOT USED
C     JINGR        GRAPHIC DEVICE INPUT UNIT
C     JOUTGR       GRAPHIC DEVICE OUTPUT UNIT
C     JHRDGR       =0  PLOTTING IN INTERACTIVE MODE
C                  >0  UNIT NUMBER FOR QMS FILE (HARDCOPY)
C     IDEVGR       GRAPHIC DEVICE TYPE
C     IHRDGR       NOT USED
C     IDUMGR       NOT USED
C     JTRPGR       =.FALSE. TRAP NOT SPRING
C                  =.TRUE.  TRAP IS SPRUNG
C     IMODGR       =1 INTERACTIVE (PLOTLIB)
C                  =2 QMS OUTPUT (LANDSCAPE)
C                  =3 TEX/QMS (PORTRAIT)
C     XOLDGR       ABSCISSA OF TARGET OF PREVIOUS MOVE/DRAW
C     YOLDGR       ORDINATE OF TARGET OF PREVIOUS MOVE/DRAW
C     IOLDGR       =0  TARGET OF PREVIOUS MOVE/DRAW WAS OUT OF BOUNDS
C                  =1  TARGET OF PREVIOUS MOVE/DRAW WAS IN BOUNDS
C     ASCLGR       VALID SCALING INCREMENTS (/10**N)
C     NSCLGR       NUMBER OF VALID SCALING INCREMENTS
C     XSCLGR       DISTANCE (IN INCHES) BETWEEN X-AXIS TICKS
C     YSCLGR       DISTANCE (IN INCHES) BETWEEN Y-AXIS TICKS
C     NXINGR       NUMBER OF X-AXIS INTERVALS (TICKS-1)
C     NYINGR       NUMBER OF Y-AXIS INTERVALS (TICKS-1)
C     TEXSGR       SCALE FACTOR FOR TEX OUTPUT
C     TITLGR       TITLE TO BE PUT ON TOP OF ALL PLOTS
C     DATEGR       DATE OF LAST CALL TO GRINIT OR GRTIME
C     TIMEGR       TIME OF LAST CALL TO GRINIT OR GRTIME
C
      LOGICAL*4 JTRPGR
C
      COMMON /GRCOMN/ XMINGR,XMAXGR,YMINGR,YMAXGR,
     &                XMNCGR,XMXCGR,YMNCGR,YMXCGR,
     &                INO1GR,INO2GR,INO3GR,INO4GR,INO5GR,INO6GR,
     &                INO7GR,INO8GR,INO9GR,IN10GR,IN11GR,IN12GR,
     &                JINGR ,JOUTGR,JHRDGR,IDEVGR,IHRDGR,IDUMGR,
```

```
     &                  JTRPGR,IMODGR,
     &                  XOLDGR,YOLDGR,IOLDGR,
     &                  ASCLGR(11),NSCLGR,
     &                  XSCLGR,YSCLGR,NXINGR,NYINGR,TEXSGR
C
        COMMON /GRTITL/ TITLGR,DATEGR,TIMEGR
        CHARACTER*80 TITLGR
        CHARACTER*9  DATEGR
        CHARACTER*8  TIMEGR
```

GRANOT

```
        SUBROUTINE GRANOT(TEXT
     &
     &                        )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE WRITES ANNOTATION ON A PLOT
C
        CHARACTER*(*) TEXT
C
C*************************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*************************************************************************
C
        IF(IOLDGR.NE.1) RETURN
C
        XWID=XSCLGR*NXINGR
        YWID=YSCLGR*NYINGR
C
        IF(IMODGR.EQ.1) THEN
           XX=0.5+XWID*(XOLDGR-XMINGR)/(XMAXGR-XMINGR)
           YY=0.5+YWID*(YOLDGR-YMINGR)/(YMAXGR-YMINGR)
           CALL SYMBOL(XX,YY,0.10,TEXT,0.0,LEN(TEXT))
        ELSEIF(IMODGR.EQ.2) THEN
           IXX=1050.0+1000.0*YWID*(YOLDGR-YMINGR)/(YMAXGR-YMINGR)
           IYY=1600.0+1000.0*XWID*(XOLDGR-XMINGR)/(XMAXGR-XMINGR)
           WRITE(JHRDGR,10) IYY,IXX,TEXT
10         FORMAT(' ^IGE'/
     &            ' ^IV',I5.5,'^IH',I5.5,'^SV00204',A,'^G'/
     &            ' ^IGV'                                  )
        ELSEIF(IMODGR.EQ.3) THEN
           XX=YWID*(XOLDGR-XMINGR)/(XMAXGR-XMINGR)
           YY=YWID*(YOLDGR-YMINGR)/(YMAXGR-YMINGR)
           WRITE(JHRDGR+1,20) XX,YY,TEXT
20         FORMAT(' \put(',F8.4,',',F8.4,'){\makebox(0,0)[bl]{',A,'}}')
        ENDIF
```

```
C
      RETURN
      END
```

GRAPHC

```
      SUBROUTINE GRAPHC(GRPKG,INDGR,PLTITL,
     &                  X$,Y$,NPTS,GRDUMY
     &                                    )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CONTAINS THE CONTROL LOGIC FOR GRAPHICS
C
C     IND VALUE                 =0                =1
C      01    1        SCALES IN COMMON       CALCULATE SCALES
C      02    2        INDEPENDENT SCALING    DEPENDENT SCALING
C      03    4                               DRAW AXES
C      04    8                               DRAW GRID
C      05   16        AUTO-HARD COPY         USE MENU
C      06   32        INCHES                 CENTIMETERS
C
C     GRPKG IS A USER SUPPLIED PLOTTING PACKAGE WHICH CREATES
C        AN IMAGE BY SUITABLE CALLS TO GRDRAW, GRMOVE, AND GRANOT
C        THE CALLING SEQUENCE IS:
C            CALL GRPKG(X$,Y$,NPTS,GRDUMY)
C
      DIMENSION X$(NPTS),Y$(NPTS),GRDUMY(1)
C
      CHARACTER*(*) PLTITL
C
C*********************************************************************
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
      DIMENSION    XPUT(100),YPUT(100),IPUT(100)
      DIMENSION    XANOT(20),YANOT(20)
      CHARACTER*40 CANOT(20)
      CHARACTER*40 OUTFIL
      CHARACTER*1  JCHAR
C
C*********************************************************************
C
C     INITIALLY THERE IS NO ANNOTATION AND NO PUTS
C
      NANOT=0
      NPUT =0
C
C     CALCULATE INDICATORS ONLY IF INDGR.GT.0
C
```

```
10      IF(INDGR.LE.0) GOTO 20
C
        INDT=INDGR
C
        INO6GR=INDT/32
        INDT=INDT-INO6GR*32
C
        INO5GR=INDT/16
        INDT=INDT-INO5GR*16
C
        INO4GR=INDT/8
        INDT=INDT-INO4GR*8
C
        INO3GR=INDT/4
        INDT=INDT-INO3GR*4
C
        INO2GR=INDT/2
        INDT=INDT-INO2GR*2
C
        INO1GR=INDT
C
C     SAVE INDICATOR FOR SCALING
C
20      INDS=INO1GR
C
C     SET UP PLOT FORMAT FACTORS
C
        IF(INO6GR.EQ.0) THEN
           XSCLGR=1.000000
           YSCLGR=1.000000
           NXINGR=8
           NYINGR=6
           TEXSGR=0.500000
        ELSE
           XSCLGR=1.968504
           YSCLGR=1.968504
           NXINGR=4
           NYINGR=3
           TEXSGR=0.500000
        ENDIF
C
C     CALCULATE MINIMUMS AND MAXIMUMS TO PLOT (FULL DATA)
C
30      XMIN=X$(1)
        XMAX=XMIN
        YMIN=Y$(1)
        YMAX=YMIN
C
        DO 40 I=2,NPTS
        IF(X$(I).LT.XMIN) XMIN=X$(I)
        IF(X$(I).GT.XMAX) XMAX=X$(I)
        IF(Y$(I).LT.YMIN) YMIN=Y$(I)
        IF(Y$(I).GT.YMAX) YMAX=Y$(I)
```

```
40      CONTINUE
C
C     OPEN THE APPROPRIATE GRAPHICS MODE
C
        IF(INOSGR.EQ.1) THEN
           CALL GRMODE(1)
        ELSE
           GOTO 1300
        ENDIF
C
C     ERASE SCREEN
C
50      CALL GRMODE(0)
C
C     DRAW AXES AND SCALE
C
        CALL GRAXES(PLTITL,XMIN,XMAX,YMIN,YMAX,INDS)
C
C     PLOT WITH EXTERNALLY WRITTEN PLOTTER
C
60      CALL PLOTON
        CALL GRPKG(X$,Y$,NPTS,GRDUMY)
        CALL PLOTOF
C
        DO 70 IANOT=1,NANOT
        CALL GRMOVE(XANOT(IANOT),YANOT(IANOT),0)
        CALL GRANOT(CANOT(IANOT))
70      CONTINUE
C
        DO 80 JPUT=1,NPUT
        IF(IPUT(JPUT).EQ.-1) THEN
           CALL GRDRAW(XPUT(JPUT),YPUT(JPUT),          0)
        ELSE
           CALL GRMOVE(XPUT(JPUT),YPUT(JPUT),IPUT(JPUT))
        ENDIF
80      CONTINUE
C
C       MENU FOR VT125
C
100     CALL GRTRAP
        CALL GRCURS('A B L M O P Q T V W X',JCHAR,XCURSG,YCURSG)
C
        IF(JCHAR.EQ.'A' .OR. JCHAR.EQ.'a') GOTO  200
        IF(JCHAR.EQ.'B' .OR. JCHAR.EQ.'b') GOTO  300
        IF(JCHAR.EQ.'L' .OR. JCHAR.EQ.'l') GOTO 1300
        IF(JCHAR.EQ.'M' .OR. JCHAR.EQ.'m') GOTO 1400
        IF(JCHAR.EQ.'O' .OR. JCHAR.EQ.'o') GOTO 1600
        IF(JCHAR.EQ.'P' .OR. JCHAR.EQ.'p') GOTO 1700
        IF(JCHAR.EQ.'Q' .OR. JCHAR.EQ.'q') GOTO 1800
        IF(JCHAR.EQ.'T' .OR. JCHAR.EQ.'t') GOTO 2100
        IF(JCHAR.EQ.'V' .OR. JCHAR.EQ.'v') GOTO 2300
        IF(JCHAR.EQ.'W' .OR. JCHAR.EQ.'w') GOTO 2400
        IF(JCHAR.EQ.'X' .OR. JCHAR.EQ.'x') GOTO 2500
```

```
        GOTO 100
C
C*****A*****ANNOTATE
C
200     IF(NANOT.EQ.20) THEN
           WRITE(JOUTGR,210)
210        FORMAT(' ONLY 20 ANNOTATIONS ALLOWED  ')
           GOTO 100
        ENDIF
C
        NANOT=NANOT+1
        XANOT(NANOT)=XCURSG
        YANOT(NANOT)=YCURSG
        CALL GRINPA(' DESIRED ANNOTATION',CANOT(NANOT))
C
        CALL GRMOVE(XANOT(NANOT),YANOT(NANOT),0)
        CALL GRANOT(CANOT(NANOT))
C
        GOTO 100
C
C*****B*****BLOWUP
C
300     CALL GRINPF('XMIN',XMIN)
        CALL GRINPF('XMAX',XMAX)
        CALL GRINPF('YMIN',YMIN)
        CALL GRINPF('YMAX',YMAX)
C
        IF(ABS(XMIN-XMAX).LT.1.0E-25) GOTO 100
        IF(ABS(YMIN-YMAX).LT.1.0E-25) GOTO 100
C
C     RE-SCALE AND RE-PLOT BASED ON ABOVE VALUES (DO NOT NORMALIZE
C        SCALES)
C
        CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
        INDS=0
        GOTO 50
C
C*****L*****LASERGRAFIX MODE
C
1300    CALL GRMODE(2)
C
        CALL GRAXES(PLTITL,XMIN,XMAX,YMIN,YMAX,INDS)
        CALL GRPKG(X$,Y$,NPTS,GRDUMY)
C
        DO 1350 IANOT=1,NANOT
        CALL GRMOVE(XANOT(IANOT),YANOT(IANOT),0)
        CALL GRANOT(CANOT(IANOT))
1350    CONTINUE
C
        DO 1360 JPUT=1,NPUT
        IF(IPUT(JPUT).EQ.-1) THEN
           CALL GRDPAN(XPUT(JPUT),YPUT(JPUT),          0)
        ELSE
```

```
            CALL GRMOVE(XPUT(JPUT),YPUT(JPUT),IPUT(JPUT))
          ENDIF
1360      CONTINUE
C
          CALL GRMODE(1)
C
C       RETURN IF IN AUTO-HARD COPY MODE
C
          IF(INOSGR.EQ.0) RETURN
C
          WRITE(JOUTGR,1370)
1370      FORMAT(' HARD COPY SENT TO * QMS *')
C
          GOTO 100
C
C*****M*****MIN/MAX
C
C       CALCULATE MINIMUMS AND MAXIMUMS TO PLOT (FULL DATA)
C
1400      XMIN=X$(1)
          XMAX=XMIN
          YMIN=Y$(1)
          YMAX=YMIN
C
          DO 1410 I=2,NPTS
          IF(X$(I).LT.XMIN) XMIN=X$(I)
          IF(X$(I).GT.XMAX) XMAX=X$(I)
          IF(Y$(I).LT.YMIN) YMIN=Y$(I)
          IF(Y$(I).GT.YMAX) YMAX=Y$(I)
1410      CONTINUE
C
          WRITE(JOUTGR,1420) XMIN,XMAX,YMIN,YMAX
1420      FORMAT(' MINIMUM AND MAXIMUM DATA VALUES:'/
     &           10X,'XMIN=',G15.7,10X,'XMAX=',G15.7/
     &           10X,'YMIN=',G15.7,10X,'YMAX=',G15.7/)
C
          GOTO 100
C
C*****O*****ORIGINAL SCALES
C
1600      INDS=1
          GOTO 30
C
C*****P*****PUT
C
1700      IF(NPUT.EQ.100) THEN
             WRITE(JOUTGR,1710)
1710         FORMAT(' ONLY 100 PUTS ANNOTATIONS ALLOWED  ')
             GOTO 100
          ENDIF
C
          NPUT=NPUT+1
          XPUT(NPUT)=XCURSG
```

```
        YPUT(NPUT)=YCURSG
        CALL GRINPI(' SYMBOL NUMBER OR -1 FOR DRAW',IPUT(NPUT))
C
        IF(IPUT(NPUT).EQ.-1) THEN
           CALL GRDRAW(XPUT(NPUT),YPUT(NPUT),          0)
        ELSE
           CALL GRMOVE(XPUT(NPUT),YPUT(NPUT),IPUT(NPUT))
        ENDIF
C
        GOTO 100
C
C*****Q*****QUERY
C
1800    INDS=0
C
1805    WRITE(JOUTGR,1810) INO1GR,INO2GR,INO3GR,INO4GR,
     &                     XSCLGR,YSCLGR,NXINGR,NYINGR
1810    FORMAT('  1. CALC SCALES   ',I5   /
     &         '  2. DPEN SCALES   ',I5   /
     &         '  3. DRAW AXES     ',I5   /
     &         '  4. DRAW GRID     ',I5   /
     &         ' 11. XSCLGR        ',F10.5/
     &         ' 12. YSCLGR        ',F10.5/
     &         ' 13. NXINGR        ',I5   /
     &         ' 14. NYINGR        ',I5   )
C
        CALL GRINPI('OPTION NUMBER',IOPT)
C
        IF(IOPT.EQ.11 .OR. IOPT.EQ.12 .OR.
     &     IOPT.EQ.13 .OR. IOPT.EQ.14     ) INDS=1
C
        IF(IOPT.EQ. 1) CALL GRINPI('INO1GR',INO1GR)
        IF(IOPT.EQ. 2) CALL GRINPI('INO2GR',INO2GR)
        IF(IOPT.EQ. 3) CALL GRINPI('INO3GR',INO3GR)
        IF(IOPT.EQ. 4) CALL GRINPI('INO4GR',INO4GR)
        IF(IOPT.EQ.11) CALL GRINPF('XSCLGR',XSCLGR)
        IF(IOPT.EQ.12) CALL GRINPF('YSCLGR',YSCLGR)
        IF(IOPT.EQ.13) CALL GRINPI('NXINGR',NXINGR)
        IF(IOPT.EQ.14) CALL GRINPI('NYINGR',NYINGR)
C
C     EXIT OUT OF QUERY
C
        IF(IOPT.EQ.0) GOTO 50
C
C     ELSE RETURN FOR ANOTHER QUERY
C
        GOTO 1805
C
C*****T*****TEX OUTPUT MODE
C
2100    CALL GRMODE(3)
C
        CALL GRAXES(PLTITL,XMIN,XMAX,YMIN,YMAX,INDS)
```

```
        CALL GRPKG(X$,Y$,NPTS,GRDUMY)
C
        DO 2150 IANOT=1,NANOT
        CALL GRMOVE(XANOT(IANOT),YANOT(IANOT),0)
        CALL GRANOT(CANOT(IANOT))
2150    CONTINUE
C
        DO 2160 JPUT=1,NPUT
        IF(IPUT(JPUT).EQ.-1) THEN
           CALL GRDRAW(XPUT(JPUT),YPUT(JPUT),          0)
        ELSE
           CALL GRMOVE(XPUT(JPUT),YPUT(JPUT),IPUT(JPUT))
        ENDIF
2160    CONTINUE
C
        CALL GRMODE(1)
C
        GOTO 100
C
C*****V*****VALUE
C
2300    CALL GRMOVE(XCURSG,YCURSG,2)
C
        WRITE(JOUTGR,2310) XCURSG,YCURSG
2310    FORMAT(' CURSOR LOCATION: X ',G14.7/
     &          '                 Y ',G14.7)
C
        GOTO 100
C
C*****W*****WINDOW
C
C     GET OTHER CORNER OF WINDOW
C
2400    CALL GRCURS('OPPOSITE CORNER',JCHAR,XNEW,YNEW)
C
        XMIN=MIN(XNEW,XCURSG)
        XMAX=MAX(XNEW,XCURSG)
        YMIN=MIN(YNEW,YCURSG)
        YMAX=MAX(YNEW,YCURSG)
C
        IF(ABS(XMIN-XMAX).LT.1.0E-25) GOTO 100
        IF(ABS(YMIN-YMAX).LT.1.0E-25) GOTO 100
C
C     RESCALE AND REPLOT WITH ABOVE WINDOW
C
        CALL GRSSET(XMIN,XMAX,YMIN,YMAX)
C
        INDS=0
        GOTO 50
C
C*****X*****EXIT
C
C     CLOSE THE GRAPHICS BEFORE EXITING
```

```
C
2500    CALL GRMODE(0)
C
        RETURN
C
        END
```

GRAXES

```
        SUBROUTINE GRAXES(PLTITL,XMIND,XMAXD,YMIND,YMAXD,IND
     &
     &                                                    )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CALCULATES SCALES AND DRAWS AXES
C
        CHARACTER*(*) PLTITL
C
C*********************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
        CHARACTER*13 NUM
C
C*********************************************************************
C
C     CALCULATE SCALES IF IND=1
C
        IF(IND.NE.1) GOTO 30
10      CALL GRSCAL(XMIND,XMAXD,REAL(NXINGR),XMINGR,XSTPGR)
        CALL GRSCAL(YMIND,YMAXD,REAL(NYINGR),YMINGR,YSTPGR)
C
C     SCALE DEPENDENTLY IF INO2GR=1
C
        IF(INO2GR.NE.1) GOTO 20
        IF(XSTPGR.LT.YSTPGR) XSTPGR=YSTPGR
        IF(XSTPGR.GT.YSTPGR) YSTPGR=XSTPGR
C
20      XMAXGR=XMINGR+NXINGR*XSTPGR
        YMAXGR=YMINGR+NYINGR*YSTPGR
C
C     SET UP CLIPPING PARAMETERS
C
30      XMNCGR=XMINGR
        XMXCGR=XMAXGR
        YMNCGR=YMINGR
        YMXCGR=YMAXGR
C
        XSTPGR=(XMAXGR-XMINGR)/NXINGR
```

```
        YSTPGR=(YMAXGR-YMINGR)/NYINGR
C
        IF(XSTPGR.EQ.0. .OR. YSTPGR.EQ.0.) GOTO 10
C
C     RETURN IF AXES ARE NOT DRAWN
C
        IF(INO3GR.NE.1) RETURN
C
C     DRAW THE AXES AND LABELS
C
        XWID=XSCLGR*NXINGR
        YWID=YSCLGR*NYINGR
C
        IF(IMODGR.EQ.1) THEN
C
           CALL PLOT(0.5+XWID,0.5     ,3)
           CALL PLOT(0.5     ,0.5     ,2)
           CALL PLOT(0.5     ,0.5+YWID,2)
C
           DO 40 ITIC=0,NXINGR
           XX=0.500+ITIC*XSCLGR
           CALL PLOT(XX,0.500,3)
           CALL PLOT(XX,0.400,2)
           CALL GRPRNT((XMINGR+XSTPGR*ITIC),XSTPGR,NUM)
           CALL SYMBOL(XX-0.455,0.300,0.07,NUM,0.0,13)
40         CONTINUE
C
           DO 50 ITIC=0,NYINGR
           YY=0.500+ITIC*YSCLGR
           CALL PLOT(0.500,YY,3)
           CALL PLOT(0.400,YY,2)
           CALL GRPRNT((YMINGR+YSTPGR*ITIC),YSTPGR,NUM)
           CALL SYMBOL(0.300,YY-0.455,0.07,NUM,90.0,13)
50         CONTINUE
C
           XX=-0.56+0.50*(1.0+XWID)
           YY=-0.56+0.50*(1.0+YWID)
           CALL SYMBOL(XX,0.100,0.14,PLTITL(1: 3), 0.0,8)
           CALL SYMBOL(0.200,YY,0.14,PLTITL(9:16),90.0,8)
C
           KLEN=LEN(PLTITL)-16
           CALL SYMBOL(1.0,6.6,0.14,PLTITL(17:),0.0,KLEN)
           CALL SYMBOL(1.0,6.8,0.14,TITLGR     ,0.0,80  )
C
           CALL SYMBOL(6.0 ,0.1 ,0.14,DATEGR,0.0,9)
           CALL SYMBOL(999.,999.,0.14,' '    ,0.0,1)
           CALL SYMBOL(999.,999.,0.14,TIMEGR,0.0,8)
C
        ELSEIF(IMODGR.EQ.2) THEN
C
           IXX=XWID*1000.
           IYY=YWID*1000.
           WRITE(JHRDGR,60) IYY,IXX
```

```
60        FORMAT(' ^PW09^U',I5.5,':00000^D00000:00000^D00000:',I5.5/
     &           ' ^PW05'                                          )
C
          DO 80 ITIC=0,NXINGR
          IXX =ITIC*XSCLGR*1000
          IXX2=IXX+1100
          CALL GRPRNT((XMINGR+XSTPGR*ITIC),XSTPGR,NUM)
          WRITE(JHRDGR,70) IXX,IXX2,NUM
70        FORMAT(' ^U00000:',I5.5,'^D-00050:+00000^IGE^IV',I5.5,
     &           '^IH00650^SV00204',A,'^G^IGV'                    )
80        CONTINUE
C
          DO 100 ITIC=0,NYINGR
          IYY =ITIC*YSCLGR*1000
          IYY2=IYY+550
          CALL GRPRNT((YMINGR+YSTPGR*ITIC),YSTPGR,NUM)
          WRITE(JHRDGR,90) IYY,IYY2,NUM
90        FORMAT(' ^U',I5.5,':00000^D+00000:-00050^IGE^IV01400',
     &           '^IH',I5.5,'^SM00204',A,'^G^IGV'                 )
100       CONTINUE
C
          WRITE(JHRDGR,110) PLTITL( 1:8 ),
     &                      PLTITL( 9:16),
     &                      PLTITL(17:  ),
     &                      TITLGR,
     &                      DATEGR,TIMEGR
110       FORMAT(' ^IGE^IV05200^IH00500^SV00204',A,'^G'/
     &           '     ^IV01250^IH03650^SM00204',A,'^G'/
     &           '     ^IV01600^IH07100^SV00204',A,'^G'/
     &           '     ^IV01600^IH07300^SV00204',A,'^G'/
     &           '     ^IV07600^IH00500^SV00204',A,1X,A,'^G^IGV')
C
        ELSEIF(IMODGR.EQ.3) THEN
C
          XWID=XSCLGR*NXINGR
          YWID=YSCLGR*NYINGR
          WRITE(JHRDGR+1,120) XWID,YWID
120       FORMAT(' \put(0,0){\line(1,0){',F6.4,'}}'/
     &           ' \put(0,0){\line(0,1){',F6.4,'}}')
C
          DO 140 ITIC=0,NXINGR
          CALL GRPRNT((XMINGR+XSTPGR*ITIC),XSTPGR,NUM)
          WRITE(JHRDGR+1,130) (XSCLGR*ITIC),(0.05/TEXSGR),
     &                        (XSCLGR*ITIC),(-0.07/TEXSGR),NUM
130       FORMAT(' \put(',F8.4,',0){\line(0,-1){',F5.3,'}}'          /
     &           ' \put(',F8.4,',',F5.3,'){\makebox(0,0)[t]{$',A,'$}}')
140       CONTINUE
C
          DO 160 ITIC=0,NYINGR-1
          CALL GRPRNT((YMINGR+YSTPGR*ITIC),YSTPGR,NUM)
          WRITE(JHRDGR+1,150) (YSCLGR*ITIC),(0.05/TEXSGR),
     &                        (-0.07/TEXSGR),(YSCLGR*ITIC),NUM
150       FORMAT(' \put(0,',F8.4,'){\line(-1,0){',F5.3,'}}'          /
```

```
     &            ' \put(',F5.3,',',F8.4,'){\makebox(0,0)[r]{$',A,'$}}')
160        CONTINUE
           CALL GRPRNT(YMAXGR,YSTPGR,NUM)
           WRITE(JHRDGR+1,165) (YSCLGR*NYINGR),(0.05/TEXSGR),
     &                         (-0.07/TEXSGR),(YSCLGR*NYINGR),
     &                          PLTITL(9:16),NUM
165        FORMAT(' \put(0,',F8.4,'){\line(-1,0){',F5.3,'}}'           /
     &            ' \put(',F5.3,',',F8.4,'){\makebox(0,0)[r]{'          ,
     &                                        A,' \ \ $',A,'$}}')
C
           WRITE(JHRDGR+1,170) (XWID/2.00),(-0.2/TEXSGR),PLTITL( 1: 8),
     &                         (YWID+0.05/TEXSGR),TITLGR,
     &                         (YWID+0.20/TEXSGR),PLTITL(17:  )
170        FORMAT(' \put(',F8.4,',',F5.3,'){\makebox(0,0){',A,'}}'/
     &            ' \put(0.0,',F8.4,'){\makebox(0,0)[bl]{',A,'}}'/
     &            ' \put(0.0,',F8.4,'){\makebox(0,0)[bl]{',A,'}}')
C
           RETURN
C
         ENDIF
C
C     PLOT GRID IF DESIRED
C
         IF(IN04GR.NE.991) RETURN
C
         IF(IMODGR.EQ.1) THEN
            IF(IN06GR.EQ.0) THEN
               CALL GRID(0.5,0.5, 8,1.000000, 6,1.000000,'8080'X)
            ELSE
               CALL GRID(0.5,0.5,20,0.393701,15,0.393701,'8000'X)
               CALL GRID(0.5,0.5, 4,1.968504, 3,1.968504,'8080'X)
            ENDIF
         ELSE
            IF(IN06GR.EQ.0) THEN
               WRITE(JHRDGR,9110)
9110           FORMAT(' ^V2^PW01'/
     &                ' ^U01000:00000^D+00000:+08000'/
     &                ' ^U02000:00000^D+00000:+08000'/
     &                ' ^U03000:00000^D+00000:+08000'/
     &                ' ^U04000:00000^D+00000:+08000'/
     &                ' ^U05000:00000^D+00000:+08000'/
     &                ' ^U06000:00000^D+00000:+08000'/
     &                ' ^U00000:01000^D+06000:+00000'/
     &                ' ^U00000:02000^D+06000:+00000'/
     &                ' ^U00000:03000^D+06000:+00000'/
     &                ' ^U00000:04000^D+06000:+00000'/
     &                ' ^U00000:05000^D+06000:+00000'/
     &                ' ^U00000:06000^D+06000:+00000'/
     &                ' ^U00000:07000^D+06000:+00000'/
     &                ' ^U00000:08000^D+06000:+00000'/
     &                ' ^V0^PW03')
            ELSE
               WRITE(JHRDGR,9120)
```

```
            WRITE(JHRDGR,9130)
9120        FORMAT(' ^PW01'/
     &              ' ^V2^U00394:00000^D+00000:+07874'/
     &              ' ^V2^U00787:00000^D+00000:+07874'/
     &              ' ^V2^U01181:00000^D+00000:+07874'/
     &              ' ^V2^U01575:00000^D+00000:+07874'/
     &              ' ^V3^U01969:00000^D+00000:+07874'/
     &              ' ^V2^U02362:00000^D+00000:+07874'/
     &              ' ^V2^U02756:00000^D+00000:+07874'/
     &              ' ^V2^U03150:00000^D+00000:+07874'/
     &              ' ^V2^U03543:00000^D+00000:+07874'/
     &              ' ^V3^U03937:00000^D+00000:+07874'/
     &              ' ^V2^U04331:00000^D+00000:+07874'/
     &              ' ^V2^U04724:00000^D+00000:+07874'/
     &              ' ^V2^U05118:00000^D+00000:+07874'/
     &              ' ^V2^U05512:00000^D+00000:+07874'/
     &              ' ^V3^U05907:00000^D+00000:+07874')
9130        FORMAT(' ^V2^U00000:00394^D+05907:+00000'/
     &              ' ^V2^U00000:00787^D+05907:+00000'/
     &              ' ^V2^U00000:01181^D+05907:+00000'/
     &              ' ^V2^U00000:01575^D+05907:+00000'/
     &              ' ^V3^U00000:01969^D+05907:+00000'/
     &              ' ^V2^U00000:02362^D+05907:+00000'/
     &              ' ^V2^U00000:02756^D+05907:+00000'/
     &              ' ^V2^U00000:03150^D+05907:+00000'/
     &              ' ^V2^U00000:03543^D+05907:+00000'/
     &              ' ^V3^U00000:03937^D+05907:+00000'/
     &              ' ^V2^U00000:04331^D+05907:+00000'/
     &              ' ^V2^U00000:04724^D+05907:+00000'/
     &              ' ^V2^U00000:05118^D+05907:+00000'/
     &              ' ^V2^U00000:05512^D+05907:+00000'/
     &              ' ^V3^U00000:05906^D+05907:+00000'/
     &              ' ^V2^U00000:06299^D+05907:+00000'/
     &              ' ^V2^U00000:06693^D+05907:+00000'/
     &              ' ^V2^U00000:07087^D+05907:+00000'/
     &              ' ^V2^U00000:07480^D+05907:+00000'/
     &              ' ^V3^U00000:07876^D+05907:+00000'/
     &              ' ^V0^PW03')
          ENDIF
        ENDIF
C
        RETURN
        END
```

GRCTRI

```
        SUBROUTINE GRCTRI(X1,Y1,F1,X2,Y2,F2,X3,Y3,F3,FCONT
     &
     &                                                    )
C
```

```
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE DRAWS CONTOURS FOR A TRIANGLE
C
C*************************************************************************
C
C     BRANCH OUT, DEPENDING ON THE TOPOLOGY OF THE COUTOUR
C
        IF(F1-FCONT)  10, 50, 90
C
10      IF(F2-FCONT)  20, 30, 40
20      IF(F3-FCONT) 290,290,270
30      IF(F3-FCONT) 290,210,240
40      IF(F3-FCONT) 260,250,280
C
50      IF(F2-FCONT)  60, 70, 80
60      IF(F3-FCONT) 290,220,290
70      IF(F3-FCONT) 200,290,200
80      IF(F3-FCONT) 230,220,230
C
90      IF(F2-FCONT) 100,110,120
100     IF(F3-FCONT) 280,250,260
110     IF(F3-FCONT) 240,210,290
120     IF(F3-FCONT) 270,290,290
C
C     CONTOUR THROUGH 1 AND 2
C
200     CALL GRMOVE(X1,Y1,0)
        CALL GRDRAW(X2,Y2,0)
        RETURN
C
C     CONTOUR THROUGH 2 AND 3
C
210     CALL GRMOVE(X2,Y2,0)
        CALL GRDRAW(X3,Y3,0)
        RETURN
C
C     CONTOUR THROUGH 3 AND 1
C
220     CALL GRMOVE(X3,Y3,0)
        CALL GRDRAW(X1,Y1,0)
        RETURN
C
C     CONTOUR THROUGH 1 AND 2-3
C
230     CALL GRMOVE(X1,Y1,0)
        XX=X2+(FCONT-F2)*(X3-X2)/(F3-F2)
        YY=Y2+(FCONT-F2)*(Y3-Y2)/(F3-F2)
        CALL GRDRAW(XX,YY,0)
        RETURN
C
C     CONTOUR THROUGH 2 AND 3-1
C
```

```
240     CALL GRMOVE(X2,Y2,0)
        XX=X3+(FCONT-F3)*(X1-X3)/(F1-F3)
        YY=Y3+(FCONT-F3)*(Y1-Y3)/(F1-F3)
        CALL GRDRAW(XX,YY,0)
        RETURN
C
C     CONTOUR THROUGH 3 AND 1-2
C
250     CALL GRMOVE(X3,Y3,0)
        XX=X1+(FCONT-F1)*(X2-X1)/(F2-F1)
        YY=Y1+(FCONT-F1)*(Y2-Y1)/(F2-F1)
        CALL GRDRAW(XX,YY,0)
        RETURN
C
C     CONTOUR THROUGH 1-2 AND 2-3
C
260     XX=X1+(FCONT-F1)*(X2-X1)/(F2-F1)
        YY=Y1+(FCONT-F1)*(Y2-Y1)/(F2-F1)
        CALL GRMOVE(XX,YY,0)
        XX=X2+(FCONT-F2)*(X3-X2)/(F3-F2)
        YY=Y2+(FCONT-F2)*(Y3-Y2)/(F3-F2)
        CALL GRDRAW(XX,YY,0)
        RETURN
C
C     CONTOUR THROUGH 2-3 AND 3-1
C
270     XX=X2+(FCONT-F2)*(X3-X2)/(F3-F2)
        YY=Y2+(FCONT-F2)*(Y3-Y2)/(F3-F2)
        CALL GRMOVE(XX,YY,0)
        XX=X3+(FCONT-F3)*(X1-X3)/(F1-F3)
        YY=Y3+(FCONT-F3)*(Y1-Y3)/(F1-F3)
        CALL GRDRAW(XX,YY,0)
        RETURN
C
C     CONTOUR THROUGH 3-1 AND 1-2
C
280     XX=X3+(FCONT-F3)*(X1-X3)/(F1-F3)
        YY=Y3+(FCONT-F3)*(Y1-Y3)/(F1-F3)
        CALL GRMOVE(XX,YY,0)
        XX=X1+(FCONT-F1)*(X2-X1)/(F2-F1)
        YY=Y1+(FCONT-F1)*(Y2-Y1)/(F2-F1)
        CALL GRDRAW(XX,YY,0)
        RETURN
C
C     NO CONTOUR
C
290     RETURN
C
        END
```

GRCURS

```
      SUBROUTINE GRCURS(ITITLE,
     &
     &                  JCHAR,X,Y)
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE DISPLAYS THE CURSORS AND READS THE LOCATION
C       AND KEY PRESSED
C
      CHARACTER ITITLE*(*),JCHAR*1
C
C*********************************************************************
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
C     USE GRAPHICS INPUT ROUTINE IN PLOTLIB
C
      CALL GIN(ITITLE,JCHAR,XX,YY)
C
C     CONVERT BACK TO VIRTUAL COORDINATES
C
      X=XMINGR+(XX-0.50)*(XMAXGR-XMINGR)/(XSCLGR*NXINGR)
      Y=YMINGR+(YY-0.50)*(YMAXGR-YMINGR)/(YSCLGR*NYINGR)
C
      RETURN
      END
```

GRDRAW

```
      SUBROUTINE GRDRAW(XNEWGR,YNEWGR,ISYM
     &
     &                                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE DRAWS A LINE TO THE POINT (XNEWGR,YNEWGR)
C       AND PLOTS A SYMBOL THERE IF ISYM.NE.0.
C
C*********************************************************************
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
C     IMMEDIATE RETURN IF CTRL-C HAS BEEN ACTIVATED
```

```
C
      IF(JTRPGR) RETURN
C
C     IS NEW POINT IN BOUNDS?
C
      IF(XNEWGR.LT.XMNCGR .OR. XNEWGR.GT.XMXCGR .OR.
     &   YNEWGR.LT.YMNCGR .OR. YNEWGR.GT.YMXCGR    ) THEN
         INEWGR=0
      ELSE
         INEWGR=1
      ENDIF
C
C     IF BOTH POINTS ARE IN BOUNDS, SIMPLE DRAW
C
      IF(IOLDGR.EQ.1 .AND. INEWGR.EQ.1) THEN
         IMOVE=0
C
         XDRAW=XNEWGR
         YDRAW=YNEWGR
C
C     IF OLD POINT IS IN BOUNDS AND NEW ONE ISN'T, THEN CLIP
C       LOCATION TO WHICH WE MUST DRAW
C
      ELSEIF(IOLDGR.EQ.1 .AND. INEWGR.EQ.0) THEN
         IMOVE=0
C
         IF(XNEWGR.GT.XMXCGR) THEN
            XDRAW=XMXCGR
            YDRAW=YOLDGR+(XMXCGR-XOLDGR)/(XNEWGR-XOLDGR)
     &                                  *(YNEWGR-YOLDGR)
         ELSEIF(XNEWGR.LT.XMNCGR) THEN
            XDRAW=XMNCGR
            YDRAW=YOLDGR+(XMNCGR-XOLDGR)/(XNEWGR-XOLDGR)
     &                                  *(YNEWGR-YOLDGR)
         ELSE
            XDRAW=XNEWGR
            YDRAW=YNEWGR
         ENDIF
C
         IF(YDRAW .GT.YMXCGR) THEN
            XDRAW=XOLDGR+(YMXCGR-YOLDGR)/(YDRAW -YOLDGR)
     &                                  *(XDRAW -XOLDGR)
            YDRAW=YMXCGR
         ELSEIF(YDRAW .LT.YMNCGR) THEN
            XDRAW=XOLDGR+(YMNCGR-YOLDGR)/(YDRAW -YOLDGR)
     &                                  *(XDRAW -XOLDGR)
            YDRAW=YMNCGR
         ENDIF
C
C     IF OLD POINT IS OUT OF BOUNDS, AND NEW ONE IS IN, THEN FIND
C       THE POINT TO WHICH WE MUST MOVE BEFORE DRAWING
C
      ELSEIF(IOLDGR.EQ.0 .AND. INEWGR.EQ.1) THEN
```

```
            IMOVE=1
            IF(XOLDGR.GT.XMXCGR) THEN
               XMOVE=XMXCGR
               YMOVE=YNEWGR+(XMXCGR-XNEWGR)/(XOLDGR-XNEWGR)
     &                                      *(YOLDGR-YNEWGR)
            ELSEIF(XOLDGR.LT.XMNCGR) THEN
               XMOVE=XMNCGR
               YMOVE=YNEWGR+(XMNCGR-XNEWGR)/(XOLDGR-XNEWGR)
     &                                      *(YOLDGR-YNEWGR)
            ELSE
               XMOVE=XOLDGR
               YMOVE=YOLDGR
            ENDIF
C
            IF(YMOVE .GT.YMXCGR) THEN
               XMOVE=XNEWGR+(YMXCGR-YNEWGR)/(YMOVE -YNEWGR)
     &                                      *(XMOVE -XNEWGR)
               YMOVE=YMXCGR
            ELSEIF(YMOVE .LT.YMNCGR) THEN
               XMOVE=XNEWGR+(YMNCGR-YNEWGR)/(YMOVE -YNEWGR)
     &                                      *(XMOVE -XNEWGR)
               YMOVE=YMNCGR
            ENDIF
C
            XDRAW=XNEWGR
            YDRAW=YNEWGR
C
C     BOTH POINTS ARE OUT OF BOUNDS, SO FIRST CULL OUT OBVIOUS
C        SEGMENTS THAT DON'T NEED TO BE DRAWN
C
         ELSE
            IF(MIN(XOLDGR,XNEWGR).GT.XMXCGR .OR.
     &         MAX(XOLDGR,XNEWGR).LT.XMNCGR .OR.
     &         MIN(YOLDGR,YNEWGR).GT.YMXCGR .OR.
     &         MAX(YOLDGR,YNEWGR).LT.YMNCGR     ) GOTO 40
C
         IMOVE=1
C
C     X-CLIPPING
C
            IF(XNEWGR.LT.XOLDGR) THEN
               XMOVE=XNEWGR
               YMOVE=YNEWGR
               XDRAW=XOLDGR
               YDRAW=YOLDGR
            ELSE
               XMOVE=XOLDGR
               YMOVE=YOLDGR
               XDRAW=XNEWGR
               YDRAW=YNEWGR
            ENDIF
C
            IF(XDRAW.GT.XMXCGR) THEN
```

```
            YDRAW=YMOVE+(XMXCGR-XMOVE)/(XDRAW-XMOVE)*(YDRAW-YMOVE)
            XDRAW=XMXCGR
         ENDIF
C
         IF(XMOVE.LT.XMNCGR) THEN
            YMOVE=YDRAW+(XMNCGR-XDRAW)/(XMOVE-XDRAW)*(YMOVE-YDRAW)
            XMOVE=XMNCGR
         ENDIF
C
C     Y-CLIPPING
C
         IF(YMOVE.GE.YDRAW) THEN
            XTEMP=XDRAW
            XDRAW=XMOVE
            XMOVE=XTEMP
            YTEMP=YDRAW
            YDRAW=YMOVE
            YMOVE=YTEMP
         ENDIF
C
         IF(YMOVE.GT.YMXCGR) GOTO 40
         IF(YDRAW.GT.YMXCGR) THEN
            XDRAW=XMOVE+(YMXCGR-YMOVE)/(YDRAW-YMOVE)*(XDRAW-XMOVE)
            YDRAW=YMXCGR
         ENDIF
C
         IF(YDRAW.LT.YMNCGR) GOTO 40
         IF(YMOVE.LT.YMNCGR) THEN
            XMOVE=XDRAW+(YMNCGR-YDRAW)/(YMOVE-YDRAW)*(XMOVE-XDRAW)
            YMOVE=YMNCGR
         ENDIF
C
       ENDIF
C
C     MOVING LOGIC
C
       XWID=XSCLGR*NXINGR
       YWID=YSCLGR*NYINGR
C
       IF(IMOVE.EQ.0) GOTO 20
C
       IF(IMODGR.EQ.1) THEN
          XX=0.5+XWID*(XMOVE-XMINGR)/(XMAXGR-XMINGR)
          YY=0.5+YWID*(YMOVE-YMINGR)/(YMAXGR-YMINGR)
          CALL PLOT(XX,YY,3)
       ELSEIF(IMODGR.EQ.2) THEN
          IXX=1000.0*YWID*(YMOVE-YMINGR)/(YMAXGR-YMINGR)
          IYY=1000.0*XWID*(XMOVE-XMINGR)/(XMAXGR-XMINGR)
          WRITE(JHRDGR,10) IXX,IYY
10        FORMAT(' ^U',I5.5,',',I5.5)
       ELSEIF(IMODGR.EQ.3) THEN
          IXX=1000.0*XWID*TEXSGR*(XMOVE-XMINGR)/(XMAXGR-XMINGR)
          IYY=1000.0*YWID*TEXSGR*(YMAXGR-YMOVE)/(YMAXGR-YMINGR)
```

```
         WRITE(JHRDGR,10) IXX,IYY
       ENDIF
C
C     DRAWING LOGIC
C
20     IF(IMODGR.EQ.1) THEN
         XX=0.5+XWID*(XDRAW-XMINGR)/(XMAXGR-XMINGR)
         YY=0.5+YWID*(YDRAW-YMINGR)/(YMAXGR-YMINGR)
         CALL PLOT(XX,YY,2)
       ELSEIF(IMODGR.EQ.2) THEN
         IXX=1000.0*YWID*(YDRAW-YMINGR)/(YMAXGR-YMINGR)
         IYY=1000.0*XWID*(XDRAW-XMINGR)/(XMAXGR-XMINGR)
         WRITE(JHRDGR,30) IXX,IYY
30       FORMAT(' ^D',I5.5,':',I5.5)
       ELSEIF(IMODGR.EQ.3) THEN
         IXX=1000.0*XWID*TEXSGR*(XDRAW-XMINGR)/(XMAXGR-XMINGR)
         IYY=1000.0*YWID*TEXSGR*(YMAXGR-YDRAW)/(YMAXGR-YMINGR)
         WRITE(JHRDGR,30) IXX,IYY
       ENDIF
C
C     DRAW THE SYMBOL
C
       IF(INEWGR.EQ.1) CALL GRSYMB(XNEWGR,YNEWGR,ISYM)
C
C     STORE PRESENT POINT AS OLD POINT FOR NEXT CALL
C
40     XOLDGR=XNEWGR
       YOLDGR=YNEWGR
       IOLDGR=INEWGR
C
       RETURN
       END
```

GRINIT

```
       SUBROUTINE GRINIT(JTERMI,JTERMO,TITLE)
C
       INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE INITIALIZES THE GRAPHIC PACKAGE.  IT SHOULD BE
C       CALLED BEFORE ANY OTHER GRAPHICS ROUTINE.
C
C       JTERMI  - TERMINAL  INPUT UNIT (USUALLY 5)
C       JTERMO  - TERMINAL OUTPUT UNIT (USUALLY 6)
C       TITLE   - CASE TITLE
C
       CHARACTER TITLE*(*)
C
C*********************************************************************
C
```

```
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*****************************************************************************
C
C     STORE THE INPUT AND OUTPUT UNITS
C
        JINGR =JTERMI
        JOUTGR=JTERMO
        JHRDGR=47
        IMODGR=0
C
C     SAVE THE TITLE
C
        TITLGR=TITLE
C
C     INITIALIZE THE GRAPHICS PACKAGE
C
        INITGR=0
C
C     GET THE CURRENT TIME (FOR PLOT ANNOTATION)
C
        CALL GRTIME
C
C     INITIALIZE THE CTRL-C AST TRAP
C
        JTRPGR=.TRUE.
        CALL GRTRAP
C
C     SET UP THE INITIAL SCALING PARAMETERS
C
        NSCLGR=7
        ASCLGR(1)=1.
        ASCLGR(2)=2.
        ASCLGR(3)=4.
        ASCLGR(4)=5.
        ASCLGR(5)=10.
        ASCLGR(6)=20.
        ASCLGR(7)=40.
C
        IOLDGR=0
C
        RETURN
        END
```

GRINPA

```
        SUBROUTINE GRINPA(ITITLE,
     &
     &                  TEXT  )
C
```

```
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS A TEXT STRING, TEXT$, WHICH IS AT MOST
C        NCHARS CHARACTERS LONG
C
        CHARACTER*(*) ITITLE,TEXT
C
C*********************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
        NCHARS=LEN(TEXT)
C
10      WRITE(JOUTGR,20) ITITLE,NCHARS
20      FORMAT(' ENTER ',A,'    (',I2,' CHARACTERS MAX) >'$)
C
        READ(JINGR,30,END=10,ERR=10) TEXT
30      FORMAT(A)
C
        RETURN
        END
```

GRINPF

```
        SUBROUTINE GRINPF(ITITLE,
     &
     &                    VALUE )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN A REAL NUMBER, VALUE
C
        CHARACTER*(*) ITITLE
C
C*********************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
10      WRITE(JOUTGR,20)ITITLE
20      FORMAT(' ENTER ',A,5X,'(F20.0) >'$)
C
        READ(JINGR ,*,END=10,ERR=10) VALUE
C
        RETURN
        END
```

GRINPI

```
      SUBROUTINE GRINPI(ITITLE,
     &
     &                  IVALUE)
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE READS IN AN INTEGER NUMBER, IVALUE
C
      CHARACTER*(*) ITITLE
C
C*********************************************************************
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
10    WRITE(JOUTGR,20)ITITLE
20    FORMAT(' ENTER ',A,' >'$)
C
      READ(JINGR ,*,END=10,ERR=10) IVALUE
C
      RETURN
      END
```

GRKEY

```
      SUBROUTINE GRKEY(NSYM,FMIN,FINC,IOPT
     &
     &                                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE SETS UP AND DRAWS KEYS ON THE RIGHT-HAND-SIDE
C       OF A PLOT.  THERE ARE NSYM SYMBOLS IN THE CENTERS OF INTERVALS
C       GIVEN BY (FMIN,FMIN+FINC), (FMIN+FINC,FMIN+2*FINC), ...,
C       (FMIN+(NSYM-1)*FINC,FMIN+NSYM*FINC)
C
C          IF IOPT = 1 INITIALIZE AND SET UP CLIPPING PARAMETER
C                  = 2 DRAW KEY AND RESET CLIPPING PARAMETER
C
C*********************************************************************
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
      CHARACTER*10 ANNOT
C
C*********************************************************************
```

```
C
        GOTO (10,20), IOPT
C
C     SET UP AN AREA FOR THE KEY
C
10      XKEY=0.125*XMINGR+0.875*XMAXGR
        CALL GRMOVE(XKEY,YMINGR,0)
        CALL GRDRAW(XKEY,YMAXGR,0)
C
C     SET THE CLIPPING PARAMETER
C
        XMXCGR=XKEY
C
        RETURN
C
C     RESET CLIPPING PARAMETER AND WRITE KEY
C
20      XMXCGR=XMAXGR
C
        XX=0.85*XKEY+0.15*XMAXGR
C
        DO 30 ISYM=1,NSYM
        YY=YMAXGR-(ISYM+1.0)/(NSYM+2.0)*(YMAXGR-YMINGR)
        CALL GRMOVE(XX,YY,ISYM)
30      CONTINUE
C
        DO 50 ISYM=1,NSYM-1
        YY=YMAXGR-(ISYM+1.7)/(NSYM+2.0)*(YMAXGR-YMINGR)
        CALL GRMOVE(XX,YY,0)
        WRITE(ANNOT,40) FMIN+ISYM*FINC
40      FORMAT(F10.5)
        CALL GRANOT(ANNOT)
50      CONTINUE
C
        RETURN
C
        END
```

GRLIN1

```
        SUBROUTINE GRLIN1(X$,Y$,NDUM,GRDUMY
     &
     &                                    )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE DOES MULTIPLE LINE PLOTTING
C
        DIMENSION X$(*),Y$(*),GRDUMY(*)
C
```

```
C*******************************************************************************
C
       INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*******************************************************************************
C
C     STATEMENT FUNCTIONS FOR DECODING
C
        X(I)=X$(ISTART+I)
        Y(I)=Y$(ISTART+I)
C
        ILINLN(I)=NINT(GRDUMY(I+1))
        ISYMLN(I)=NINT(GRDUMY(I+NLINGR+1))
        NPERLN(I)=NINT(GRDUMY(I+NLINGR+NLINCR+1))
C
C*******************************************************************************
C
C     DECODE GRDUMY
C
        NLINGR=NINT(GRDUMY(1))
C
C     FIRST POINT IN LINE-1
C
        ISTART=0
C
C    LOOP THROUGH EACH LINE
C
        DO 30 ILINE=1,NLINGR
C
C     DECODE OPTIONS FOR THIS LINE
C
C     MOVE TO INITIAL POINT
C
        CALL GRMOVE(X(1),Y(1),ISYMLN(ILINE))
C
C     DRAW REST OF LINE
C
        DO 10 IPOINT=2,NPERLN(ILINE)
C
        IF(ILINLN(ILINE).EQ.0) THEN
           CALL GRMOVE(X(IPOINT),Y(IPOINT),ISYMLN(ILINE))
        ELSE
           CALL GRDRAW(X(IPOINT),Y(IPOINT),ISYMLN(ILINE))
        ENDIF
C
10      CONTINUE
C
C     INCREASE ISTART FOR NEXT LINE
C
20      ISTART=ISTART+NPERLN(ILINE)
C
C     NEXT LINE
C
```

```
30      CONTINUE
C
        RETURN
        END
```

GRLINE

```
        SUBROUTINE GRLINE(ILIN$,ISYM$,NLINE,PLTITL,INDGR,
     &                    X$,Y$,N$
     &                                                  )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CONTROLS THE MULTIPLE LINE PLOTTING PACKAGE
C
        DIMENSION X$(*),Y$(*),N$(NLINE),ILIN$(NLINE),ISYM$(NLINE)
C
        CHARACTER*(*) PLTITL
C
C**********************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
        DIMENSION GRDUMY(301)
C
        EXTERNAL GRLIN1
C
C**********************************************************************
C
C     SAVE NUMBER OF LINES
C
        NLINGR=NLINE
        IF(NLINGR.LT.1) NLINGR=1
C
C     CALCULATE TOTAL NUMBER OF POINTS
C
        NPTS=0
        DO 10 I=1,NLINGR
        NPTS=NPTS+N$(I)
10      CONTINUE
C
C     ENCODE GRDUMY
C
        GRDUMY(1)=NLINGR
C
        DO 20 I=1,NLINGR
        GRDUMY(I+1)=ILIN$(I)
        GRDUMY(I+NLINGR+1)=ISYM$(I)
        GRDUMY(I+NLINGR+NLINGR+1)=N$(I)
20      CONTINUE
```

```
C
C     PASS ARRAY NAMES AND PLOT THE LINES
C
        CALL GRAPHC(GRLIN1,INDGR,PLTITL,X$,Y$,NPTS,GRDUMY)
C
        RETURN
        END
```

GRMODE

```
        SUBROUTINE GRMODE(IMODE
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CHANGES PLOTTING MODES
C
C        IF IMODE = 0  CLEAR SCREEN
C                 = 1  INTERACTIVE
C                 = 2  QMS (LANDSCAPE)
C                 = 3  TEX/QMS (PORTRAIT)
C
C*************************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
        CHARACTER*40 NAME
C
        DATA INIT /0/
C
C*************************************************************************
C
        GOTO (100,200,300,400), (IMODE+1)
C
C*****CLEAR SCREEN
C
C     END PLOT (CLOSE FRAME AND ERASE SCREEN)
C
100     IF(INIT.NE.0) THEN
           CALL PLOT(0.,0.,+3)
           CALL PLOT(0.,0.,+2)
           CALL PLOT(0.,0.,-999)
           CALL FACTOR(1.500)
        ENDIF
C
        RETURN
C
C*****INTERACTIVE (PLOTLIB)
C
```

```
C     INITIALIZE SCREEN PLOTTING
C
200      IF(IMODGR.EQ.0) THEN
            CALL MENU(IDEVGR)
            CALL PLOTS(-1,40,IDEVGR)
            CALL FACTOR(1.500)
            INIT=1
         ELSEIF(IMODGR.EQ.2) THEN
            WRITE(JHRDGR,110)
110         FORMAT(' ^IGE^0^-'/
     &              ' ^-^PN^-')
            CLOSE(UNIT=JHRDGR)
         ELSEIF(IMODGR.EQ.3) THEN
            WRITE(JHRDGR,120)
120         FORMAT(' ^VO^IGE^-^0^-^PN^-')
            WRITE(JHRDGR+1,130)
130         FORMAT(' \end{picture}}')
            CLOSE(UNIT=JHRDGR  )
            CLOSE(UNIT=JHRDGR+1)
         ENDIF
C
         IMODGR=1
C
         RETURN
C
C*****QMS PLOTTING (LANDSCAPE)
C
300      JHRDGR=47
C
         OPEN(UNIT=JHRDGR,FORM='FORMATTED',NAME='QMS:',
     &                                      TYPE='NEW',ERR=310)
         GOTO 330
C
310      WRITE(JOUTGR,320)
320      FORMAT(' QMS IS NOT AVAILABLE,'
     &          ' FILE ''QMS.QMS'' WILL BE CREATED')
C
         OPEN(UNIT=JHRDGR,FORM='FORMATTED',NAME='QMS.QMS',TYPE='NEW')
C
330      WRITE(JHRDGR,340)
340      FORMAT(' ^PY^-'/
     &          ' ^F^-'/
     &       ' ^IOP^IMV00000011000^IMH0000008500^IGV^PW03^J01600^T01050')
C
         IMODGR=2
         RETURN
C
C*****TEX/QMS OUTPUT (PORTRAIT)
C
400      JHRDGR=47
C
         CALL GRINPA('NAME OF OUTPUT FILES',NAME)
         LEN=INDEX(NAME,' ')-1
```

```
      IRB=INDEX(NAME,']')+1
C
      CALL GRINPF('SCALE FACTOR',TEXSGR)
C
      OPEN(UNIT=JHRDGR  ,FORM='FORMATTED',NAME=NAME(1:LEN)//'.QMS',
     &                                    TYPE='NEW',ERR=410)
      OPEN(UNIT=JHRDGR+1,FORM='FORMATTED',NAME=NAME(1:LEN)//'.TEX',
     &                                    TYPE='NEW',ERR=410)
      GOTO 430
C
410   WRITE(JOUTGR,420)
420   FORMAT(' ONE OF THE OUTPUT FILES COULD NOT BE CREATED')
      IMODGR=1
      RETURN
C
430   WRITE(JHRDGR,440)
440   FORMAT(' ^PY^-'/
     &       ' ^F^-'/
     &     ' ^IOP^IMV0000011000^IMH0000008500^IGV^PW03^J00000^T00000')
C
      XSIZE=0.500/TEXSGR+NXINGR*XSCLGR
      YSIZE=0.500/TEXSGR+NYINGR*YSCLGR
      XORIG=            NXINGR*XSCLGR
      YORIG=            NYINGR*YSCLGR
C
      WRITE(JHRDGR+1,450) DATEGR,TIMEGR,
     &                    TEXSGR,
     &                    XSIZE,YSIZE,(-.5/TEXSGR),(-.25/TEXSGR),
     &                    YORIG,XORIG,YORIG,
     &                    NAME(IRB:LEN)
450   FORMAT(' {%GRAFIC output on ',A,' at ',A                      /
     &       ' \setlength{\unitlength}{',F6.3,'in}'                 /
     &       ' \thicklines'                                         /
     &       ' \scriptsize'                                         /
     &       ' \begin{picture}(',F6.3,',',F6.3,')(',
     &                                        F6.3,',',F6.3,')'/
     &       ' \put(0,',F6.3,'){\begin{picture}(',F6.3,',',F6.3,')'/
     &       ' \special{include(',A,'.QMS  origin noorigin '       ,
     &       ' noorient nofree fortran norelative)}'                /
     &       ' \end{picture}}'                                      )
C
      IMODGR=3
      RETURN
C
      END
```

GRMOVE

```
      SUBROUTINE GRMOVE(XNEWGR,YNEWGR,ISYM
     &
```

```
     &                                        )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE MOVE TO THE POINT (XNEWGR,YNEWGR)
C        AND PLOTS A SYMBOL THERE IF ISYM.NE.0.
C
C*********************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
C     IMMEDIATE RETURN IF CTRL-C HAS BEEN ACTIVATED
C
        IF(JTRPGR) RETURN
C
C     IF IT IS IN BOUNDS, MOVE THERE AND PLOT SYMBOL
C
        IF(XNEWGR.LT.XMNCGR .OR. XNEWGR.GT.XMXCGR .OR.
     &     YNEWGR.LT.YMNCGR .OR. YNEWGR.GT.YMXCGR    ) THEN
           INEWGR=0
        ELSE
           XWID=XSCLGR*NXINGR
           YWID=YSCLGR*NYINGR
C
           IF(IMODGR.EQ.1) THEN
              XX=0.5+XWID*(XNEWGR-XMINGR)/(XMAXGR-XMINGR)
              YY=0.5+YWID*(YNEWGR-YMINGR)/(YMAXGR-YMINGR)
              CALL PLOT(XX,YY,3)
           ELSEIF(IMODGR.EQ.2) THEN
              IXX=1000.0*YWID*(YNEWGR-YMINGR)/(YMAXGR-YMINGR)
              IYY=1000.0*XWID*(XNEWGR-XMINGR)/(XMAXGR-XMINGR)
              WRITE(JHRDGR,10) IXX,IYY
10            FORMAT(' ^U',I5.5,':',I5.5)
           ELSEIF(IMODGR.EQ.3) THEN
              IXX=1000.0*XWID*TEXSGR*(XNEWGR-XMINGR)/(XMAXGR-XMINGR)
              IYY=1000.0*YWID*TEXSGR*(YMAXGR-YNEWGR)/(YMAXGR-YMINGR)
              WRITE(JHRDGR,10) IXX,IYY
           ENDIF
           CALL GRSYMB(XNEWGR,YNEWGR,ISYM)
           INEWGR=1
        ENDIF
C
C     STORE PRESENT POINT AS OLD POINT FOR NEXT CALL
C
        XOLDGR=XNEWGR
        YOLDGR=YNEWGR
        IOLDGR=INEWGR
C
        RETURN
        END
```

GRPER1

```
      SUBROUTINE GRPER1(X$,Y$$,NDUM,GRDUMY
     &
     &                                  )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES PERSPECTIVE PLOTS
C
      DIMENSION X$(*),Y$$(*),GRDUMY(*)
C
C*****************************************************************************
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*****************************************************************************
C
C     STATEMENT FUNCTIONS FOR INDEXING
C
      KK(I,J)=I+(J-1)*NPERLN
C
C     DECODE GRDUMY
C
      IBEG=NINT(GRDUMY(1))
      IEND=NINT(GRDUMY(2))
      INCR=NINT(GRDUMY(3))
      XINC=     GRDUMY(4)
      YINC=     GRDUMY(5)
C
      XSAVE1=X$(1)
      YSAVE1=Y$$(1)
      XSAVE2=X$(2)
      YSAVE2=Y$$(2)
C
      X$(1) =GRDUMY( 7)
      Y$$(1)=GRDUMY( 8)
      X$(2) =GRDUMY( 9)
      Y$$(2)=GRDUMY(10)
C
      NPERLN=NINT(GRDUMY(6))
C
C     SAVE FIRST AND LAST POINT OF FIRST LINE SO THAT SYMBOLS CAN
C       BE PLOTTED
C
      XLEFT=X$ (        1       )
      YLEFT=Y$$(KK(     1,IBEG))
      XRITE=X$ (   NPERLN       )
      YRITE=Y$$(KK(NPERLN,IBEG))
C
C     STEP THROUGH EACH LINE TO BE PLOTTED
```

```
C
        XOFF=-XINC
        YOFF=-YINC
C
        DO 30 ILINE=IBEG,IEND,INCR
C
        XOFF=XOFF+XINC
        YOFF=YOFF+YINC
C
C     DRAW  SYMBOLS AT THE FIRST AND LAST POINTS TO SHOW PERSPECTIVE
C
        CALL GRMOVE(XLEFT+XOFF,YLEFT+YOFF,4)
        CALL GRMOVE(XRITE+XOFF,YRITE+YOFF,4)
C
C     MOVE TO THE FIRST POINT OF THIS LINE
C
        XX=X$ (   1       )+XOFF
        YY=Y$$(KK(1,ILINE))+YOFF
        CALL GRMOVE(XX,YY,0)
C
C     DRAW TO THE REST OF THE POINTS ON THIS LINE
C
        DO 20 IPNT=2,NPERLN
C
        XX=X$ (   IPNT       )+XOFF
        YY=Y$$(KK(IPNT,ILINE))+YOFF
        CALL GRDRAW(XX,YY,0)
C
20      CONTINUE
C
30      CONTINUE
C
C     LOOP THROUGH EACH POINT FOR CONNECTING LINES
C
        DO 50 IPNT=1,NPERLN
C
C     MOVE TO THAT POINT ON THE FIRST LINE
C
        XX=X$ (   IPNT      )
        YY=Y$$(KK(IPNT,IBEG))
        CALL GRMOVE(XX,YY,0)
C
C     LOOP THROUGH ALL REMAINING LINES, DRAWING TO EACH IN TURN
C
        XOFF=0.0
        YOFF=0.0
C
        DO 40 ILINE=IBEG+INCR,IEND,INCR
C
        XOFF=XOFF+XINC
        YOFF=YOFF+YINC
C
        XX=X$ (   IPNT       )+XOFF
```

```
        YY=Y$$(KK(IPST,ILINE))+YOFF
        CALL GRDRAW(XX,YY,0)
C
40      CONTINUE
C
50      CONTINUE
C
        X$(1) =XSAVE1
        Y$$(1)=YSAVE1
        X$(2) =XSAVE2
        Y$$(2)=YSAVE2
C
        RETURN
        END
```

GRPERS

```
        SUBROUTINE GRPERS(X$,Y$$,NPERLN,NLINES,OPT$,PLTITL,INDGR
     &
     &                                                        )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CONTROLS PERSPECTIVE LINE PLOTS.
C
        DIMENSION X$(NPERLN),Y$$(NPERLN,NLINES),OPT$(*)
C
        CHARACTER*(*) PLTITL
C
C*****************************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMM.INC'
C
        EXTERNAL G2PER1
C
        DIMENSION GRDUMY(10)
C
        CHARACTER*76 PLTIT
C
C*****************************************************************************
C
C     IF OPTIONS ARE GIVEN, SIMPLY ENCODE GRDUMY
C
10      IF(OPT$(1).NE.0.0) THEN
           GRDUMY(1)=OPT$(1)
           GRDUMY(2)=OPT$(2)
           GRDUMY(3)=OPT$(3)
           GRDUMY(4)=OPT$(4)
           GRDUMY(5)=OPT$(5)
           GRDUMY(6)=NPERLN
```

```
C
C     OTHERWISE, ASK QUESTIONS AS TO SETTINGS
C
        ELSE
           WRITE(JOUTGR,20) NLINES,NPERLN
20         FORMAT(' THERE ARE',I4,' LINES OF',I4,' POINTS EACH')
C
           CALL GRINPI('First line to plot  ',IN)
C
           IF(IN.EQ.0) THEN
              GRDUMY(1)=1.0
              GRDUMY(2)=NLINES
              GRDUMY(3)=1.0
              GRDUMY(4)=0.0
              GRDUMY(5)=0.0
              GRDUMY(6)=NPERLN
           ELSEIF(IN.EQ.NLINES) THEN
              GRDUMY(1)=NLINES
              GRDUMY(2)=NLINES
              GRDUMY(3)=1.0
              GRDUMY(4)=0.0
              GRDUMY(5)=0.0
              GRDUMY(6)=NPERLN
           ELSE
              GRDUMY(1)=MAX(IN,1)
              CALL GRINPI('Last  line to plot  ',IEND)
              GRDUMY(2)=MIN(IEND,NLINES)
              CALL GRINPI('Increment           ',INCR)
              GRDUMY(3)=MAX(INCR,1)
              CALL GRINPF('X-offset',GRDUMY(4))
              CALL GRINPF('Y-offset',GRDUMY(5))
              GRDUMY(6)=NPERLN
           ENDIF
        ENDIF
C
        IBEG=NINT(GRDUMY(1))
        IEND=NINT(GRDUMY(2))
        INCR=NINT(GRDUMY(3))
        XOFF=     GRDUMY(4)
        YOFF=     GRDUMY(5)
C
C     COMPUTE THE MINIMUM AND MAXIMUM DATA VALUES FOR SCALING
C
        XMIN=X$ (1     )
        XMAX=X$ (1     )
        YMIN=Y$$(1,IBEG)
        YMAX=Y$$(1,IBEG)
C
C     SCAN THROUGH EACH LINE
C
        DO 40 ILINE=IBEG,IEND,INCR
        XOFF=XOFF+GRDUMY(4)
        YOFF=YOFF+GRDUMY(5)
```

```
C
C     SCAN THROUGH EACH POINT ON THIS LINE
C
        DO 30 IPNT=1,NPERLN
        XX=X$ (IPNT      )+XOFF
        YY=Y$$(IPNT,ILINE)+YOFF
C
        XMIN=MIN(XMIN,XX)
        XMAX=MAX(XMAX,XX)
        YMIN=MIN(YMIN,YY)
        YMAX=MAX(YMAX,YY)
C
30      CONTINUE
C
40      CONTINUE
C
C     INCLUDE LINE IDENTIFICATION IN TITLE
C
        WRITE(PLTIT,45) PLTITL(1:MIN(60,LEN(PLTITL))),IBEG,IEND,INCR
45      FORMAT(A,'  (',I4,',',I4,',',I4,')')
C
        GRDUMY( 7)=X$(1)
        GRDUMY( 8)=Y$$(1,1)
        GRDUMY( 9)=X$(2)
        GRDUMY(10)=Y$$(2,1)
C
        X$(1)   =XMIN
        Y$$(1,1)=XMIN
        X$(2)   =XMAX
        Y$$(2,1)=YMAX
C
C     PLOT
C
        CALL GRAPHC(GRPER1,INDGR,PLTIT,X$,Y$$,2,GRDUMY)
C
        X$(1)   =GRDUMY( 7)
        Y$$(1,1)=GRDUMY( 8)
        X$(2)   =GRDUMY( 9)
        Y$$(2,1)=GRDUMY(10)
C
C     GO BACK IF OPTIONS WERE SELECTED INTERACTIVELY
C
        IF(OPT$(1).NE.0.0) RETURN
C
50      CALL GRINPI('Repeat with another set of options (0/1)',IANS)
        IF(IANS.EQ.1) GOTO 60
        IF(IANS.NE.0) GOTO 50
                      RETURN
C
C     WRITE LAST SET OF OPTIONS FOR COMPARISON
C
60      WRITE(JOUTGR,70) IBEG,GRDUMY(4),
     &                   IEND,GRDUMY(5),
```

```
     &                    INCR
70      FORMAT(' PREVIOUS SETTINGS:'                /
     &         '    IBEG=',I5,'     X-OFFSET=',F10.6/
     &         '    IEND=',I5,'     Y-OFFSET=',F10.6/
     &         '    INCR=',I5                       )
C
        GOTO 10
C
        END
```

GRPRNT

```
        SUBROUTINE GRPRNT(XNUM,XINC,
     &
     &                    NUMBER   )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE CONVERTS THE REAL NUMBER XNUM INTO A THE
C        CHARACTER*13 STRING NUMBER.  THE FORMAT DEPENDS ON THE SIZE
C        OF XNUM AND THE INCREMENT XINC
C
        CHARACTER*13 NUMBER
C
C*********************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
        CHARACTER*12 FORMAT
C
C*********************************************************************
C
C     THE NUMBER OF PLACES BEFORE THE DECIMAL DEPENDS ON THE SIZE OF
C        THE NUMBER
C
        IF(XNUM.EQ.0) THEN
           MDEC=1
        ELSE
           MDEC=MAX(INT(LOG10(ABS(XNUM))),0)+1
        ENDIF
C
C     THE NUMBER OF PLACES AFTER THE DECIMAL DEPENDS ON THE INCREMENT
C
        NDEC=MAX(INT(0.999-LOG10(ABS(XINC))),0)
C
C     COMPUTE THE TOTAL WIDTH
C
        IWID=MDEC+NDEC+2
C
        IF(MDEC.LE.8 .AND. NDEC.LE.5 .AND. IWID.LE.13) THEN
```

```
            NSPACE=(13-IWID)/2
            WRITE(FORMAT,10) NSPACE,IWID,NDEC
10          FORMAT('(',I2,'X,F',I2,'.',I2,')')
            WRITE(NUMBER,FORMAT) XNUM
         ELSE
            WRITE(NUMBER,20) XNUM
20          FORMAT(E13.7)
         ENDIF
C
         RETURN
         END
```

GRSCAL

```
         SUBROUTINE GRSCAL(XMIN,XMAX,AXLEN,
     &
     &                     XSTART,XINC    )
C
         INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE GENERATES SCALE FACTORS FOR PLOTTING
C
C*************************************************************************
C
         INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*************************************************************************
C
C     CALCULATE THE ROUGH BREAKUP
C
         F=(XMAX-XMIN)/AXLEN
         IF(F.EQ.0.) F=XMAX/AXLEN
C
C     FIND THE PROPER EXPONENT FOR SCALING
C
         J=-30
         DO 10 I=1,60
         AK=F*10.0**J
         IF(AK.GE.1.0) GOTO 20
         J=J+1
10       CONTINUE
C
C     FIND THE PROPER MANTISSA FOR SCALING
C
20       MSCLGR=NSCLGR-1
         DO 30 II=2,MSCLGR
         I=II
         IF(ASCLGR(I).GT.AK) GOTO 40
30       CONTINUE
C
```

```
C     ENSURE THAT 0.0 FALLS ON A GRID LINE (OR AN EXTRAPOLATION
C        OF ONE)
C
40      XINC=ASCLGR(I-1)/10.0**J
        K=XMIN/XINC
        IF((XMIN/XINC).LT.0.0) K=K-1
        XSTART=K*XINC
C
C     CHECK THAT THE SCALING IS ADEQUATE
C
        XHI=(XMAX-XSTART)/XINC
        XLO=(XMIN-XSTART)/XINC
        IF(XHI.LT.(AXLEN+0.00001) .AND. XLO.GT.-.00001) GOTO 50
        I=I+1
        IF(I.GT.NSCLGR) GOTO 50
        GOTO 40
C
50      RETURN
        END
```

GRSSET

```
        SUBROUTINE GRSSET(XMIN,XMAX,YMIN,YMAX
     &
     &                                    )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE IS USED TO SET SCALES FOR THE GRAPHICS PACKAGE
C
C*********************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*********************************************************************
C
        XMINGR=XMIN
        XMAXGR=XMAX
C
        YMINGR=YMIN
        YMAXGR=YMAX
C
        RETURN
        END
```

GRSYMB

```
      SUBROUTINE GRSYMB(XIN,YIN,ISYM
     &
     &                              )
C
      INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE PLOTS A SYMBOL AT THE CURRENT LOCATION
C        IF ISYM.NE.0.
C
C     ISYM = 0  NO SYMBOL            ISYM = 8  PICNIC TABLE
C            1  SQUARE                    9  Z
C            2  CIRCLE                   10  Y
C            3  TRIANGLE                 11  SMALL SQUARE WITH X
C            4  +                        12  *
C            5  X                        13  HOUR-GLASS
C            6  DIAMOND                  14  VERTICAL BAR
C            7  ARROW (POINTING UP)      15  STAR
C
C*************************************************************************
C
      INCLUDE '[.GRAFIC]GRCOMN.INC'
C
      CHARACTER*48 ISYM2(  15)
      CHARACTER*1 ISYMB,IUPDN(0:1)
C
      DIMENSION ISYM3(3,15,15)
C
      DATA ISYM2/
     &     '^P0009FF8080808080808080F8808080808080808080FF80^G^IGV',
     &     '^P00093E0041008080808080F8808080808C8041003E00^G^IGV',
     &     '^P0009C000B0008C008300F88083008C00B000C000^G^IGV',
     &     '^P000908000800080008000FF8008000800080008000800^G^IGV',
     &     '^P0009808041002200140008001400220041008080^G^IGV',
     &     '^P000908001400220041008F8804100220014000800^G^IGV',
     &     '^P00090800C000A000900FF8009000A000C000800^G^IGV',
     &     '^P0009308041802280148008801480228041808080^G^IGV',
     &     '^P0009808C080A8809880888808C808A8081308080^G^IGV',
     &     '^P0009008001000200040008000400020001000080^G^IGV',
     &     '^P0009808041003E0036002A0036003E0041008080^G^IGV',
     &     '^P0009888049002A001C00FF801C002A0049008880^G^IGV',
     &     '^P00098080C180A2809480888094B0A280C1808080^G^IGV',
     &     '^P000900000000000000000FF8000000000000000000^G^IGV',
     &     '^P000922003E0036006300A280630036003E002200^G^IGV'/
C
      DATA IUPDN/'U','D'/
C
      DATA (((ISYM3(I,J,K),J=1,15),I=1,3),K= 1, 5)/
     &          0,-4, 0, 8, 0,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &         -4, 0, 8, 0,-8, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
     &           1, 1, 1, 1, 1, 1, 0,-1, 0, 0, 0, 0, 0, 0, 7,
C
     &           0,-2,-2, 0, 2, 4, 2, 0,-2,-2, 0, 0, 0, 0, 0,
     &          -4, 0, 2, 4, 2, 0,-2,-4,-2, 0, 4, 0, 0, 0, 0,
     &           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,-1, 0, 0,11,
C
     &           0,-4, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 8, 0,-8, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           1, 1, 1, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 5,
C
     &           0, 0,-4, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           0, 1, 0, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 5,
C
     &          -4, 8, 0,-8, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 8,-8, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           0, 1, 0, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 5/
C
      DATA (((ISYM3(I,J,K),J=1,15),I=1,3),K= 6,10)/
     &           0,-4, 4, 4,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 4, 4,-4,-4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0,
     &           1, 1, 1, 1, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 6,
C
     &           0, 0,-4, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           4,-8, 4, 0,-4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           0, 1, 1, 1, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 6,
C
     &          -4, 8,-8, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           4,-8, 0, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           0, 1, 1, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 5,
C
     &          -4, 8,-8, 8, 0,-8, 4, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 0, 8, 0,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           0, 1, 1, 1, 0, 1, 0,-1, 0, 0, 0, 0, 0, 0, 7,
C
     &          -4, 4, 4,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 4,-4, 4, 4,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           0, 1, 1, 0, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 6/
C
      DATA (((ISYM3(I,J,K),J=1,15),I=1,3),K=11,15)/
     &          -4, 8, 0,-8, 2, 0, 4, 0,-4, 2, 0, 0, 0, 0, 0,
     &          -4, 8,-8, 8,-2,-4, 0, 4, 0,-2, 0, 0, 0, 0, 0,
     &           0, 1, 0, 1, 0, 1, 1, 1, 1, 0,-1, 0, 0, 0,10,
C
     &          -4, 8, 0,-8, 0, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 8,-8, 8,-4, 0,-4, 8,-4, 0, 0, 0, 0, 0, 0,
     &           0, 1, 0, 1, 0, 1, 0, 1, 0,-1, 0, 0, 0, 0, 9,
C
     &          -4, 8,-8, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 0, 8, 0,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           1, 1, 1, 1, 1,-1, 0, 0, 0, 0, 0, 0, 0, 0, 5,
C
     &           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
     &          -4, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           0, 1, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3,
C
     &          -4, 8,-8, 8,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &          -4, 0, 8, 0,-4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     &           1, 1, 1, 1, 1,-1, 0, 0, 0, 0, 0, 0, 0, 0, 5/
C
C*************************************************************************
C
        IF(ISYM.LE.0) RETURN
C
        XWID=XSCLGR*NXINGR
        YWID=YSCLGR*NYINGR
C
C     DRAW THE SYMBOL
C
        IF(IMODGR.EQ.1) THEN
           XX=0.5+XWID*(XIN-XMINGR)/(XMAXGR-XMINGR)
           YY=0.5+YWID*(YIN-YMINGR)/(YMAXGR-YMINGR)
           ISYMB=CHAR(ABS(ISYM)-1)
           CALL SYMBOL(XX,YY,0.05,ISYMB,0.0,-1)
C
        ELSEIF(IMODGR.EQ.2) THEN
           IXX=1000.0*YWID*(YIN-YMINGR)/(YMAXGR-YMINGR)
           IYY=1000.0*XWID*(XIN-XMINGR)/(XMAXGR-XMINGR)
           JSYM=MAX(1,MIN(ISYM,15))
           WRITE(JHRDGR,10) IXX+1024,IYY+1574,
     &                      ISYM2(JSYM),
     &                      IXX,IYY
10         FORMAT(' ^IGE^IH',I5.5,'^IV',I5.5,'^IPO202'/
     &           1X,A,' ^U',I5.5,':',I5.5            )
           ELSEIF(IMODGR.EQ.3) THEN
           IXX=1000.0*XWID*TEXSGR*(XIN-XMINGR)/(XMAXGR-XMINGR)
           IYY=1000.0*YWID*TEXSGR*(YMAXGR-YIN)/(YMAXGR-YMINGR)
           JSYM=MAX(1,MIN(ISYM,15))
           WRITE(JHRDGR,20) IXX,IYY
20         FORMAT(' ^U',I5.5,':',I5.5)
           WRITE(JHRDGR,30) (IUPDN(ISYM3(3,J,JSYM)),
     &                           5*ISYM3(1,J,JSYM),
     &                           5*ISYM3(2,J,JSYM),J=1,ISYM3(3,15,JSYM))
30         FORMAT(' ^',A,SPI6.5,':',SPI6.5,:,' ^',A,SPI6.5,':',SPI6.5,:,
     &           ' ^',A,SPI6.5,':',SPI6.5,:,' ^',A,SPI6.5,':',SPI6.5,:)
           ENDIF
C
        RETURN
        END
```

GRTIME

```
        SUBROUTINE GRTIME
```

```
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE UPDATES IDATE AND ITIME
C
C*************************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*************************************************************************
C
        CALL DATE(DATEGR)
        CALL TIME(TIMEGR)
C
        RETURN
        END
```

GRTRAP

```
        SUBROUTINE GRTRAP
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE ENABLES THE ASYNCHRONOUS TRAP (CTRL-C)
C
C*************************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
        INCLUDE '($iodef)'
        INTEGER SYS$ASSIGN,SYS$QIOW
        INTEGER*2 INPUT_CHANNEL
        INTEGER*4 CODE,set_c
        EXTERNAL GRTRPS
C
        STRUCTURE /IOSTAT_BLOCK/
           INTEGER*2 IOSTAT
           BYTE TRANSMIT,RECEIVE,CRFILL,LFFILL,PARITY,ZERO
        END STRUCTURE
        RECORD /IOSTAT_BLOCK/ IOSB
C
C*************************************************************************
C
C     IF JTRPGR IS NOT TRUE, THEN THERE IS NO REASON TO SET A TRAP
C
        IF(.NOT.JTRPGR) RETURN
C
        JTRPGR=.FALSE.
C
C     THE FOLLOWING CODE IS A MODIFICATION OF RICH SHAPIRO'S
```

```
C
        STATUS=SYS$ASSIGN('SYS$INPUT',INPUT_CHAN,,)
        CODE  =IO$_SETMODE .OR. IO$M_CTRLCAST
        STATUS=SYS$QIOW(,%VAL(INPUT_CHAN),%VAL(CODE),IOSB,
     &                                              ,,GRTRPS,JTRPGR,,,,)
C
        RETURN
        END
```

GRTRPS

```
        SUBROUTINE GRTRPS(ASTREC
     &
     &                          )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
C     THIS SUBROUTINE IS CALLED BY THE AST TRAP HANDLER
C
        LOGICAL ASTREC
C
C*************************************************************************
C
        ASTREC=.TRUE.
C
        RETURN
        END
```

GRTSET

```
        SUBROUTINE GRTSET(DATE,TIME
     &
     &                             )
C
        INCLUDE '[.UTILITY]PROLOG.INC'
C
        CHARACTER*9  DATE
        CHARACTER*8  TIME
C
C     THIS SUBROUTINE SETS DATE AND TIME TO USER DEFINED VALUES
C
C*************************************************************************
C
        INCLUDE '[.GRAFIC]GRCOMN.INC'
C
C*************************************************************************
C
```

```
      DATEGR=DATE
      TIMEGR=TIME
C
      RETURN
      END
```

Appendix C

MITOSIS Sampl Output

This appendix contains a listing of th basic test case given in chapter 3. The case stream Mach number of $M_\infty = 0.75$ and a computational grid is a 33×17 O-type me from the airfoil leading edge.

The knowledge base is that which was us in this thesis. The special rules which were a at the bottom of the file.

C.1

A060

0

1.0

0.

-0

2

33

1.

10.

1.

0.9

0.9

0.9

0.9

0.9

0.9

0.9

0.9

0.9

0.9

0.9

0.9

0.9

0.8

0.8

0.8

0.8

0.8

0.7

0.7

0.7

0.7

0.7

0.6

0.6

0.6

0.6

0.5

0.5

0.5

0.5

0.5

0.4

0.4

0.
0.
0.

0.2
0.2
0.2
0.2
0.2
0.3
0.3
0.3
0.3
0.4
0.4
0.4
0.4
0.5
0.5
0.5
0.5
0.5
0.6
0.6
0.6
0.6
0.7
0.7
0.7
0.7
0.7
0.8
0.8
0.8
0.8
0.8
0.9
0.9
0.9
0.9
0.9
0.9
0.9
0.9
0.9
0.9
0.9
0.9
0.9
1.0

C

C
C

C

T
D
P
O

C

I
I
I
I
I
I
I

I
I
I

C

I
I
I
I
I
I

I
I
I

I
I

I
I
I

```
IN
IN
IN
IN

IN
IN
IN
IN
IN
IN

CO

RU
CO
IF
TH
AN
AN
AN
AN
AN
AN
AN

CO

RU
CO
IF
TH
AN
AN
AN
AN

RU
CO
IF
TH

RU
CO
IF
TH
```

RU
CO
IF
TH

CO

RU
CO
IF
TH
AN
AN
AN
AN
AN
AN
AN
AN
AN
AN
AN
AN
AN

CO

RU
CO
IF
TH
AN
AN
AN

RU
CO
IF
AN
AN
TH
AN

RU
CO
IF
TH

RU

```
CO
IF
TH
AN
AN
AN
AN
AN
AN
AN

RU
CO
IF
TH
AN
AN
AN

CO

RU
CO
IF
TH
AN
AN
AN
AN
AN
AN

RU
CO
IF
TH

RU
CO
IF
TH

CO

RU
CO
IF
AN
```

```
THEN

RULE
CONT
IF
ANDI
THEN

RULE
CONT
IF
ANDI
ANDI
ANDI
THEN
ANDT

COMM

RULE
PRIO
CONT
IF
THEN
ANDI
ANDI

END
```

Appendix D

EXPROC User's Guide

This appendix describes the EXPROC language and gives examples of both its forward- and backward-chaining inference engines.

The appendix begins with a discussion of the EXPROC language elements, followed by the rules for formatting and ordering EXPROC statements within the knowledge base. A description of each of the EXPROC statement types and how to use them completes the first part of the appendix.

The second part of the appendix discusses the knowledge base used in the local compressible flow example in chapter 7. It begins with a listing of the knowledge base, followed by the output of the EXPROC compiler. The appendix finishes with the printed output which is generated by application of both the forward- and backward-chaining inference engines.

D.1 Language Elements

D.1.1 Attributes

Attributes are the part of the knowledge base where data is stored. Attributes are composed of an *attribute name* and a *value*. The attribute name is a character string (up to 10 characters long) and the value is a REAL, single-precision number.

Attributes are created in three ways:

- automatically by the EXPROC compiler. Each time a knowledge base is read, the following six attributes are created: `*`, `#1`, `#2`, `#3`, `#4`, and `UNKNOWN`; the first five are initialized with the value `0.0` whereas the latter is given the value `UNKNOWN`.

- directly by an `INITIAL` statement. In this case the attribute is initialized with the value specified by the `INITIAL` statement.

- indirectly by reference in an `IF`, `ANDIF`, `THEN`, or `ANDTHEN` statement. In this case, the attribute is given the value `UNKNOWN`. An exception to this rule occurs if the first character of the attribute name is a `+` or `-`; in this case, the value is initialized to the value given by the attribute name. For example, the attribute `+2` is initialized with the value `+2.0`.

D.1.2 Contexts

Contexts are names of groups of rules which are available for execution at any time. On every cycle of the forward- and backward-chainers, only rules with the same context as the current context are considered for inclusion in the conflict set.

Both inference engines begin with the current context set to `initial`. The context remains at its current value until a `CONSIDER` statement is executed in an action clause of a fired rule.

Contexts names are character strings of from 1 to 10 characters. They are generated either by the processing of a CONTEXT statement or by reference in an action clause where the action is CONSIDER.

D.1.3 Rules

The second part of a knowledge base is composed of rules, that is statements about the inter-relations of attributes. Each rule is composed of three parts:

- The header, which itself contains:
  - The rule name, which is printed every time that the rule is fired.
  - The context, which specifies when the rule should be scanned for inclusion in the conflict set. If no context is specified, the rule is available for inclusion in the conflict set regardless of the current context.
  - The priority, which is used in the conflict resolution if PRIORITY is specified on the ORDER statement.
- Zero or more premise clauses, as specified by the IF and ANDIF statements. For rules with zero premises, the IF statement should take the form: IF * NOP.
- One or more action clauses, as specified by the THEN and ANDTHEN statements.

D.2 EXPROC Statements

Knowledge bases written in the EXPROC language follow the set of rules given below. The first two sections are concerned with the format and ordering of EXPROC statements while the final section describes each of the statement types in the EXPROC language.

D.2.1 Format

An EXPROC knowledge base consists of a file of EXPROC statements, with one statement per record. EXPROC statements are basically free-form, with the following rules:

- EXPROC statements begin in column 1 and continue to either the first ; symbol or column 80 (the ; denotes the begining of a comment).
- A blank line (or one whose first non-blank character is ;) is considered a `COMMENT` statement.
- Each statement contains up to four tokens (depending on statement type).
- Tokens are separated by one or more blank characters.
- Tokens consist of from 1 to 10 characters.
- Valid characters in tokens include the entire printable ASCII character set except the ; sign which is used to signify the beginning of a comment (see above) and the % sign which is used to indicate if an attribute is input to or output from an action.
- There is no case-folding in the tokens (`a` and `A` are different tokens).

D.2.2 Order

The statements in an EXPROC knowledge base may be ordered according to the rules given in the table below. Each row of the table indicates the statement types which the statement type given on the left of the table may follow. For example, an `ANDIF` statement can follow either an `ANDIF` or an `IF` statement.

	s t a r t	A N D I F	A N D T H E N	C O M M E N T	D O E S N O T E X I S T	E N D	I F	I N I T I A L	O R D E R	P R O D U C T I O N	P R I O R I T Y	R U L E	T H E N	T R A C E
ANDIF		x					x							
ANDTHEN			x									x		
COMMENT	x	x	x	x	x	x	x	x	x	x	x	x	x	x
CONTEXT				x						x		x		
DYNAMIC	x				x			x	x		x			x
END			x									x		
IF	x				x			x	x		x			x
INITIAL	x				x			x	x		x			x
ORDER	x				x			x	x		x			x
PRIORITY				x						x		x		
PRODCTN	x				x			x	x		x			x
RULE	x		x		x			x	x		x		x	x
THEN		x					x							
TRACE	x				x			x	x		x			x

D.2.3 Language summary

In this section, the syntax and a description of each type of EXPROC statement is given.

The specification of the syntax follows these rules:

UPPERCASE A word given in upper-case characters indicates a control

word and should be typed exactly as shown (in upper-case).

lowercase Lower-case words indicate names or values for which any valid name or value can be used (in either upper-case, lower-case, or mixed-case).

[] Square brakets are used to enclose optional arguments.

| A vertical bar is used to indicate that one of the choices which it joins is to be used.

ANDIF

This statement is used to define a premise clause of a rule (other than the first). Its syntax is:

```
ANDIF attribute_1 comparator attribute_2
```

where the arguments are the same as for statement type IF.

ANDTHEN

This statement defines an action clause of a rule (other than the first). Its syntax is:

```
ANDTHEN action attribute_A attribute_B attribute_C
```

where the arguments are the same as for statement type THEN.

COMMENT

This statement serves only as a comment in the knowledge base and is not processed at all by the EXPROC compiler. Its syntax is:

```
COMMENT comment
```

CONTEXT

This statement, which appears in the header of a rule, specifies the context of the rule. Its syntax is:

```
CONTEXT rule_context
```

where `rule_context` is a valid context name. If no `CONTEXT` statement is given for a rule, then the rule is scanned on every pass, independent of the current context.

DYNAMIC

This control statement turns dynamic rule ordering on or off. Its syntax is:

```
DYNAMIC ON|OFF
```

If dynamic rule ordering is `ON`, then only the first rule in the conflict set is fired and the conflict set is discarded before the next cycle; thus all the rules are scanned on every cycle. If dynamic rule ordering is `OFF`, then all rules in the conflict set are fired before the rules are scanned again. The default value is `OFF`.

END

This statement marks the end of the knowledge base. Its syntax is:

```
END
```

IF

This statement is used to define the first premise clause in a rule. Its syntax is:

```
IF attribute_1 comparator attribute_2
```

where attribute_1 and attribute_2 are the names of two attributes. The comparator is one of the following:

```
.EQ.     Equal to
.LT.     Less than
.LE.     Less than or equal to
.GE.     Greater than or equal to
.GT.     Greater than
.NE.     Not equal to
 NOP     No operation
```

The premise clause is true if the algebraic comparison of the given attributes' values is true. The comparator NOP is used to indicate that the rule does not have a premise; typically * is used as a place-holder for comparator_1.

INITIAL

This statement creates an attribute and gives it an initial value. Its syntax is:

```
INITIAL attribute_name initial_value
```

where attribute_name is any valid attribute name and initial_value is a REAL number.

ORDER

This control statement determines the conflict resolution strategy to be employed. Its syntax is:

```
ORDER first [second [third]]
```

where first, second, and third represent the first (most important), second, and third (least important) orderings to use. They are chosen from the list:

PRIORITY	Largest priority is executed first.
PREMISES	Most premises is executed first.
L_RECENT	Least recently used is executed first.
M_RECENT	Most recently used is executed first.
UNKNOWNS	Rule with least unknowns is executed first.
RANDOM	Conflict set is put in a random order.

The default is to resolve conflicts such that the rule with the smallest rule number dominates.

PRIORITY

This statement, which appears in the header of a rule, is used to set a priority for the rule. Its syntax is:

```
PRIORITY priority_value
```

where priority_value is a REAL number. If no PRIORITY statement is given for a rule, the default value of 0.0 is used.

PRODCTN

This control statement turns production running on or off for forward-chaining. Its syntax is:

```
PRODCTN ON|OFF
```

If production running is ON, then all rules without false premises are included in the conflict set, regardless of whether they assert a value for an unknown attribute or not. On the other hand, if production running is OFF, then rules are only included in the conflict set if they produce a value for an attribute which is currently unknown. The default value is OFF.

RULE

This statement is used to begin a rule definition and give the rule its name. Its syntax is:

```
RULE rule_title
```

where **rule_title** is a character string of length 40 or less. If the rule's title is not blank, it is printed along with timer information every time the rule is fired.

THEN

This statement is used to define the first action clause for a given rule. Its syntax is:

```
THEN action attribute_A attribute_B attribute_C
```

where **attribute_A**, **attribute_B**, and **attribute_C** are the names of three attributes. Attribute names which are preceeded by a **%** sign indicate that the value of the attribute is set (asserted) by the action. The action is taken from the list:

```
NOP         no operation
SET         vA<==vB
GET         vA<==vB(offset=vC)  indexed fetch
PUT         vA(offset=vB)<==vC  indexed store
RESET       vA<==UNKNOWN
ADD         vA<==vB+vC
SUBTRACT    vA<==vB-vC
MULTIPLY    vA<==vB*vC
DIVIDE      vA<==vB/vC
ABS         vA<==ABS(vB)
SQRT        vA<==SQRT(vB)
```

EXPON	vA<==vB**vC
MIN	vA<==MIN(vB,vC)
MAX	vA<==MAX(vB,vC)
SHOW	Writes vA, vB, and vC to print file.
CONSIDER	Changes context to A.

where vA indicates the *value of attribute* A. GET and PUT are indexed operations which are used to access/store values in tables (which are set up using the INITIAL statement). In addition, the action may be the name of a subroutine call for a procedural element, in which case the meanings of the attributes are given by the procedure's function; the latter is the technique used to link the expert and procedural elements.

TRACE

This control statement selects the level of tracing which the inference engine outputs. Its syntax is:

```
TRACE 0|1|2|3|4
```

If 0 is chosen, no tracing is output and if 4 is chosen, the results of all rule examinations, conflict resolution, rule firings, etc. are output on the print unit. The default value is 0.

D.3 Example EXPROC Program

This section contains listing of the knowledge base used in the local compressible flow analysis example in chapter 7. The section begins with a copy of the knowledge base which was input to EXPROC, followed by the cross-reference map which EXPROC prints.

The final two sections contain the output which results from applying the forward-chaining and backward-chaining inference engines to the knowledge base. In the backward-chaining example, the initial goal is temp_0.

The output in the last three sections is slightly rearranged to make it fit properly on the pages.

D.3.1 Knowledge base

```
; Knowledge base for local compressible flow analysis - Version 2

TRACE   2
DYNAMIC ON
ORDER   UNKNOWNS PREMISES L_RECENT

INITIAL Rgas     +1716.   ;gas constant
INITIAL gamma    +1.40    ;ratio of specific heats

INITIAL pressure +2116.   ;pressure
INITIAL temp     +519.0   ;temperature
INITIAL massflow +3.715   ;mass flow rate
INITIAL area     +2.00    ;cross-sectional area

;       Mach             ;Mach number
;       density          ;density
;       temp_0           ;stagnation temperature
;       velocity         ;velocity
;       pres_0           ;stagnation pressure


RULE      pressure  (by state equation)
;                    pressure=density*Rgas*temp
IF      * NOP
THEN    MULTIPLY  #1        density   Rgas
ANDTHEN MULTIPLY  %pressure #1        temp

RULE      density   (by state equation)
;                    density=pressure/(Rgas*temp)
IF      * NOP
THEN    DIVIDE    #1        pressure  Rgas
ANDTHEN DIVIDE    %density  #1        temp

RULE      temp      (by state equation)
;                    temp=pressure/(density*Rgas)
IF      * NOP
THEN    DIVIDE    #1        pressure  density
ANDTHEN DIVIDE    %temp     #1        Rgas


RULE      sound     (by sound-relationship)
```

```
;                    sound=SQRT(gamma*Rgas*temp)
IF       * NOP
THEN     MULTIPLY  #1         gamma      Rgas
ANDTHEN  MULTIPLY  #1         #1         temp
ANDTHEN  SQRT      %sound     #1

RULE       temp       (by sound-relationship)
;                     temp=(sound**2)/(gamma*Rgas)
IF       * NOP
THEN     MULTIPLY  #1         sound      sound
ANDTHEN  DIVIDE    #1         #1         gamma
ANDTHEN  DIVIDE    %temp      #1         Rgas


RULE       massflow  (by mass-flow-definition)
;                     massflow=density*velocity*area
IF       * NOP
THEN     MULTIPLY  #1         density    velocity
ANDTHEN  MULTIPLY  %massflow  #1         area

RULE       density   (by mass-flow-definition)
;                     density=massflow/(velocity*area)
IF       * NOP
THEN     DIVIDE    #1         massflow   velocity
ANDTHEN  DIVIDE    %density   #1         area

RULE       velocity  (by mass-flow-definition)
;                     velocity=massflow/(velocity*area)
IF       * NOP
THEN     DIVIDE    #1         massflow   density
ANDTHEN  DIVIDE    %velocity  #1         area

RULE       area       (by mass-flow-definition)
;                     area=massflow/(velocity*density)
IF       * NOP
THEN     DIVIDE    #1         massflow   density
ANDTHEN  DIVIDE    %area      #1         velocity


RULE       Mach       (by Mach-definition)
;                     Mach=velocity/sound
IF       * NOP
THEN     DIVIDE    %Mach      velocity   sound

RULE       velocity  (by Mach-definition)
;                     velocity=Mach*sound
IF       * NOP
THEN     MULTIPLY  %velocity  Mach       sound
```

```
RULE      sound      (by Mach-definition)
;                     sound=velocity/Mach
IF        * NOP
THEN      DIVIDE     %sound     velocity  Mach


RULE      temp_0     (by energy equation)
;                     temp_0=temp*(1+((gamma-1)/2)*Mach**2)
IF        * NOP
THEN      SUBTRACT   #1         gamma     +1.0
ANDTHEN   DIVIDE     #1         #1        +2.0
ANDTHEN   MULTIPLY   #1         #1        Mach
ANDTHEN   MULTIPLY   #1         #1        Mach
ANDTHEN   ADD        #1         #1        +1.0
ANDTHEN   MULTIPLY   %temp_0    temp      #1

RULE      temp       (by energy equation)
;                     temp=temp_0/(1+((gamma-1)/2)*Mach**2)
IF        * NOP
THEN      SUBTRACT   #1         gamma     +1.0
ANDTHEN   DIVIDE     #1         #1        +2.0
ANDTHEN   MULTIPLY   #1         #1        Mach
ANDTHEN   MULTIPLY   #1         #1        Mach
ANDTHEN   ADD        #1         #1        +1.0
ANDTHEN   DIVIDE     %temp      temp_0    #1

RULE      Mach       (by energy equation)
;                     Mach=SQRT((2/(gamma-1))*(temp/temp_0-1))
IF        * NOP
THEN      DIVIDE     #1         temp_0    temp
ANDTHEN   SUBTRACT   #1         #1        +1.0
ANDTHEN   MULTIPLY   #1         #1        +2.0
ANDTHEN   SUBTRACT   #2         gamma     +1.0
ANDTHEN   DIVIDE     #1         #1        #2
ANDTHEN   SQRT       %Mach      #1


RULE      pres_0     (by isentropic relation)
;                     pres_0=pressure*((temp_0/temp)**(gamma/(gamma-1)))
IF        * NOP
THEN      DIVIDE     #1         temp_0    temp
ANDTHEN   SUBTRACT   #2         gamma     +1.0
ANDTHEN   DIVIDE     #2         gamma     #2
ANDTHEN   EXPON      #1         #1        #2
ANDTHEN   MULTIPLY   %pres_0    pressure  #1

RULE      pressure   (by isentropic relation)
```

```
;                    pressure=pres_0*((temp/temp_0)**(gamma/(gamma-1)))
IF       * NOP
THEN     DIVIDE     #1          temp        temp_0
ANDTHEN  SUBTRACT   #2          gamma       +1.0
ANDTHEN  DIVIDE     #2          gamma       #2
ANDTHEN  EXPON      #1          #1          #2
ANDTHEN  MULTIPLY   %pressure   pres_0      #1

RULE       temp_0     (by isentropic relation)
;                     temp_0=temp*((pres_0/pressure)**((gamma-1)/gamma))
IF       * NOP
THEN     DIVIDE     #1          pres_0      pressure
ANDTHEN  SUBTRACT   #2          gamma       +1.0
ANDTHEN  DIVIDE     #2          #2          gamma
ANDTHEN  EXPON      #1          #1          #2
ANDTHEN  MULTIPLY   %temp_0     temp        #1

RULE       temp       (by isentropic relation)
;                     temp=temp_0*((pressure/pres_0)**((gamma-1)/gamma))
IF       * NOP
THEN     DIVIDE     #1          pressure    pres_0
ANDTHEN  SUBTRACT   #2          gamma       +1.0
ANDTHEN  DIVIDE     #2          #2          gamma
ANDTHEN  EXPON      #1          #1          #2
ANDTHEN  MULTIPLY   %temp       temp_0      #1

END
```

D.3.2 Cross-reference map

This section contains the output of the EXPROC compiler for the knowledge base given above. It contains a listing of the attributes and their initial values, a copy of the rules, and a cross-reference listing of the attributes and contexts and where they are used.

```
EXPERT SYSTEM ATTRIBUTES:

|           |= 0.0000000E+00   |*          |= 0.0000000E+00
|#1         |= 0.0000000E+00   |#2         |= 0.0000000E+00
|#3         |= 0.0000000E+00   |#4         |= 0.0000000E+00
|UNKNOWN    |=-0.1234568E+39   |Rgas       |=  1716.000
|gamma      |=  1.400000       |pressure   |=  2116.000
|temp       |=  519.0000       |massflow   |=  3.715000
|area       |=  2.000000       |density    |=-0.1234568E+39
|sound      |=-0.1234568E+39   |velocity   |=-0.1234568E+39
|Mach       |=-0.1234568E+39   |+1.0       |=  1.000000
```

```
   |+2.0       |=  2.000000        |temp_0     |=-0.1234568E+39
   |pres_0     |=-0.1234568E+39

EXPERT SYSTEM CONTEXTS:

   |initial   |

EXPERT SYSTEM RULES:

Rule   1       :pressure  (by state equation)
    priority  :    0.0000
    context   :-- NONE --
          if  :*            NOP
          then:MULTIPLY     #1          density     Rgas
              :MULTIPLY    %pressure    #1          temp

Rule   2       :density   (by state equation)
    priority  :    0.0000
    context   :-- NONE --
          if  :*            NOP
          then:DIVIDE       #1          pressure    Rgas
              :DIVIDE      %density     #1          temp

Rule   3       :temp      (by state equation)
    priority  :    0.0000
    context   :-- NONE --
          if  :*            NOP
          then:DIVIDE       #1          pressure    density
              :DIVIDE      %temp        #1          Rgas

Rule   4       :sound     (by sound-relationship)
    priority  :    0.0000
    context   :-- NONE --
          if  :*            NOP
          then:MULTIPLY     #1          gamma       Rgas
              :MULTIPLY     #1          #1          temp
              :SQRT        %sound       #1

Rule   5       :temp      (by sound-relationship)
    priority  :    0.0000
    context   :-- NONE --
          if  :*            NOP
          then:MULTIPLY     #1          sound       sound
              :DIVIDE       #1          #1          gamma
              :DIVIDE      %temp        #1          Rgas

Rule   6       :massflow  (by mass-flow-definition)
    priority  :    0.0000
```

```
    context   :-- NONE --
         if   :*            NOP
         then:MULTIPLY      #1            density      velocity
              :MULTIPLY     %massflow     #1           area

Rule  7       :density  (by mass-flow-definition)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:DIVIDE        #1            massflow     velocity
              :DIVIDE       %density      #1           area

Rule  8       :velocity  (by mass-flow-definition)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:DIVIDE        #1            massflow     density
              :DIVIDE       %velocity     #1           area

Rule  9       :area      (by mass-flow-definition)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:DIVIDE        #1            massflow     density
              :DIVIDE       %area         #1           velocity

Rule  10      :Mach      (by Mach-definition)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:DIVIDE       %Mach          velocity     sound

Rule  11      :velocity  (by Mach-definition)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:MULTIPLY     %velocity      Mach         sound

Rule  12      :sound     (by Mach-definition)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:DIVIDE       %sound         velocity     Mach

Rule  13      :temp_0    (by energy equation)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
```

```
        then:SUBTRACT     #1          gamma       +1.0
            :DIVIDE       #1          #1          +2.0
            :MULTIPLY     #1          #1          Mach
            :MULTIPLY     #1          #1          Mach
            :ADD          #1          #1          +1.0
            :MULTIPLY     %temp_0     temp        #1

Rule 14     :temp      (by energy equation)
    priority :    0.0000
    context  :-- NONE --
        if  :*            NOP
        then:SUBTRACT     #1          gamma       +1.0
            :DIVIDE       #1          #1          +2.0
            :MULTIPLY     #1          #1          Mach
            :MULTIPLY     #1          #1          Mach
            :ADD          #1          #1          +1.0
            :DIVIDE       %temp       temp_0      #1

Rule 15     :Mach      (by energy equation)
    priority :    0.0000
    context  :-- NONE --
        if  :*            NOP
        then:DIVIDE       #1          temp_0      temp
            :SUBTRACT     #1          #1          +1.0
            :MULTIPLY     #1          #1          +2.0
            :SUBTRACT     #2          gamma       +1.0
            :DIVIDE       #1          #1          #2
            :SQRT         %Mach       #1

Rule 16     :pres_0    (by isentropic relation)
    priority :    0.0000
    context  :-- NONE --
        if  :*            NOP
        then:DIVIDE       #1          temp_0      temp
            :SUBTRACT     #2          gamma       +1.0
            :DIVIDE       #2          gamma       #2
            :EXPON        #1          #1          #2
            :MULTIPLY     %pres_0     pressure    #1

Rule 17     :pressure  (by isentropic relation)
    priority :    0.0000
    context  :-- NONE --
        if  :*            NOP
        then:DIVIDE       #1          temp        temp_0
            :SUBTRACT     #2          gamma       +1.0
            :DIVIDE       #2          gamma       #2
            :EXPON        #1          #1          #2
            :MULTIPLY     %pressure   pres_0      #1
```

```
Rule  18      :temp_0    (by isentropic relation)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:DIVIDE        #1          pres_0      pressure
              :SUBTRACT     #2          gamma       +1.0
              :DIVIDE       #2          #2          gamma
              :EXPON        #1          #1          #2
              :MULTIPLY     %temp_0     temp        #1

Rule  19      :temp      (by isentropic relation)
    priority  :    0.0000
    context   :-- NONE --
         if   :*            NOP
         then:DIVIDE        #1          pressure    pres_0
              :SUBTRACT     #2          gamma       +1.0
              :DIVIDE       #2          #2          gamma
              :EXPON        #1          #1          #2
              :MULTIPLY     %temp       temp_0      #1

CROSS REFERENCE TABLE (ATTRIBUTES):
Attribute:Rgas          1(A 1)   2(A 1)   3(A 2)   4(A 1)   5(A 3)
Attribute:gamma         4(A 1)   5(A 2)  13(A 1)  14(A 1)  15(A 4)
                       16(A 2)  16(A 3)  17(A 2)  17(A 3)  18(A 2)
                       18(A 3)  19(A 2)  19(A 3)
Attribute:pressure      1(A 2)   2(A 1)   3(A 1)  16(A 5)  17(A 5)
                       18(A 1)  19(A 1)
Attribute:temp          1(A 2)   2(A 2)   3(A 2)   4(A 2)   5(A 3)
                       13(A 6)  14(A 6)  15(A 1)  16(A 1)  17(A 1)
                       18(A 5)  19(A 5)
Attribute:massflow      6(A 2)   7(A 1)   8(A 1)   9(A 1)
Attribute:area          6(A 2)   7(A 2)   8(A 2)   9(A 2)
Attribute:density       1(A 1)   2(A 2)   3(A 1)   6(A 1)   7(A 2)
                        8(A 1)   9(A 1)
Attribute:sound         4(A 3)   5(A 1)  10(A 1)  11(A 1)  12(A 1)
Attribute:velocity      6(A 1)   7(A 1)   8(A 2)   9(A 2)  10(A 1)
                       11(A 1)  12(A 1)
Attribute:Mach         10(A 1)  11(A 1)  12(A 1)  13(A 3)  13(A 4)
                       14(A 3)  14(A 4)  15(A 3)
Attribute:+1.0         13(A 1)  13(A 5)  14(A 1)  14(A 5)  15(A 2)
                       15(A 4)  16(A 2)  17(A 2)  18(A 2)  19(A 2)
Attribute:+2.0         13(A 2)  14(A 2)  15(A 3)
Attribute:temp_0       13(A 6)  14(A 6)  15(A 1)  16(A 1)  17(A 1)
                       18(A 5)  19(A 5)
Attribute:pres_0       16(A 5)  17(A 5)  18(A 1)  19(A 1)

CROSS REFERENCE TABLE (CONTEXTS):
```

```
Context  :initial
```

D.3.3 Forward-chaining output

This section contains a listing of the output file produced by the forward-chaining inference engine, **EXFWRD**, followed by a list of the attributes and their values after the forward-chainer has completed. It should be noted that the output has been modified slightly to make it fit on the printed page.

```
FORWARD CHAINING EXPERT SYSTEM INVOKED


RULES ORDERED BY (MAJOR TO MINOR):UNKNOWNS  PREMISES  L_RECENT  NUMERIC
DYNAMIC RULE ORDERING             :ON
PRODUCTION-TYPE RUN               :OFF

 EX-TRACE>> RULE STACK AFTER PREVIEWING AND ORDERING  (CONTEXT=initial   )
 EX-TRACE>>         4    2
 EX-TRACE>> FIRING RULE    2
**TIMER** INCREMENTAL CPU TIME =     0.91 SEC
               TOTAL CPU TIME =     1.83 SEC
                                   density   (by state equation)
 EX-TRACE>> RULE STACK AFTER PREVIEWING AND ORDERING  (CONTEXT=initial   )
 EX-TRACE>>         8    4
 EX-TRACE>> FIRING RULE    4
**TIMER** INCREMENTAL CPU TIME =     0.04 SEC
               TOTAL CPU TIME =     1.92 SEC
                                   sound     (by sound-relationship)
 EX-TRACE>> RULE STACK AFTER PREVIEWING AND ORDERING  (CONTEXT=initial   )
 EX-TRACE>>         8
 EX-TRACE>> FIRING RULE    8
**TIMER** INCREMENTAL CPU TIME =     0.03 SEC
               TOTAL CPU TIME =     1.95 SEC
                                   velocity  (by mass-flow-definition)
 EX-TRACE>> RULE STACK AFTER PREVIEWING AND ORDERING  (CONTEXT=initial   )
 EX-TRACE>>        10
 EX-TRACE>> FIRING RULE   10
**TIMER** INCREMENTAL CPU TIME =     0.03 SEC
               TOTAL CPU TIME =     1.98 SEC
                                   Mach      (by Mach-definition)
 EX-TRACE>> RULE STACK AFTER PREVIEWING AND ORDERING  (CONTEXT=initial   )
 EX-TRACE>>        13
 EX-TRACE>> FIRING RULE   13
**TIMER** INCREMENTAL CPU TIME =     0.03 SEC
```

```
                TOTAL CPU TIME =      2.01 SEC
                                    temp_0    (by energy equation)
 EX-TRACE>> RULE STACK AFTER PREVIEWING AND ORDERING  (CONTEXT=initial   )
 EX-TRACE>>        16
 EX-TRACE>> FIRING RULE   16
**TIMER** INCREMENTAL CPU TIME =      0.03 SEC
                TOTAL CPU TIME =      2.04 SEC
                                    pres_0    (by isentropic relation)
EXPERT SYSTEM TERMINATING

FINAL CONTEXT: initial
STATISTICS   : NUMBER OF RULES EXAMINED  =  133
               NUMBER OF RULES TRIGGERED =  133
               NUMBER OF RULES FIRED     =    8
               LONGEST STACK LENGTH      =    2

EXPERT SYSTEM ATTRIBUTES:

   |           |= 0.0000000E+00  |*          |= 0.0000000E+00
   |#1         |=  1.387288      |#2         |=  3.500000
   |#3         |= 0.0000000E+00  |#4         |= 0.0000000E+00
   |UNKNOWN    |=-0.1234568E+30  |Rgas       |=  1716.000
   |gamma      |=  1.400000      |pressure   |=  2116.000
   |temp       |=  519.0000      |massflow   |=  3.715000
   |area       |=  2.000000      |density    |= 0.2375916E-02
   |sound      |=  1116.622      |velocity   |=  781.8038
   |Mach       |= 0.7001507      |+1.0       |=  1.000000
   |+2.0       |=  2.000000      |temp_0     |=  569.8839
   |pres_0     |=  2935.502
```

D.3.4 Backward-chaining output

This section contains a listing of the output file produced by the backward-chaining inference engine, EXBACK, starting with temp_0 as the goal. That is followed by a list of the attributes and their values after the backward-chainer has completed. It should be noted that the output has been modified slightly to make it fit on the printed page.

```
 BACKWARD CHAINING EXPERT SYSTEM INVOKED


 RULES ORDERED BY (MAJOR TO MINOR):UNKNOWNS  PREMISES  L_RECENT  NUMERIC

  EX-TRACE>> CURRENT GOAL STACK:  -20
```

```
 EX-TRACE>> CURRENT RULE STACK:   18   13
 EX-TRACE>> ADDING    1 GOALS TO STACK
 EX-TRACE>> CURRENT GOAL STACK:   20  -17
 EX-TRACE>> CURRENT RULE STACK:   18   13   10
 EX-TRACE>> ADDING    2 GOALS TO STACK
 EX-TRACE>> CURRENT GOAL STACK:   20   17  -16  -15
 EX-TRACE>> CURRENT RULE STACK:   18   13   10    4
 EX-TRACE>> FIRING RULE    4
**TIMER** INCREMENTAL CPU TIME =     0.13 SEC
               TOTAL CPU TIME =     3.04 SEC
                                    sound     (by sound-relationship)
 EX-TRACE>> DELETING RULE    4 FROM STACK
 EX-TRACE>> CURRENT GOAL STACK:   20   17  -16   15
 EX-TRACE>> CURRENT GOAL STACK:   20   17  -16
 EX-TRACE>> CURRENT RULE STACK:   18   13   10    8
 EX-TRACE>> ADDING    1 GOALS TO STACK
 EX-TRACE>> CURRENT GOAL STACK:   20   17   16  -14
 EX-TRACE>> CURRENT RULE STACK:   18   13   10    8    2
 EX-TRACE>> FIRING RULE    2
**TIMER** INCREMENTAL CPU TIME =     0.07 SEC
               TOTAL CPU TIME =     3.11 SEC
                                    density   (by state equation)
 EX-TRACE>> DELETING RULE    2 FROM STACK
 EX-TRACE>> CURRENT GOAL STACK:   20   17   16   14
 EX-TRACE>> CURRENT GOAL STACK:   20   17   16
 EX-TRACE>> FIRING RULE    8
**TIMER** INCREMENTAL CPU TIME =     0.02 SEC
               TOTAL CPU TIME =     3.13 SEC
                                    velocity  (by mass-flow-definition)
 EX-TRACE>> DELETING RULE    8 FROM STACK
 EX-TRACE>> CURRENT GOAL STACK:   20   17   16
 EX-TRACE>> CURRENT GOAL STACK:   20   17
 EX-TRACE>> FIRING RULE   10
**TIMER** INCREMENTAL CPU TIME =     0.01 SEC
               TOTAL CPU TIME =     3.14 SEC
                                    Mach      (by Mach-definition)
 EX-TRACE>> DELETING RULE   10 FROM STACK
 EX-TRACE>> CURRENT GOAL STACK:   20   17
 EX-TRACE>> CURRENT GOAL STACK:   20
 EX-TRACE>> FIRING RULE   13
**TIMER** INCREMENTAL CPU TIME =     0.02 SEC
               TOTAL CPU TIME =     3.16 SEC
                                    temp_0    (by energy equation)
 EX-TRACE>> DELETING RULE   13 FROM STACK
 EX-TRACE>> DELETING RULE   18 FROM STACK
 EX-TRACE>> CURRENT GOAL STACK:   20
EXPERT SYSTEM TERMINATING
```

```
BACKWARD CHAINING DEDUCED THAT temp_0    =   569.8839

FINAL CONTEXT: initial
STATISTICS   : NUMBER OF RULES EXAMINED  =  103
               NUMBER OF RULES TRIGGERED =  103
               NUMBER OF RULES FIRED     =    5
               LONGEST STACK LENGTH      =    0

EXPERT SYSTEM ATTRIBUTES:

   |            |= 0.0000000E+00   |*           |= 0.0000000E+00
   |#1          |=  1.098042       |#2          |= 0.0000000E+00
   |#3          |= 0.0000000E+00   |#4          |= 0.0000000E+00
   |UNKNOWN     |=-0.1234568E+39   |Rgas        |=  1716.000
   |gamma       |=  1.400000       |pressure    |=  2116.000
   |temp        |=  519.0000       |massflow    |=  3.715000
   |area        |=  2.000000       |density     |= 0.2375916E-02
   |sound       |=  1116.622       |velocity    |=  781.8038
   |Mach        |= 0.7001507       |+1.0        |=  1.000000
   |+2.0        |=  2.000000       |temp_0      |=  569.8839
   |pres_0      |=-0.1234568E+39
```