RICE UNIVERSITY

AN ORTHOGONAL ADAPTIVE GRID MODULE TO COMPLEMENT EXISTING FLUID DYNAMICS AND HEAT TRANSFER CODES

by

MATTHEW R. BARRY

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

APPROVED, THESIS COMMITTEE:

Dr. F. A. Wierum, Professor of Mechanical and Aerospace Engineering, Director

Dr. J. E. Akin, Professor of Mechanical Engineering

Dr. A. J. Chapman, Professor of Mechanical Engineering

Houston, Texas

April, 1985

3 1272 00289 0844
AN ORTHOGONAL ADAPTIVE GRID MODULE TO COMPLEMENT EXISTING FLUID DYNAMICS AND HEAT TRANSFER CODES

MATTHEW R. BARRY

ABSTRACT

A versatile, orthogonal adaptive grid scheme for two-dimensional numerical fluid dynamics and heat transfer problems is presented. The scheme employs a one-dimensional adaptation sweep to define one family of physical grid lines. A second sweep then uses a technique called AOT, developed herein, to fit an orthogonal family of lines to the solution-adapted lines. These procedures are fast, require little core storage, and do not change the original domain boundaries.

Each subroutine developed performs a specific function and together they form the adaptive grid module. The versatility of these subroutines allows their usage in various combinations and in either grid direction. Modifications required to implement this scheme into existing fluid dynamics and heat transfer codes, therefore, are minimal.

Code documentation and sample applications showing the implementation and versatility of the scheme are presented.
ACKNOWLEDGEMENT

I am particularly indebted to Dr. C. P. Li of the NASA Johnson Space Center for suggesting the topic and for allowing the use of his codes and computer time. I also wish to thank Dr. Li and the other members of the Aerothermodynamics branch at NASA/JSC for their guidance and patience while we implemented example applications on a computer system with which I was unfamiliar.

Appreciation is also in order to the members of the faculty committee for their guidance during the research and development of the work, and for their editing of the manuscript.

Finally, I would like to thank my family and friends for their interest and moral support provided during this research.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>First derivative multiplier</td>
</tr>
<tr>
<td>β</td>
<td>Second derivative multiplier</td>
</tr>
<tr>
<td>ε</td>
<td>Area coordinate</td>
</tr>
<tr>
<td>ξ</td>
<td>Computational grid coordinate</td>
</tr>
<tr>
<td>η</td>
<td>Computational grid coordinate</td>
</tr>
<tr>
<td>φ</td>
<td>Interpolation variable</td>
</tr>
<tr>
<td>g</td>
<td>Second derivative control function</td>
</tr>
<tr>
<td>H</td>
<td>Interpolation functions</td>
</tr>
<tr>
<td>N</td>
<td>Arc length</td>
</tr>
<tr>
<td>P</td>
<td>Dependent variable for adaptation</td>
</tr>
<tr>
<td>R₁</td>
<td>First derivative allocation parameter</td>
</tr>
<tr>
<td>R₂</td>
<td>Second derivative allocation parameter</td>
</tr>
<tr>
<td>S</td>
<td>Arc length</td>
</tr>
<tr>
<td>w</td>
<td>Weight function</td>
</tr>
<tr>
<td>x</td>
<td>Physical grid coordinate</td>
</tr>
<tr>
<td>y</td>
<td>Physical grid coordinate</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Computational Space Transformation</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Adaptation Marching Technique</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>Approximate Orthogonal Trajectory Process</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Area Coordinate Interpolation</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>Subroutine Interaction</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>Bracketing for Adaptation Interpolation</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>Node Numbers for Triangular Interpolation</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>Initial Grid for Test Cases</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>Surface Plot for Test Case 1</td>
<td>63</td>
</tr>
<tr>
<td>10</td>
<td>Adapted Grids for Test Case 1</td>
<td>65</td>
</tr>
<tr>
<td>11</td>
<td>Surface Plot for Test Case 2</td>
<td>68</td>
</tr>
<tr>
<td>12</td>
<td>Adapted Grids for Test Case 2</td>
<td>69</td>
</tr>
<tr>
<td>13</td>
<td>Typical Subroutine CALL Sequence</td>
<td>71</td>
</tr>
<tr>
<td>14</td>
<td>Shock Reflection Problem</td>
<td>72</td>
</tr>
<tr>
<td>15</td>
<td>Example Problem, Fixed Grid Solutions</td>
<td>75</td>
</tr>
<tr>
<td>16</td>
<td>Example Problem, Adaptive Grid Solutions</td>
<td>85</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Implementation Parameters for Test Cases</td>
<td>64</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgement</td>
<td>iii</td>
</tr>
<tr>
<td>Nomenclature</td>
<td>iv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vii</td>
</tr>
</tbody>
</table>

I. INTRODUCTION TO THE PROBLEM   1

II. THEORETICAL DEVELOPMENT     7
   II-1. Direction Control      7
   II-2. Grid Adaptation Sweep 8
   II-3. Orthogonalization Sweep 22
   II-4. Interpolation Onto New Grid 27

III. NUMERICAL SOLUTION        31
   III-1. Direction Control    35
   III-2. Grid Adaptation Sweep 36
   III-3. Orthogonalization Sweep 47
   III-4. Interpolation Onto New Grid 54

IV. EXAMPLES                   61
   IV-1. Test Cases            61
   IV-2. Implementation in Existing Codes 67
I. INTRODUCTION TO THE PROBLEM

Numerical methods for solving the governing partial differential equations of fluid dynamics and heat transfer have received a great amount of attention in recent years. Various techniques for increasing the accuracy of these numerical approximations have been developed. One of the most important factors involved in an accurate numerical solution is the choice of an appropriate coordinate system.

When the governing partial differential equations are discretized, a large number of grid points are required in order to reduce the error in the solution approximation. This requires large amounts of computer storage. If a knowledge of where the approximation is inadequate in the physical space is available a priori, the initial grid chosen may concentrate points in these regions, leaving the adequately approximated regions relatively sparse. However, if these regions are not known a priori, some method whereby the grid may updated during the solution is required.

Techniques for dynamically redefining the physical grid points during the course of the numerical solution of partial differential equations are called adaptive grid methods. Dwyer [1] presents criteria for the use of these methods. These techniques provide an increased mesh point
density in areas of the physical plane where large gradients, errors, or other solution extrema occur, at the expense of removing points from the lower gradient or more well-behaved regions.

As Anderson [2] notes, the use of an adaptive grid method implies that the partial differential equation governing some physical process is being computed by an iterative or marching technique. Anderson's example of a typical problem is described as follows. A partial differential equation governing a problem is transformed from an arbitrary physical domain into an evenly spaced computational domain. For instance, consider the one-dimensional first-order wave equation,

$$\frac{\partial u}{\partial t} + (c)\frac{\partial u}{\partial x} = 0$$

with $u(x,t)$ being the unknown dependent variable and $c$ the constant wave speed. For a finite difference solution, the physical domain is discretized by a fixed and finite number of grid points. In many problems, the occurrence of rapid variations of $u$ in certain portions of the solution domain may result in inaccuracies when the original grid used is fixed and unchanging. For such a case, a transformation into a different computational domain during the course of solution may provide a better distribution of grid points. For example, the transformation $\tau = t$ and $\xi = \xi(x,t)$ where $\xi$ and $\tau$ are the new computational coordinates allows uniformly
spaced grid points in \((\xi, \tau)\) to correspond to non-uniform spacing in the \((x,t)\) - space. The wave equation is then rewritten as

\[
\partial u/\partial \tau + (\xi_t + c\xi_x)\partial u/\partial \xi = 0
\]

where \(\xi_t\) and \(\xi_x\) relate the physical and the transformed spaces. For this transformation, it is possible to write

\[
\xi_t = -x_{\tau} / x_{\xi},
\]

and

\[
\xi_x = 1/x_{\xi}.
\]

Substitution of these relations into the wave equation results in

\[
\partial u/\partial \tau - ((c^2 - c)/x_{\xi})\partial u/\partial \xi = 0
\]

which is to be solved in the uniform \((\xi, \tau)\) - space. The metric term \(x_{\xi}\) relates the ratio of arc lengths in the physical and computational domains along lines of constant \(\tau\). When a fixed grid is used, \(x_{\xi}\) is constant and \(x_{\tau}\) is zero. However, if the grid is adjusted after a predetermined number of iterations, as is the case with an adaptive grid, \(x_{\tau}\) (called the grid speed), is non-zero due to the changing grid geometry, and the metric term \(x_{\xi}\) is not constant and must be recalculated since the grid has been altered. The iteration process is then continued. The temporal metric or grid speed term, \(x_{\tau}\), thereby provides the coupling of the moving or adaptive grid with the numerical solution. The adaptive grid scheme provides a means for estimating these terms in the transformed equation. Thus, the adaptive grid
process is an intermediate step used to reassign existing grid point locations according to the developing solution of the partial differential equations. Several other illustrative examples of the use of adaptive grids are given by Dwyer [3], Dwyer et al [4], Rai and Anderson [5], and Anderson and Rai [6].

Adaptive grids are generally more effective in the solution of larger systems of partial differential equations, such as the 2- or 3-dimensional forms of the Euler or Navier-Stokes equations. Many numerical solutions of these systems have been developed in finite difference form, such as those described in papers by Briley and McDonald [7], Beam and Warming [8], Steger and Warming [9], and Li [10,11]. Excellent presentations of many existing methods are given by Anderson et al [12], including a discussion of the use of adaptive grids. Adaptive grid schemes should further enhance the accuracy and operation of these numerical solutions.

This thesis presents a technique whereby an orthogonal adaptive grid module may be added to existing fluid dynamics or heat transfer codes. The module consists of several FORTRAN subroutines which may be invoked by the simple addition of CALL statements within a fixed-grid problem code. These subroutines were designed with versatility and ease of implementation in mind. The goals from which this
adaptive grid module was developed were straightforward:

(1) The module should operate on the two dimensional simply-connected grids initially defined by some external grid generation procedure.

(2) The module should be able to operate with any dependent variable defined by the problem code at each of the grid nodes in the physical space.

(3) The code developed should have sub-modules or subroutines which may be easily modified or which are interchangeable with other schemes.

(4) The module should operate in either the $x$ or $y$ physical space direction without code modification.

(5) In order to be economical, the module should operate in a minimum amount of CPU time, and the amount of extra data storage required at each node should be small.

(6) Extension to 3-dimensions or reduction to 1-dimension should be easily achieved.

In consideration of the above goals, the orthogonal adaptive grid module developed herein consists of several components. Each component performs a specific function, and thus each is interchangeable. These components consist of a pair of unidirectional sweeps, one for grid adaptation
and one for orthogonalization, a sweep direction control component, and an interpolation component. Throughout the following discussion, emphasis will be placed on the numerical techniques used in the individual routines.
II. THEORETICAL DEVELOPMENT

II-1. Direction Control

The goal of allowing use of the adaptation subroutines in either space direction is accomplished by designing a direction control subroutine. The choice of the space location \((\xi, \eta)\) is analogous to choosing the array indices \((I,J)\). Hence, to allow adaptation in both \(\xi\) and \(\eta\) directions, this subroutine furnishes correct array indices based on the direction of adaptation. This allows operation of one block of code in either direction by returning \(I\) and \(J\) as a function of a direction indicator variable \(IDIR\) and the loop counters \(M\) and \(N\). The \(M\) value designates a point on the current adaptation line specified by \(N\). This also allows proper boundary point differencing without adding code wherever boundaries may cause numerical differencing or array accessing problems.

The purpose of this routine is to provide values for \(I, J, IP=I+1, JP=J+1, IM=I-1,\) and \(JM=J-1\). If the current adaptation direction is the \(J\) direction, for example, the values of \(I+1\) and \(I-1\) default to \(I\). Similarly, for adaptation in the \(I\) direction, \(J+1\) and \(J-1\) default to \(J\). This allows the indexing of arrays such as \(P(IM, JM)\), with actual meaning \(P(I, J-1)\) for \(IDIR=1\) or \(P(I-1, J)\) for \(IDIR=0\). Thus, one block of code using these dummy indices replaces
two blocks of code which operate in specific directions. This reduces the size of the subroutines.

II-2. Grid Adaptation Sweep

In order to accurately resolve steep gradients in a dependent variable over some region in the physical plane it is necessary to adapt the grid after some multiple of solution iterations by the main program. These high-gradient regions are generally thin layers such as shocks, flames or boundary layers, and may occur at an arbitrary orientation in the internal grid geometry. The adaptation process uses the variable data defined at the grid node points and redefines the physical grid points $x$ and $y$ in order to cluster them near these high-gradient regions. Several methods have been derived to perform this adaptation.

Thompson [13] presents a survey of several dynamically adaptive grid methods of various derivation and application. In general, the physical domain $(x,y)$ is mapped onto an equally spaced and stationary computational domain $(\xi,\eta)$ (Fig.1) so that

$$
\begin{bmatrix}
x \\
y
\end{bmatrix} =
\begin{bmatrix}
x(\xi,\eta) \\
y(\xi,\eta)
\end{bmatrix}
$$

When the inverse mapping also exists, then
FIG (1)

PHYSICAL TO COMPUTATIONAL SPACE TRANSFORMATION
In differential notation,
\[
\begin{bmatrix}
\frac{dx}{d\xi} & \frac{dy}{d\xi} \\
\frac{dx}{d\eta} & \frac{dy}{d\eta}
\end{bmatrix}
= \begin{bmatrix} x_\xi & x_\eta \\
y_\xi & y_\eta \end{bmatrix} \begin{bmatrix} d\xi \\
d\eta \end{bmatrix}
\]
or
\[
\begin{bmatrix}
\frac{d\xi}{d\eta} \\
\frac{d\eta}{d\xi}
\end{bmatrix}
= \frac{1}{J} \begin{bmatrix} y_\eta & -x_\eta \\
y_\xi & x_\xi \end{bmatrix} \begin{bmatrix} dx \\
dy \end{bmatrix}
\]
where \( J \) is the Jacobian of the transformation and is defined by
\[
J = x_\xi y_\eta - x_\eta y_\xi.
\]
The partial differential equations governing the solution are transformed so that the independent variables are now \( \xi \) and \( \eta \), and calculations are performed on this computational plane.

Anderson [14] groups the current grid adaptation techniques into two classes. Type A schemes use a mathematical rule to construct the transformation to be solved for the new points \( x \) and \( y \). The temporal metric terms are then evaluated by upwind (backward in time) differences between the old and new grid points. Type B schemes use a postulation for the grid point distribution to solve directly for the grid speeds, which are then integrated with respect to time to obtain the new \( x \) and \( y \) coordinates. This integration is generally coupled with the
PDE solution, and the new metric terms may then be calculated.

Included in the type A methods are the variational schemes. A two-dimensional adaptation scheme based on a variational approach was applied by Brackbill and Saltzman [15]. This technique considers properties of the computational mesh which are generated by the differential properties of the mapping between physical and computational space [15,16]. The variation in mesh spacing along constant computational lines is given by the gradients $\nabla \xi$ and $\nabla \eta$. Thus the measure of grid smoothness is given by

$$(\nabla \xi)^2 + (\nabla \eta)^2,$$

and the orthogonality of line intersections is given by the scalar product of these gradients,

$$\nabla \xi \cdot \nabla \eta$$

which is zero if the lines are orthogonal. The volume (area) of cells in the computational space is related to the Jacobian of the mapping, given in two dimensions by

$$J = x_\xi \, y_\eta - x_\eta \, y_\xi.$$

A weighted measure of cell volume may be determined by multiplying the Jacobian by some weight function $w(\xi, \eta)$. Similarly, a volume weighted measure of orthogonality may be determined by multiplying the Jacobian by the orthogonality measure.
In this method, a linear combination of three integrals is minimized. These integrals are
\[ I_s = \int \int \left( x_e^2 + x_n^2 + y_e^2 + y_n^2 \right) / J \, d\xi d\eta, \]  
\[ I_v = \int \int w J^3 d\xi d\eta, \]  
\[ I_o = \int \int (x_e x_n + y_e y_n)^2 d\xi d\eta, \]
and
where \( J \) is the Jacobian and \( w \) is a weight function. These integrals, then, represent measures of grid smoothness, cell volume variation, and orthogonality, respectively. The minimization of a linear combination of these integrals results in the minimization of
\[ I = I_s + \lambda_v I_v + \lambda_o I_o, \]
where \( \lambda_v \) and \( \lambda_o \) are positive constants. The minimization formulation is obtained by determining a pair of Euler-Lagrange equations for each of the individual integrals. The solution is then obtained by solving a system of six quasi-linear partial differential equations for the grid points \( x \) and \( y \). These equations are given in [15] and will not be repeated here.

An example of a type B method is presented by Anderson and Rai [6], and Rai and Anderson [17]. This method is based on an attraction approach which distributes physical grid points in order to equidistribute some measure of error in a computed time-asymptotic solution. In two dimensions, with \( u \) an error measure or physical quantity, the method is
represented by the equations

\[(\xi_{i,j}) = K1 \{ \sum_{k=1}^{M} \sum_{l=1}^{N} [ |u_{x}|_{k,l} - |u_{x}|_{av} ]/r^n - \sum_{k=1}^{M} [ |u_{x}|_{k,l} - |u_{x}|_{av} ]/r^n \} \]

and

\[(\eta_{i,j}) = K2 \{ \sum_{k=1}^{M} \sum_{l=1}^{N} [ |u_{y}|_{k,l} - |u_{y}|_{av} ]/r^n - \sum_{k=1}^{M} [ |u_{y}|_{k,l} - |u_{y}|_{av} ]/r^n \} \]

where

\[r = \sqrt{(i-k)^2 + (j-1)^2} \]

and \(K1, K2,\) and \(n\) are constants, \(N\) is the number of \(\xi\) points, \(M\) is the number of \(\eta\) points, and \(av\) means average over the domain. The temporal metric terms (grid speeds) are then found from

\[(x_{\tau})_{i,j} = [(\eta_{y})_{i,j} \ (\xi_{i,j})_{\tau} - (\xi_{y})_{i,j} \ (\eta_{i,j})_{\tau} ]/J \]

and

\[(y_{\tau})_{i,j} = [(\xi_{x})_{i,j} \ (\eta_{i,j})_{\tau} - (\eta_{x})_{i,j} \ (\xi_{i,j})_{\tau} ]/J \]

where \(J\) is the Jacobian defined previously. These grid speed equations are simply integrated with respect to time in order to determine the new physical grid point distribution.

Determination of the error measure or other control for grid point spacing for use in the type B methods is generally difficult. This error measure is problem dependent and not readily defined in general terms, but the solution is readily obtained once a specific error measure is properly formulated. The type A schemes are derived from
sound mathematical foundations, and may be used on a variety of problems, but generally entail the solution of additional PDE's such as those in the Brackbill and Saltzman approach. However, the generality and sound mathematical background of the variational methods have led to several smaller scale derivations.

The present adaptation scheme is based on these variational derivations. Many two-dimensional problems are suitable for one-dimensional adaptation, although selection of which computational direction to adapt is based largely on intuition. Brackbill and Saltzman [15] show that the error due to solution approximation is minimized by minimizing the integral

\[ I(S) = \int_{0}^{\xi_2} w'(S) S_\xi d\xi \]

where \( S \) is the arc length measured along a constant \( \eta \) line (Fig.1) and the integration limits correspond to \( \xi_1 = 0 \) and \( \xi_2 = 1 \). This is the one-dimensional form of the volume variation integral given in equation (1b). The Euler-Lagrange equation corresponding to this integral is

\[ \frac{\partial}{\partial \xi} \left( \frac{\partial}{\partial S} \frac{\partial}{\partial \xi S} \right) (w' S_\xi) = 0 \]

which reduces to

\[ w'(\xi)^\eta S_\xi = \text{constant}. \]  \( (2) \)

Replacing \( w'(\xi)^\eta \) by \( w(\xi) \) for convenience, and with the boundary conditions

\[ S(\xi_1) = 0 \]
and

\[ S(\xi_{2}) = 1 \]

the solution of (2) is given by

\[ \frac{S}{\text{S}_{\text{max}}} \left[ \int_{\eta}^{\xi} (1/w(\xi)) \, d\xi \right] \]

\[ \left( 1 + \int_{\eta}^{\xi} (1/w(\xi)) \, d\xi \right) \]

(3)

where \( w(\xi) \) is a weight function. Thus grid points are distributed along arcs \( S(\xi) \) in physical space for each \( n=\text{constant} \) line in proportion to some weight function \( w(\xi) \). It is for this reason that schemes of this type are called equidistribution schemes.

In order to allow for adaptation in both gradient regions and solution extrema regions, the form of the weight function is chosen to be

\[ w(\xi) = 1 + \alpha |\partial p/\partial \xi| + \beta |g(\partial^2 p/\partial \xi^2)| \]

where \( \alpha \) and \( \beta \) are constant multipliers, \( p \) is the adaptation-dependent variable, and \( g \) is a control function for the second derivative term, described later. The equidistribution principal requires that a large value of the weight function correspond to a compressed grid point distribution. It may be seen that the weight function above will have a large value at points where the first or second derivatives of \( p \) are large, and will thus satisfy this requirement. The multipliers \( \alpha \) and \( \beta \) control the emphasis of the respective terms in the adaptation process.
It may be desired to put some physical meaning into the selection of these constants, rather than choosing them by trial and error. Dwyer [3] uses the above form of the weight function and defines two allocation parameters R1 and R2 to determine \( \alpha \) and \( \beta \) based on a relative emphasis of each weight function term in the adaptation process. These allocation parameters are defined in the range zero to one, with zero allowing no emphasis on that term, and one allowing full concentration. For this form of the arc length equation, the parameter \( R1 \) is the allocation factor for point concentration to the first derivative term, given by

\[
R1 = \frac{\int \alpha |\partial p/\partial \xi| d\xi}{1 + \int \alpha |\partial p/\partial \xi| d\xi}. \tag{4a}
\]

Similarly, \( R2 \) is the allocation factor for large second derivative concentrations, given by

\[
R2 = \frac{\int \beta |g(\partial^2 p/\partial \xi^2)| d\xi}{1 + \int \alpha |\partial p/\partial \xi| d\xi + \int \beta |g(\partial^2 p/\partial \xi^2)| d\xi}. \tag{4b}
\]

If \( R1, R2, \alpha \) and \( \beta \) are considered constant along an \( \eta \)-constant line, equations (4a) and (4b) may be solved for the weight function parameters \( \alpha \) and \( \beta \).

Including the second derivative term in the weight function is inspired by the fact that this term will also allow concentration of points near solution extrema, whereas the first derivative term alone will not. The function \( g \) is used primarily to control possible grid oscillations caused by the use of this second derivative term. Dwyer considers
three expressions for $g$, and states that the most generally applicable form for this function may be the difference between the first-order forward and backward first derivatives of $p$ with respect to $\xi$ given by

$$g = \left| \frac{P_{l+1} - P_l}{\Delta \xi} - \frac{P_l - P_{l-1}}{\Delta \xi} \right|$$

where

$$\Delta \xi = \frac{1}{(N-1)}$$

and $N$ is the number of points on the $\eta$ line.

With the allocation parameters $R1$ and $R2$, the grid points $x$ and $y$, and a dependent variable $p$, all evaluated at each of the grid nodes at the end of one solution step, the adaptation scheme is implemented by iterating for new arc length distributions. This process operates entirely on one grid line, say $\eta_j$, until converged, at which point $\eta_j$ is increased in a marching fashion from $\eta=1$ to $\eta=\eta_{\text{max}}$, illustrated in Fig.(2) for a typical problem. The algorithm describing this process may be described as follows:

1) Fix $\eta_j$

2) Assign guess values of $S$ along $\eta$ from $\xi = 0$ to $\xi = 1$, where $\Delta \xi = 1/(N-1)$ and $N$ is the number of points on $\eta$

3) For all $\xi$ determine the integrals

$$\int_{\xi_i}^{\xi_f} \frac{\partial p}{\partial \xi} d\xi$$

and
GRID ADAPTATION ALONG FIXED ETA LINES
GRID ADAPTATION ALONG FIXED ETA LINES

ETA = 15
ETA = 17
ETA = 19
ETA = 21

final adapted grid

FIG 2
\[ \int_{\xi_i}^{\xi_f} g(\frac{a^2P}{\partial \xi^2}) \, d\xi \]

by some quadrature rule.

4) Determine the constants \( \alpha \) and \( \beta \)

5) For all \( \xi \) determine the new arc lengths from equation (3).

6) For all \( \xi \) scale \( S \) by \( S_i = S_i / S_{\text{max}} \), and interpolate the new grid point locations \( X, Y \) and the new value of the dependent variable \( P \) at this point.

7) Test convergence in a least-squares fashion by

\[ \text{CON} = \sum_{\xi} (S_{\text{new}}(i) - S_{\text{old}}(i))^2 \]

If CON is close to zero, conditions are satisfied on \( \eta \) and the procedure returns to step (1), otherwise, exchange \( S_{\text{old}} \) with \( S_{\text{new}} \), and proceed with step (3).

The grid speed terms are then calculated by using a first order upwind difference approach between the old and new grid points. For time-asymptotic solutions or cases where the grid movement is very small, these grid speed terms may be set to zero. A more detailed discussion of the numerical procedure is presented in section III-2.
II-3. Orthogonalization Sweep

After the adaptation sweep is performed, some or all of the resulting grid cells may be skew, and further computations in the adapted domain may produce poor results and increased truncation error. Some degree of control over grid orthogonality must then be used. Several procedures for invoking this control are possible.

In most cases with grid adaptation, the orthogonality control is built into the adaptation scheme. Brackbill and Saltzman include an orthogonality measure integral, equation (1c), in the variational approach described earlier. Anderson and Rajendran [18] present a procedure whereby the adapted lines are determined by a technique similar to the equidistribution scheme described earlier for arc lengths in the S direction (Fig.(1)). However, an orthogonality control is included by adding an arc length measure in the N direction

\[ N = \int_0^\eta \{J(w/C_l)/\sin \theta\} d\eta \]

with

\[ C_l = \frac{S_{\text{max}}(\eta)}{\int_0^{S_{\text{max}}}(1/w) d\xi} \]

This equation controls the line intersection angles, \( \theta \), and is added to the equidistribution scheme. The resulting system is solved in hyperbolic fashion and requires a floating outer boundary. In the same paper, an orthogonality scheme based on a method for locating points
orthogonality scheme based on a method for locating points as a function of cell area variation is given. This technique provides grid speed equations which are then integrated to obtain the new grid points.

Control functions in differential equation form may generally be derived from automatic mesh generation routines, such as those developed by Thompson et al [19], Sorenson [20], or Steger and Sorenson [21]. A review of current procedures may be found in [22].

Yet another method of controlling orthogonality is the orthogonal trajectory scheme, generally performed exterior to the adaptation routine since it fits an orthogonal set of lines to an existing (adapted) set. Potter and Tuttle [23] describe a procedure for doubly-connected regions by which an \((N + M) \times (N + M)\) matrix equation is solved to determine orthogonal line segments between adjacent fixed lines. Some impressive results are given for highly skew initial meshes. However, even for an average size flow field grid, this method is computationally expensive.

A technique for generating nearly orthogonal trajectories is given by McNally [24]. In this method, a normal from a selected point on the fixed (adapted) grid lines is constructed and its intersection with the adjacent fixed line is determined. From this intersection point a normal
to the adjacent line is constructed and then translated to pass through the original point. A second point on the adjacent line is then determined from the original adjacent line intersection point defined above. The new "orthogonal" line segment between these two fixed lines is then defined by the original point and a point halfway between the two points on the adjacent line. The method used herein, called an approximate orthogonal trajectory (AOT) scheme, is similar to the one just described. However, it allows points on the adjacent lines freedom to move in either physical space direction.

The AOT scheme may be described as follows. The adapted lines are presumed to be the constant $\xi$ lines, and the $\eta$ lines, defined by the old points used in starting the AOT process, are to be replaced (i.e., made orthogonal to the $\xi$ lines). The original $\eta$ line segment endpoints are assigned to be $A$ and $A'$ in a typical H-shaped configuration (Fig.3(a)). The adapted lines are approximated by the neighboring points as lines 1 and 2 in Fig.(3(b)). A normal to line 1 is constructed through point $A$ and its intersection with line 2 is defined as point $B$. From point $B$, a normal to line 2 is constructed and its intersection with line 1 is defined as point $C$ (Fig.3(c)). Similarly, a normal to line 2 through point $A'$ intersects line 1 at point $B'$. 
TYPICAL INITIAL H GEOMETRY

LINE 1
SLOPE 1
TR1

LINE 2
SLOPE 2
TR2

[a] TYPICAL INITIAL H GEOMETRY

LINE APPROXIMATIONS FOR SLOPES AND TRAJECTORIES

[c] NORMAL TRAJECTORIES FROM POINT A TO B TO C

[d] NORMAL TRAJECTORIES FROM POINT A' TO B' TO C'

[e] NEW POINTS SWMF = 1

[f] NEW POINTS SWMF = 0

FIG (3)
APPROXIMATE ORTHOGONAL TRAJECTORY PROCESS FOR TYPICAL GEOMETRY
Point C' is then defined as the intersection of the normal to line 1 from point B' with line 2 (Fig. 3(d)). The new endpoints E and E' for the (orthogonal) line segment of η between lines 1 (ξ_{l-1}) and 2 (ξ_{l}) are determined as the points half the distance between points B' and C on line 1 and points B and C' on line 2 (Fig. 3(e)). The resulting lines in general will not be orthogonal, and in fact their slopes across a line may be discontinuous, but their deviation from orthogonality will be smaller than with the original η lines.

In this scheme, all interior grid points may be relocated twice as the process is marched from one boundary to the other. Several consecutive AOT sweeps will converge the grid to a nearly orthogonal state, although only one sweep is generally necessary.

By approximating the ξ lines with the two neighboring points as shown (Fig. 3(b)), the final intersection points E and E' may not lie on the actual ξ lines. This creates good orthogonality, but may relax the adaptation process excessively. Therefore, a parameter SWMF, which ranges from 0 to 1, is included in the algorithm to allow control of the amount of relaxation to be tolerated. With full relaxation (SWMF=1), both x and y are redefined as in Fig(3.(e)). With no relaxation allowed (SWMF=0), the x coordinates are not redefined (Fig.(3(f))).
II-4. Interpolation Onto New Grid

In general, the dependent variables in the problem must be interpolated onto the new grid before the solution can continue. The method of interpolation is dependent upon the solution technique, and in certain codes and problems may not be necessary. In some cases the adaptation process may not cause a drastic movement of points within the grid, and the solution technique may absorb the value differences of the variables between old and new nodes without interpolation.

In many cases, local linear interpolation is sufficient. It is for these instances that two routines have been included in the grid adaptation package presented herein. These perform the necessary variable updating process. The first routine concerns that of locating which old grid cell, where the variable values are currently known, contains the new point. With the new point so located, interpolation may be performed within the cell to define data at the new point. Janssen [25] describes a technique by which a point may be described as inside, on, or outside a given triangle. This technique uses the area coordinates transformation for the triangle region and test point. With 1, 2, and 3 describing the nodes of the triangle cell, this transformation is
or inversely,

\[
\begin{bmatrix}
\epsilon_1 \\
\epsilon_2 \\
\epsilon_3
\end{bmatrix} = \frac{1}{2A} \begin{bmatrix}
x_2 y_3 - x_3 y_2 & y_2 x_3 - y_3 x_2 \\
x_3 y_1 - x_1 y_3 & y_3 x_1 - y_1 x_3 \\
x_1 y_2 - x_2 y_1 & y_1 x_2 - y_2 x_1
\end{bmatrix} \begin{bmatrix}1 \\ x \\ y \end{bmatrix}
\]

where the determinate of the transformation is

\[
2A = x_{13} y_{23} - y_{13} x_{23}
\]

and the matrix elements are defined as

\[
x_{\alpha \beta} = x_\alpha - x_\beta, \text{ etc.}
\]

By definition, \( \epsilon_1 = A_1/A, \epsilon_2 = A_2/A, \epsilon_3 = A_3/A, \) and \( \epsilon_1 + \epsilon_2 + \epsilon_3 = 1. \) (Fig. 4, with \( E \) representing \( \epsilon_1 \)). If all three of the area coordinates \( \epsilon \) are in the range zero to one, the point of interest is enclosed by the triangle. If one \( \epsilon \) coordinate is zero, the point lies on the corresponding side. If the minimum \( \epsilon \) coordinate is zero and the maximum \( 1, \) the point lies on a triangle node. In all other circumstances, the point lies outside the triangle.

The cell search algorithm utilizes this transformation by dividing the current test cell into two triangles and testing the location of the new point. If an area coordinate is negative, the triangle is "flipped" over the corresponding side, and retested until the triangle contains the point.
FIG (4)

AREA COORDINATE INTERPOLATION

POINT (X,Y) CONTAINED IN SHADED REGION OF QUADRILATERAL (A) IS INTERPOLATED BY AREA COORDINATES DEFINED IN (B)
The convenience of this method lies in the fact that for isoparametric elements such as these triangle, the area coordinates are also the shape, or interpolation, functions for the triangle, as noted by Zienkiewicz [26]. Thus, when the triangle containing the point is located, the interpolation functions needed for updating the variable values at this node are known, and are stored for convenient use later.

The second routine uses the interpolation functions and triangle indices for the \((i,j)\) node obtained in the first routine, along with variable data from the old grid to interpolate these variables onto the new grid. This is accomplished through the relation

\[
\phi' = \sum_{i=1}^{3} H_i \phi_i
\]

where \(\phi'\) is the new variable value, \(H_i\) are the interpolation functions (the area coordinates \(\varepsilon_i\)) and \(\phi_i\) are the old grid variable values at the local triangle nodes \(i\). Since the interpolation functions \(H_i\) are the same for all solution variables, this routine is repeated for each variable to be updated.
III. NUMERICAL SOLUTION

Six subroutines make up the adaptive grid module. Their descriptions are presented in some detail in this section. Together they perform the adaptation, AOT and interpolation operations. These subroutines are

AGDIR Direction control
AGRID Adaptation
AGFDE Difference expressions
AGORTH AOT
AGINTC Interpolation cells
AGINTF Variable updating

Their interaction or independence is illustrated in Fig.(5). The prefix AG (Adaptive Grid) is used to distinguish these subroutines from others that may exist in the main program.

Use of the AG subroutines assumes that an initial grid has already been generated and that the dependent variable to be used for adaptation has also been defined at each point in the adaptation domain. Ordinarily this is accomplished automatically by the main program solution procedure, but it may be done manually at any step in the solution procedure.
FIG (5)

INTERACTIVE:

AGRID

AGDIR

AGFDE

AGINTC

INDEPENDENT:

AGORTH

AGDIR

AGINTC

AGINTF

SUBROUTINE INTERACTION AND INDEPENDENCE
Implementation of the AG subroutines requires few modifications to the main program and the individual subroutines. A named COMMON block, ADAPT, which includes the arrays XOLD, YOLD, CINT and FINT, must be added to the main program. With dimensioning performed in the COMMON statement, the format will be

\[
\text{COMMON/ADAPT/ XOLD(II,JJ),YOLD(II,JJ), CINT(II,JJ,4),FINT(II,JJ,3)}
\]

where XOLD, YOLD, CINT, and FINT are defined within the AG routines described below.

The array dimensions II and JJ are defined in PARAMETER statements in each of the AG subroutines to allow for variable dimension arrays without adding the dimensions to the argument lists. The PARAMETER MAX is to be set to the greater of II or JJ. Thus the PARAMETER statement in each of the subroutines must be checked for proper values of II, JJ, and MAX before execution. Other variables are included in the PARAMETER list for some of the subroutines, and will be discussed later. These are assigned default values but may be changed for specific problems.

Adaptation and orthogonalization directions are determined from the argument IDIR, for which the valid values are 0 for the $\xi$ direction and 1 for the $\eta$ direction. Other required arguments are also discussed later with the individual routines.
The bulk of the storage space required by the AG routines is needed for the arrays SN(MAX), S(MAX), DPA(II,JJ,2), DPB(II,JJ,2), XOLD(II,JJ), YOLD(II,JJ), CINT(II,JJ,4), FINT(II,JJ,3), V(II,JJ) and VOLD(II,JJ). Some of the space allocated by these arrays will not be used when the adaptation domain is a subregion of the computational grid space. Using the minimum array size was not deemed appropriate since the subroutine simplicity and comprehensibility would diminish due to the extra coding needed to access these arrays. When interpolation is not used, the arrays CINT, FINT, and VOLD are not required, and the first two may be removed from the COMMON block if the additional subroutine modifications are made. The maximum array storage space required is then 15*II*JJ + 2*MAX elements. Without interpolation, this may be reduced to 7*II*JJ + 2*MAX elements. This may be further reduced to 3*II*JJ + 2*MAX by recalculating the derivatives (sacrificing efficiency) instead of storing them.

It is presumed above that contiguous storage space for the 3-dimensional arrays is not a problem. These arrays were designed for facility and clarification of the coding. However, if space for these arrays is not available in contiguous blocks, the arrays may be broken into several 2-dimensional arrays. For example, FINT(II,JJ,3) may be redefined by FINT1(II,JJ), FINT2(II,JJ), and FINT3(II,JJ).
Modifications are then required in the COMMON/ADAPT/ block and in the subroutine coding where FINT is used.

### III-1. Direction Control

Direction control for subroutines AGRID and AGFDE is provided by subroutine AGDIR.

The variable IDIR, passed as an argument throughout the subroutines, must be set to 0 for adaptation in the $\xi$ direction or to 1 for adaptation in the $\eta$ direction. Dummy loop control indices $M$ and $N$ are passed to AGDIR, where proper array indices are determined based on direction and adaptation boundaries $MMIN$ and $MMAX$.

For $\xi$ direction adaptation, IDIR=0 and the array indices returned are

\[
\begin{align*}
  I &= M \\
  J &= N \\
  IM &= M - 1 \\
  JM &= N \\
  IP &= M + 1 \\
  JP &= N.
\end{align*}
\]

If the point $M$ is on the adaptation boundary, the points $IM$ or $IP$ are set to $M$. For $\eta$ direction adaptation, IDIR=1 and the returned indices are

\[
\begin{align*}
  I &= N \\
  J &= M
\end{align*}
\]
\[ IM = N \]
\[ JM = M - 1 \]
\[ IP = N \]
\[ JP = M + 1 \]

and boundary points again assign a value of \( M \) to \( JM \) or \( JP \). Note that the forward indices \( IP \) and \( JP \), or the backward indices \( IM \) and \( JM \), default to \( I \) or \( J \) if adaptation is not performed in the respective direction.

III-2. Grid Adaptation Sweep

The adaptation process is contained in subroutine AGRID. The algorithm actually coded is similar to that described in section II-2., but contains several added features. The data required to implement AGRID consists of the COMMON/ADAPT/ block, an argument list and PARAMETER specifications. The argument list consists of the allocation constants \( R1 \) and \( R2 \), the adaptation boundary limits \( MMIN \) and \( MMAX \), the direction control variable \( IDIR \), a smoothing flag \( ISMT \), the dependent variable for adaptation \( V \), and the grid point arrays \( X \) and \( Y \). PARAMETER specifications include \( II \), \( JJ \) and \( MAX \), as well as a maximum iteration limit \( ITMAX \). The convergence tolerance \( EPS \) is assigned through a DATA statement. This routine will operate in either the \( \xi \) or \( \eta \) direction by specifying \( IDIR=0 \) or \( IDIR=1 \) respectively in the argument list.
An adaptation sub-domain, used by Steinbrenner, Anderson and Tassa [27], may be defined as

\[
D(\text{adaptive}) = \begin{cases} 
  x = x(\xi, \eta) & \xi_1 \leq \xi \leq \xi_2 \\
  y = y(\xi, \eta) & 1 \leq \eta \leq n_{\text{max}}
\end{cases}
\]

or

\[
D(\text{adaptive}) = \begin{cases} 
  x = x(\xi, \eta) & 1 \leq \xi \leq \xi_{\text{max}} \\
  y = y(\xi, \eta) & \eta_1 \leq \eta \leq n_2
\end{cases}
\]

depending on IDIR. By specifying dummy indices \(N\) for the outer (fixed \(\eta\) or \(\xi\)) loop and \(M\) for the inner (variable \(\xi\) or \(\eta\)) loops, the adaptive domain boundaries \(MMIN\) and \(MMAX\) are the range limits for \(M\). The adaptation domain may consist of the entire \((\xi, \eta)\) domain if desired. Since the AGRID routine does not move the boundaries, the adaptation domain boundary lines will remain defined within the full domain after adaptation.

For convenience, two arrays \(XOLD\) and \(YOLD\) are filled with the current \(X\) and \(Y\) values, respectively, upon entry to the subroutine. These arrays are used in the future interpolations and in routine AGINTC. The dependent variable array \(V\) is also put into an array \(P\), so that the smoothing and interpolation procedures within the adaptation scheme do not modify the original data. This assignment process may be moved to the main program if the AGRID routine is used more than once during a particular grid updating sequence.
The outer loop controls the constant computational grid line being adapted, and ranges from 1 to NN, where NN is determined by the direction of adaptation. Similarly, the maximum range of the inner loops is set to MM. These limits are set to NN=JJ and MM=II for IDIR=0, or to NN=II and MM=JJ for IDIR=1. The limit MM is used to determine the grid interval spacing DM by

$$DM = \frac{1.0}{(MM - 1)}.$$ 

In order to fix the domain boundaries, the arc length vector SN elements MMIN and MMAX are normalized to zero and one respectively. Since the numerical integration is performed in an upwind fashion, the element SN(MMIN) is set to zero as a default and the counter range for M is MMIN + 1 to MMAX.

Most routines smooth the local dependent variable P before the adaptation process begins in order to gain better grid control in rough data or high gradient regions. This is optional in the AGRID routine. The flag argument ISMT is used to turn the smoothing algorithm on or off. If ISMT is other than zero, a filtering scheme is applied to P over all N and M, by

$$P(I,J) = (P(IM,JM) + 2*P(I,J) + P(IP,JP)) \times 0.25,$$

where I, J, IM, JM, IP and JP are a function of N and M and are determined by subroutine AGDIR.

The outer loop over N contains five individual loops over M. The first of these five loops is necessary to assign
guess values to the new arc length vector $SN$ for the iteration process and to set the old arc length vector $S$ to zero. By calling AGDIR to obtain values of $I$, $J$, $IM$ and $JM$, these guesses are arc secant approximations given by

$$SN(M) = SN(M-1) + ((X(I,J) - X(IM,JM))^2 + (Y(I,J) - Y(IM,JM))^2)^{0.5}.$$ 

The second loop scales the arc length vector $SN$ so that $SN(MMAX) = 1.0$, performs the summation necessary to test convergence of the solution, and replaces the $S$ vector elements with the $SN$ elements. This involves solving the equations

$$\frac{SN(M)}{SN(MMAX)}$$

$$CON = CON + (SN(M) - S(M))^2$$

and

$$S(M) = SN(M)$$

for all $M$, with initialization of $CON$ to 0 before the loop is encountered. Convergence is then tested for $CON$ less than or equal to the error tolerance EPS. When convergence is satisfied, control is passed to the outer loop. If convergence is not satisfied, which it will not be during the zeroth iteration, the process continues with the next inner loop.

The third loop uses a trapezoidal rule [28] to estimate the integrals

$$\int_{MMIN}^{MMAX} |\partial P/\partial M|dM$$
Current values for these integrals are named TF1M and TF2M in the code. Subroutine AGFDE, described below, is used twice, once for M-1 and once for M, to estimate the local derivatives

\[ |3P/3M| \]

and

\[ |g(3^2P/\partial M^2)| \]

which in this instance are simply the integrands. The upwind values of the first and second derivatives are stored in the matrix elements DPA(M,1) and DPA(M,2), respectively. The current (forward) values of these derivatives are similarly stored in the matrix elements DPB(M,1) and DPB(M,2). Two trapezoidal rules are then implemented by

\[ TF1M = TFIM + 0.5 \times DM \times (DPA(M,1) + DPB(M,1)) \]

for the first integral, and

\[ TF2M = TF2M + 0.5 \times DM \times (DPA(M,2) + DPB(M,2)) \]

for the second. These estimates of the integrals are used to calculate the weight function parameters \( \alpha \) and \( \beta \) by

\[ AL = R1 / (1.0 - R1) / TF1M \]

and

\[ BE = R2 / (1.0 - R2) / TF2M * (1.0 + AL * TF1M). \]

The fourth loop determines the new arc length values \( SN \) based on a trapezoidal rule and the derivatives calculated
in the previous loop. The arc lengths are calculated by
\[
DSN = 0.5 \times ((1 + AL \times DPA(M,1) + BE \times DPA(M,2)) + \\
(1 + AL \times DPB(M,1) + BE \times DPB(M,2)))
\]
and
\[
SN(M) = SN(M-1) + DM/DSN.
\]

The fifth and final loop is used to normalize the SN vector by
\[
SN(M) = SN(M) / SN(MMAX)
\]
and to perform interpolation for the new X and Y grid point values, and the P values corresponding to these points as a function of SN. Each arc length end point SN(M) on a fixed line is first bracketed in the computational space (Fig.6, with S1 meaning SN(1), etc.). The scanning index IB defines the forward and backward computational space arc length brackets by
\[
DMP = (IB - MMIN) / (MMAX - MMIN)
\]
and
\[
DMM = (IB - MMIN - 1) / (MMAX - MMIN).
\]
If SN(M) is between these points, IB is the forward bracket index. If this index fails to bracket SN(M), another scanning iteration takes place, redefining IB by
\[
IB = IB + ISIGN(1,INT(SN(M) - DMM)).
\]
With IB found, piecewise linear interpolation from "old" space values to "new" space values is performed using a similar triangle process.
ARC LENGTH INTERPOLATION IN COMPUTATIONAL SPACE

PHYSICAL SPACE ARC LENGTHS $s$ ARE BRACKETED BY NODES IN COMPUTATIONAL SPACE
Subroutine AGDIR is again invoked with the indices IB and N to determine endpoint indices IS, JS, and ISM, JSM, where ISM=IS-1 and JSM=JS-1. The X and Y values corresponding to these indices define the hypotenuse of a right triangle in physical space. The length of this hypotenuse may be regarded as \((DMP-DMM)\), or a computational space interval. The arc length \(SN(M)\) endpoint lies on this line segment, and the ratio \(DS=(SN(M)-DMM)/(DMP-DMM)\) drives the interpolation process. The projection of the hypotenuse onto the X- or Y-axis defines a leg length of this triangle:

\[
A = XOLD(IS,JS) - XOLD(ISM,JSM),
\]
\[
B = YOLD(IS,JS) - YOLD(ISM,JSM).
\]

The variation in P on the hypotenuse is

\[
C = P(IS,JS) - P(ISM,JSM).
\]

The new X, Y, and P values are then evaluated by applying the ratio DS to these line segments in piecewise linear fashion, i.e.

\[
X(I,J) = XOLD(ISM,JSM) + A * DS,
\]
\[
Y(I,J) = YOLD(ISM,JSM) + B * DS,
\]

and

\[
P(I,J) = P(ISM,JSM) + C * DS.
\]

This step completes one adaptation iteration. The iteration counter IT is then tested against ITMAX. If IT is smaller than ITMAX, another iteration may be performed by returning to the end of the first inner loop. When IT
exceeds ITMAX, control is passed to the outer loop. This condition is normally flagged to the alert the user, however unless the grid is extremely skew or sparse, adjacent lines will also cover the region where the adaptation process has not converged, and adaptation on these lines will provide similar point movement, thus damping the adapted grid error.

It may be noted that the first loop encountered after the return point for the next iteration includes another scaling of the arc length vector. This is not necessary since this vector is already properly scaled after an iteration, but is performed in order to reduce the number of programming loops required for the routine.

Although the adaptation process spaces grid points more closely in the neighborhood of large gradients or solution extrema in order to improve flow solution accuracy, high order formal accuracy on the part of the adaptation process itself is not required. Consequently, the convergence tolerance, EPS, may be rather large compared to zero (0.005 is used in the examples), second order finite difference expressions are very reasonable, and double-precision arithmetic is certainly not warranted.

Subroutine AGFDE, mentioned above, is used in AGRID to calculate the required property derivatives by finite difference expressions. Given the domain boundaries MMIN
and MMAX, the direction indicator IDIR, the adaptation dependent variable P, and a local node I, J, the first and second derivatives $\partial P/\partial \xi$, $\partial^2 P/\partial \xi^2$ or $\partial P/\partial \eta$, $\partial^2 P/\partial \eta^2$ are returned. The derivative directions are thus dependent on the direction of adaptation, i.e. with respect to $\xi$ for IDIR=0, or with respect to $\eta$ for IDIR=1. The second derivative control function $G$ is also calculated, and is applied to the second derivative before it is returned. The values returned from the subroutine are the absolute values of the first and second derivatives D1 and D2, respectively.

These derivatives are approximated by second order finite difference expressions. Central differences are used for the interior nodes, and forward or backward differences are used at the boundaries. Since the derivatives are calculated in computational space, where the grid spacing is uniform, an interval length $DX = 1.0 / (II - 1)$ or $DX = 1.0 / (JJ - 1)$ may be applied to simplify the expressions. Subroutine AGDIR is used to determine IP=I+1 and IM=I-1, or JP=J+1 and JM=J-1, again dependent on IDIR. The indices IP2=I+2, JP2=J+2, IM2=I-2, and JM2=J-2, are determined by

$$IP2 = I - 2 \times IDIR + 2,$$

or
IM2 = I + 2 * IDIR - 2,
JP2 = J + 2 * IDIR,

and

JM2 = J - 2 * IDIR.

The default values for JP, JM, JP2, and JM2 are J for IDIR=0, and similarly IP = IM = IP2 = IM2 = I for IDIR=1. In this way only one set of difference expressions need be coded, allowing for simple modification or extension to higher order formal accuracy if desired.

The terms DPM and DPP, needed to calculate the control function G, are calculated after each pair of derivative calculations, as these are also boundary dependent. For nodes on the MMIN boundary, the expressions calculated, using forward differences for D1 and D2, are

\[
D1 = (-P(IP2,JP2) + P(IP1,JP1)*4 - P(I,J)*3) * 0.5/DX,
\]
\[
D2 = ((P(IP2,JP2) - P(IP1,JP1))/DX -
(\frac{P(IP1,JP1) - P(I,J)}{DX})/DX),
\]
\[
DPP = (P(IP1,JP1) - P(I,J))/DX,
\]
\[
DPM = 0.
\]

On the MMAX boundary, backward differences are used, and the expressions are

\[
D1 = (P(I,J)*3 + P(IM1,JM1)*4 + P(IM2,JM2)) * 0.5/DX,
\]
\[
D2 = ((P(I,J) - P(IM1,JM1))/DX -
(\frac{P(IM1,JM1) - P(IM2,JM2)}{DX})/DX),
\]
\[
DPP = 0,
\]
\[ DPM = (P(I,J) - P(IM1,JM1)) \times 0.5/DX. \]

For interior node points, central differences are used, resulting in

\[ D1 = (P(IP1,JP1) - P(IM1,JM1)) \times 0.5/DX, \]
\[ D2 = \frac{((P(IP1,JP1) - P(I,J))/DX - (P(I,J) - P(IM1,JM1))/DX)/DX}{DX}, \]
\[ DPP = (P(IP1,JP1) - P(I,J))/DX, \]
\[ DPM = (P(I,J) - P(IM1,JM1))/DX. \]

In each case, program control passes to the bottom section, where \( G \) is calculated as

\[ G = \text{ABS}(DPP - DPM) \]

and the absolute values of \( D1 \) and \( G \times D2 \) are calculated prior to returning control to subroutine AGRID.

III-3. Orthogonalization Sweep

The approximate orthogonal trajectory subroutine AGORTH requires boundary specification arguments IMIN, IMAX, JMIN and JMAX, the direction control variable IDIR, PARAMETER specification of II and JJ, and the grid point arrays X and Y. Two additional variables unique to this subroutine are also required. The argument SWMF is the control factor for the allowable redefinition of adapted lines, and is valid in the range zero to one. The PARAMETER SLOPE is the tolerance of fixed adapted line slopes to zero. If one or both of the current pair of fixed lines has a local slope less than or
equal to this value, the slope of the normal to these points is assumed to be infinite, and default point locations are assumed.

The subroutine consists of two halves, one for $\xi$ direction orthogonalization and one for $\eta$ direction orthogonalization. The procedure is identical for both, however the coding differs by exchanging $X$ for $Y$ and $I$ for $J$. Thus for a given value of IDIR only half of the subroutine is actually executed.

The procedure for IDIR=1 (opposite direction in Fig(3)) may be described as follows. In this case, the lines of constant $\xi$ are to be replaced by the AOT scheme. Two nested programming loops are utilized, the outer loop over $J$ to control the current pair of adapted $\eta$ lines, and the inner loop over $I$ controlling the individual $\xi$ line segment being constructed. The segment construction takes place in an upwind fashion, therefore the limits of $J$ are $J\text{MIN}+1$ and $J\text{MAX}$. The solution is thus marched from the lower limit to the upper limit (in this case from the bottom to the top of the grid). The limits on $I$ are simply the given boundary lines IMIN and IMAX. For convenience and to limit arithmetic operations, the values

\[ J\text{M1} = J - 1 \]
\[ I\text{P1} = I + 1 \]
\[ I\text{M1} = I - 1 \]
are defined for each I and J. If the value I is on the IMIN or IMAX boundary the value IM1 or IP1 is reassigned to I. In order to allow boundary points to be relocated only along the boundary, thus keeping the boundary shape fixed, the flag IBDY is set to one if I=MIN or I=MAX, and is zero otherwise. The endpoints of the line segment to be constructed then have the indices (I,JM1) and (I,J). The local slopes of these lines, SL1 and SL2 (Fig.3(b)), are approximated by

\[
SL1 = \frac{Y(IP1,JM1) - Y(IM1,JM1)}{X(IP1,JM1) - X(IM1,JM1)}
\]

and

\[
SL2 = \frac{Y(IP1, J ) - Y(IM1, J )}{X(IP1, J ) - X(IM1, J )}.
\]

If these values are sufficiently close to zero as measured by the tolerance SLOPE, the respective slope of the normal trajectory from either point, TR1 or TR2, is set to zero as a default (meaning TR1 or TR2 actually approaches infinity). If SL1 or SL2 is not zero, the respective trajectory slopes TR1 or TR2 are calculated from the inverse reciprocal relationships

\[
TR1 = -1.0 / SL1
\]

or

\[
TR2 = -1.0 / SL2.
\]

Thus, the assignment of TR1=0 or TR2=0 as a default requires that SL1 or SL2 not be so large as to set TR1 or TR2 to single precision zero through these equations.
With these slopes and trajectories, the six points A, B, C, A', B' and C' are now determined, letting AP, BP and CP represent A', B' and C' in the coding.

The points A and AP are simply the original node points (Fig. 3(a))

\[
\begin{align*}
X_A &= X(I, J M_1), \\
Y_A &= Y(I, J M_1), \\
X_{AP} &= X(I, J), \\
\end{align*}
\]

and

\[
\begin{align*}
Y_{AP} &= Y(I, J).
\end{align*}
\]

Point B on line J is determined by the intersection point of the J line and the line constructed from point A normal to the J-1 line (Fig. 3(c)). The equation for a point XB, YB on the normal to the J-1 line is

\[
Y_B - Y_A = TR_1(X_B - X_A),
\]

and the equation for a point XB, YB on the J line is

\[
Y_B - Y_{AP} = SL_2(X_B - X_{AP}).
\]

The equations are solved simultaneously for XB and YB by letting

\[
\begin{align*}
A &= TR_1, \\
B &= SL_2, \\
C &= -TR_1 * X(I, J M_1) + Y(I, J M_1), \\
D &= -SL_2 * X(I, J) + Y(I, J).
\end{align*}
\]

Then
\[
XB = \frac{(D - C)}{(A - B)}
\]

and
\[
YB = \frac{(TR1 \times D - C \times SL2)}{(A - B)}.
\]

If the J line is the JMAX boundary, a YB default is used, defined by
\[
YB = SL2 \times (XA - XAP) + YAP.
\]
Similarly, if I is on the IMIN or IMAX boundary, a default XB is used, given as
\[
XB = XA.
\]
(In this instance, the trajectory slope is parallel to the direction of AOT application). If TR1 = 0, both of these defaults are used.

The points C, BP and CP are determined in a similar fashion. Point C is determined by the intersection of the J-1 line with the line defined by TR2 and point B (Fig.3(c)). This simultaneous system is solved for XC and YC by
\[
A = TR2,
\]
\[
B = SL1,
\]
\[
C = -TR2 \times XB + YB,
\]
\[
D = -SL1 \times X(I,J) + Y(I,J),
\]
and
\[
YC = \frac{(TR2 \times D - C \times SL1)}{(TR2 - SL1)},
\]
\[
XC = \frac{(D - C)}{(TR2 - SL1)}.
\]
The JMIN boundary default value is
YC = SL1 * (XB - XA) + YA

and the IMIN or IMAX boundary default is

XC = XB.

If TR2 = 0 both of these defaults are used.

Point BP is determined by the intersection of the J-l line with the normal constructed from AP (Fig. 3(d)). This system is solved through

A = TR2,
B = SL1,
C = -TR2 * X(I, J) + Y(I, J),
D = -SL1 * X(I, JM1) + Y(I, JM1),

and

YBP = (TR2 * D - C * SL1) / (A - B),
XBP = (D - C) / (A - B).

The default values are

YBP = SL1 * (XAP - XA) + YA
when JM1=JMIN, and

XBP = XAP
when I=IMIN or I=IMAX. Both defaults are used if TR2 = 0.

The last point, CP, is determined by the system solution of the intersection of the J line with the line defined by point BP and TR1 (Fig. 3(d)), given by

A = TR1,
B = SL2,
C = -TR1 * XBP + YBP,
\[ D = -SL2 \times X(I,J) + Y(I,J), \]
\[ YCP = (TR1 \times D - C \times SL2) / (A - B), \]
and
\[ XCP = (D - C) / (A - B). \]
The defaults are given by
\[ YCP = SL2 \times (XBP - XAP) + YAP \]
for \( J = JMAX \), and by
\[ XCP = XBP \]
on the IMIN and IMAX boundaries. Again, both defaults are used if \( TR1 = 0 \).

Finally, the new grid points (E and E' in Fig.(3(e))) are determined. Since the constant \( \eta \) lines may have been determined by adaptation, the Y grid points may remain fixed or be allowed some degree of relaxation. The argument variable SWMF controls this movement. Any value between zero and one then allows some fraction of movement, with the value one allowing full movement as defined by the AOT process (Fig.3(e)). If SWMF is given the value zero, no redefinition of the Y values is allowed (Fig.3(f)).

The new X and Y points are then defined by
\[ X(I,JM1) = (XBP + XC) \times 0.5, \]
\[ Y(I,JM1) = Y(I,JM1) + SWMF*(0.5 \times (YBP + YC) - Y(I,JM1)), \]
\[ X(I, J) = (XCP + XB) \times 0.5, \]
and
\[ Y(I, J) = Y(I, J) + SWMF*(0.5 \times (YCP + YB) - Y(I, J)). \]
Program control is then passed to the I loop and the process is repeated. The new I, J values are used as the I, JM1 values when the J loop counter is increased.

III-4. Interpolation Onto New Grid

Subroutine AGINTC is used to determine the interpolation functions necessary to update the flow variables onto the new grid. In general this subroutine will be called from the main program, and its complement, subroutine AGINTF, will be utilized immediately afterward by a call for each variable to be updated.

The operation of AGINTC requires the COMMON/ADAPT/ block, as this contains the old grid point arrays, the index limits IMIN, IMAX, JMIN and JMAX, and grid point arrays X and Y as arguments. Upon completion, the arrays CINT and FINT are returned via the COMMON/ADAPT/ block, and will include the triangular region indices and interpolation functions for each node, respectively, for use by AGINTF.

This routine is applied over the range given by the index limit arguments. This may be only the adaptation or orthogonalization domain, or the entire grid domain (1,II,1,JJ) for final variable interpolation. For each I,J pair defining a new grid node, the indices I1, I2, I3, J1, J2 and J3 define three nodes for a triangle. These triangles operate in the (XOLD,YOLD)-space by varying the
triangle node locations until the triangle contains the $X(I,J), Y(I,J)$ point.

The I index is used as the outer loop, and J the inner, although the sequence may be reversed. As each new I value is encountered, default node values must be specified. Before the J loop is encountered, these defaults are set by $I_1 = I$, $J_1 = J_{MIN}$, $J_3 = J_{MIN} + 1$, and $I_2 = I + 1$. If $I = I_1$, then $I_2 = I_1 - 1$, and if $J = J_J$, then $J_3 = J_J - 1$, in order to define triangles within the physical domain and avoid array accessing errors. By the node numbering scheme shown (Fig. 7), the four indices $I_1, J_1, I_2, J_3$ uniquely define the triangle, since $I_3 = I_1$ and $J_2 = J_1$ for any possible orientation. Thus $I_3$ and $J_2$ are defined within the iteration enclosed by the J loop.

With the node indices now defined, the test procedure begins. The coordinates of the triangle nodes are given by

$$
X_1 = X_{OLD}(I_1, J_1),
Y_1 = Y_{OLD}(I_1, J_1),
X_2 = X_{OLD}(I_2, J_2),
Y_2 = Y_{OLD}(I_2, J_2),
X_3 = X_{OLD}(I_3, J_3),
Y_3 = Y_{OLD}(I_3, J_3).
$$

The area coordinate transformation is then applied by

$$
E_1 = TA \ast ((X_2 \ast Y_3 - X_3 \ast Y_2) + (Y_2 - Y_3) \ast X(I,J) + (X_3 - X_2) \ast Y(I,J))
$$
FIG (7)
POSSIBLE ORIENTATIONS AND NODE NUMBERS
OF TRIANGLE CELLS FOR A TYPICAL NODE (1, J)
\[ E2 = TA \times ((X3 \times Y1 - X1 \times Y3) + (Y3 - Y1) \times X(I, J) + (X1 - X3) \times Y(I, J)) \]
\[ E3 = TA \times ((X1 \times Y2 - X2 \times Y1) + (Y1 - Y2) \times X(I, J) + (X2 - X1) \times Y(I, J)) \]

where \( E1 = \varepsilon_1 \), \( E2 = \varepsilon_2 \), \( E3 = \varepsilon_3 \), and \( TA \) is defined by
\[ TA = \frac{1}{(X1 - X3) \times (Y2 - Y3) - (Y1 - Y3) \times (X2 - X3)} \]

The minimum value of these coordinates are then determined by
\[ EMIN = AMIN1(E1, E2, E3) \]

Since the concern here is whether the point \( X(I, J) \), \( Y(I, J) \) is on or within the triangle, the test is assumed positive if \( EMIN \) is greater than or equal to zero. If this is true the other points will be less than or equal to one by definition, and control is passed to the storage section of the routine. When this is not the case, at least one of the area coordinates must be negative. When an area coordinate is negative, the routine "flips" the triangle over the corresponding side by redefining the indices, and invokes another test iteration.

If \( E1 \) is negative, the test point lies outside the triangle across the side 1 boundary (Fig. 4). Thus the triangle is flipped over this side by redefining the nodes as
\[ I1 = I2, \]
J1 = J3.

In order for this to be a valid orientation as defined above, the nodes I2 and J3 must be reassigned by

I2 = I3,
J3 = J2.

If E2 is negative, the triangle region is flipped over side 2 by assigning

I2 = I2 - ISIGN(2, I2 - I1).

Similarly, if E3 is negative, the triangle region is flipped over side 3 by assigning

J3 = J3 - ISIGN(2, J3 - J1).

After the nodes have been reassigned, another test process takes place by returning to the definition of node coordinates within the J loop.

When the proper triangle region has been located, the four definitive indices are stored in the array CINT by

CINT(I,J,1) = I1,
CINT(I,J,2) = J1,
CINT(I,J,3) = I2,
CINT(I,J,4) = J3,

and the area coordinates are stored as the interpolation functions in the array FINT by

FINT(I,J,1) = E1,
FINT(I,J,2) = E2,
\[ \text{FINT}(I,J,3) = E3. \]

The node indices \( I_1, I_2, I_3, J_1, J_2 \) and \( J_3 \) from the triangle found for the \( I,J \) location are used as the starting point for the \( J+1 \) iteration. This should eliminate repetition of the same sequence of triangle flips to locate adjacent points. Also, the new grid boundaries are identical to those of the old grid, so the procedure will not cause triangles to be oriented out of the domain, and no other tests for boundary points are required.

Upon completion of this procedure for all \( I \) and \( J \), control is returned to the main program where subroutine AGINTF then uses the arrays filled by the current subroutine to update the variables onto the new grid. Subroutine AGINTF uses an array variable name, designated by dummy variable \( V \) in the subroutine, and the arrays CINT and FINT, filled by AGINTC, to update this array variable onto the new grid. An array \( VOLD \) is used within the subroutine to store the incoming values of \( V \) so that interpolation between neighboring points is performed entirely with the old values of \( V \). This array is loaded by simple nested loops over \( I \) and \( J \).

The interpolation now takes place by implementing another nested loop over \( I \) and \( J \) and retrieving data from the CINT and FINT arrays. The interpolation function weights \( F1,F2, \)
and $F_3$ are assigned values stored in array $FINT$ by

$$F_1 = FINT(I,J,1),$$
$$F_2 = FINT(I,J,2),$$
and

$$F_3 = FINT(I,J,3).$$

The interpolation triangle indices are similarly recovered from values stored in the array $CINT$ by the statements

$$I_1 = CINT(I,J,1),$$
$$J_2 = CINT(I,J,2),$$
$$I_2 = CINT(I,J,3),$$
and

$$J_3 = CINT(I,J,4).$$

The indices $I_3$ and $J_2$ are by default $I_1$ and $J_1$ respectively.

The new interpolated $V$ value is then calculated by

$$V(I,J) = VOLD(I_1,J_1)F_1 + VOLD(I_2,J_2)F_2 + VOLD(I_3,J_3)F_3.$$ 

When this has been performed for all $I$ and $J$, control is returned to the main program. This subroutine is called for each variable to be updated onto the new grid. When all necessary variables have been updated, further solution of the problem may continue.
IV. EXAMPLES

IV-1. Test Cases

Two exemplary variable fields on 21 x 21 square grids were tested. The functional distributions for these cases were taken from Anderson [2]. The initial grid for both cases is generated with x and y values ranging from 0 to 20, and is shown in Fig.(8).

The first test case was a simple distribution of the field variable p by
\[
p(x,y) = 1.0 \text{ for } 0 \leq x \leq 0.4(y + 12.5) \\
p(x,y) = 0.5(7 - x) + 0.2y \text{ for } 0.4(y+12.5) \leq x \leq 0.4(y+17.5) \\
p(x,y) = 0.0 \text{ for } 0.4(y + 17.5) \leq x \leq 20.
\]
This distribution is graphically depicted in Fig.(9).

The present adaptive grid scheme was then applied to this grid and variable distribution. Several results are shown in various combinations of control parameters (Table (1)), including smoothing, allocation values, orthogonality, and adaptation domains (Figs.(10(a) through 10(j)).

The second test case used a sinusoidal distribution of p on the same initial grid, and is defined by
\[
p(x,y) = 0.0 \text{ for } 0 \leq y \leq 9 + 4\sin(2\pi x/21)
\]
INITIAL GRID FOR TEST CASES (21 x 21)
FIG (9)
TEST CASE 1
SURFACE PLOT OF VARIABLE P ON (X,Y) GRID
<table>
<thead>
<tr>
<th>FIG</th>
<th>R1</th>
<th>R2</th>
<th>IDIR</th>
<th>MMIN</th>
<th>MMAX</th>
<th>SWEEPS</th>
<th>ISMT</th>
<th>SWMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>10(a)</td>
<td>0.2</td>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>10(b)</td>
<td>0.2</td>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>10(c)</td>
<td>0.2</td>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>10(d)</td>
<td>0.2</td>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>10(e)</td>
<td>0.2</td>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>2</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>10(f)</td>
<td>0.2</td>
<td>0.1</td>
<td>0</td>
<td>3</td>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>10(g)</td>
<td>0.4</td>
<td>0.1</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>10(h)</td>
<td>0.05</td>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**TABLE 1**

IMPLEMENTATION PARAMETERS FOR TEST CASES 1 AND 2

(IDIR FOR TEST CASE 2 IS REVERSED)
FIG(10) - TEST CASE 1
\[ p(x,y) = 0.5[y-9-4\sin(2\pi x/21)] \text{ for } 9+4\sin(2\pi x/21) \leq y \leq \]
\[ 11+4\sin(2\pi x/21) \]
\[ p(x,y) = 1.0 \text{ for } 11 + 4\sin(2\pi x/21) \leq y \leq 20. \]

and is shown graphically in Fig.(11).

Several different applications are similarly shown for this distribution, as given in Table (1) and in Figs.(12(a) through 12(j)). These figures are shown only for graphic representation of the effect of the different control parameters. The most beneficial combination of these parameters (particularly R1 and R2) are application dependent.

IV-2. Implementation in Existing Codes

The adaptive grid module may generally be applied in existing problem codes by the addition of subroutine CALLs and a COMMON statement in the main program. A typical sequence of CALL statements is illustrated in Fig.(13). A summary of the pertinent implementation parameters is given in Appendix A.

The AG routines were added to a problem code developed by Dr. C. P. Li (NASA/JSC). This code solves the Navier-Stokes or Euler equations for a shock reflection problem. A simple diagram depicting a typical case is shown in Fig.(14).
FIG (11)
TEST CASE 2
SURFACE PLOT OF VARIABLE P ON (X, Y) GRID
FIG(12) - TEST CASE 2.
(1) AGRID  ADAPTATION SWEEP
(2) AGORTH  ORTHOGONALIZATION SWEEP
(3) AGINTC  DETERMINE INTERPOLATION FUNCTIONS
(4) AGINTF  INTERPOLATE VARIABLE - V1
  .  .  .
(5) AGINTF  INTERPOLATE VARIABLE - VM

FIG (13)
TYPICAL CALL SEQUENCE AFTER
COMMON AND PARAMETER SPECIFICATIONS
FIG (14)
DIAGRAM OF EXAMPLE SHOCK REFLECTION PROBLEM
An oblique shock wave is generated somehow, say by a wedge, travels across the calculation domain, and is reflected from a wall parallel to the free stream. The calculation domain does not include the wedge. The time-marching process follows the shock from the wedge to the wall, and is terminated when a time-asymptotic or steady-state solution is achieved.

Output generated by the code includes pressure contours in the calculation domain, and the pressure ratio distribution on the wall (\(Y=0\)) and at a point halfway between the wedge and the wall (\(Y=H/2\)). Output data are shown for calculations with and without an adaptive grid, all other initial data remaining the same. The fixed grid solutions are shown in Figs. (15(a) through 15(j)) at various time steps. Calculations which include the adaptive grid are shown in Figs. (16(a) through 16(j)) at the same time steps. The modified grids are also shown for these cases. It can be seen that the shock is smeared over a relatively thick region in the fixed grid case, but is resolved over a thin region with implementation of the adaptive grid scheme. The latter is a more realistic physical solution.

The adaptive grid case requires more time steps to reach steady-state due to the explicit solution of the Euler equations. The time step increment in this formulation is a function of the grid spacing, thus the clustered grid
imposes a smaller time step increment in the solution than the coarse grid. For an implicit formulation, this would not be the case, since the time steps are independent of the grid spacing.

The adaptive grid was employed with weight parameters $R_1=0.4$ and $R_2=0.0$. Adaptation according to pressure data was performed in the $\eta$ direction ($IDIR=1$), with orthogonalization performed along lines of constant $\xi$ through 350 time steps. For this problem, it was necessary to redefine the points on the left boundary after each adaptation by assigning their $Y$ values to be those of their nearest neighbor (the next node on the constant $\eta$ line). This was necessary to reduce skewness of the adapted grid in this interval. There were no pressure gradients on this line due to the geometry of the problem, and thus the points were not moved by the adaptation process. After 350 time steps (after the reflected shock is well developed), adaptation was performed in the $\xi$ direction ($IDIR=0$) in order to further resolve both the incident and reflected shocks.

Addition of the COMMON/ADAPT/ block, XOLD and YOLD array loading, subroutine CALLs, and the left boundary redefinition process were the only modifications required in the existing code to invoke the adaptive grid.
FIG 15(a)
FIXED GRID SOLUTION - 0 TIME STEPS
FIG 15(b)
FIXED GRID SOLUTION - 50 TIME STEPS
FIG 15(c)
FIXED GRID SOLUTION - 100 TIME STEPS
FIG 15(d)
FIXED GRID SOLUTION - 150 TIME STEPS
FIG 15(e)
FIXED GRID SOLUTION - 200 TIME STEPS
FIG 15(f)
FIXED GRID SOLUTION - 250 TIME STEPS
FIG 15(g)
FIXED GRID SOLUTION - 300 TIME STEPS
FIG 15(h)
FIXED GRID SOLUTION - 350 TIME STEPS
FIG 15(i)
FIXED GRID SOLUTION - 400 TIME STEPS
FIG 15(j)
FIXED GRID SOLUTION - 450 TIME STEPS
FIG 16(a)
ADAPTIVE GRID SOLUTION - 0 TIME STEPS
FIG 16(b)
ADAPTIVE GRID SOLUTION - 50 TIME STEPS
FIG 16(c)
ADAPTIVE GRID SOLUTION - 100 TIME STEPS
FIG 16(d)
ADAPTIVE GRID SOLUTION - 150 TIME STEPS
FIG 16(e)
ADAPTIVE GRID SOLUTION - 200 TIME STEPS
FIG 16(f)
ADAPTIVE GRID SOLUTION - 250 TIME STEPS
FIG 16(g)
ADAPTIVE GRID SOLUTION - 300 TIME STEPS
FIG 16(h)
ADAPTIVE GRID SOLUTION - 350 TIME STEPS
FIG 16(i)
ADAPTIVE GRID SOLUTION - 400 TIME STEPS
FIG 16(j)
ADAPTIVE GRID SOLUTION - 450 TIME STEPS
V. SUMMARY AND CONCLUSIONS

A two-dimensional orthogonal adaptive grid module which may be applied to existing fluid dynamics and heat transfer codes has been presented. This module consists of several subroutines which perform specific functions. Two unidirectional sweeps are available, one which performs a one-dimensional adaptation and the other an orthogonal curve fitting procedure. Interpolation of problem dependent variables onto the modified grid may also be performed through the adaptive grid module. The versatility of this module lies in the various combinations of adaptation and orthogonality allowed by these subroutines. Detailed descriptions of each of the subroutines include the development of the equations, program listings and variable definitions.

Modifications required to implement this module in existing codes are minimal. The adaptive grid routines are invoked by the addition of simple CALL statements in the problem code. Several test case examples show the effect of the various subroutine arguments. An example application in an existing code shows a marked improvement in shock wave resolution with the use of the adaptive grid module compared to the same application without an adaptive grid.
Extension of the module to one- or three-dimensions may be accomplished simply by changing a direction control subroutine and by the removal or addition of a dimension from certain arrays. The three-dimensional extension may prove effective even though the grid adaptation is performed in only one physical space direction. A one-dimensional version (elimination of the two-dimensional arrays) may be a very useful tool in the solution of certain partial differential equations.

The design of this module will allow for modifications for specific applications or for performance improvements. Several changes to the numerical methods used throughout the subroutines are possible. Some control of the minimum cell size may prove effective in problems where the grid is continually updated. Higher order interpolation schemes or curve fitting improvements may also be desired. For certain problems, it may also be necessary to replace the one-dimensional adaptation sweep with a two-dimensional scheme.

Use of this orthogonal adaptive grid module in general fluid dynamics and heat transfer codes will increase the accuracy of the numerical approximation without extremely fine grids, or prior knowledge of the solution.
REFERENCES


APPENDIX A: IMPLEMENTATION PARAMETERS

A-1. Subroutine AGINTC

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINT</td>
<td>Interpolation index array</td>
<td>COMMON</td>
</tr>
<tr>
<td>FINT</td>
<td>Interpolation function array</td>
<td>COMMON</td>
</tr>
<tr>
<td>II</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>IMAX</td>
<td>Interpolation domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>IMIN</td>
<td>Interpolation domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>JJ</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>JMAX</td>
<td>Interpolation domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>JMIN</td>
<td>Interpolation domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>X</td>
<td>X grid point array</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>XOLD</td>
<td>X grid point hold array</td>
<td>COMMON</td>
</tr>
<tr>
<td>Y</td>
<td>Y grid point array</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>YOLD</td>
<td>Y grid point hold array</td>
<td>COMMON</td>
</tr>
</tbody>
</table>
### A-2. Subroutine AGINTF

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINT</td>
<td>Interpolation indices array</td>
<td>COMMON</td>
</tr>
<tr>
<td>FINT</td>
<td>Interpolation function array</td>
<td>COMMON</td>
</tr>
<tr>
<td>II</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>JJ</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>VAR</td>
<td>Interpolated variable</td>
<td>ARGUMENT</td>
</tr>
</tbody>
</table>
### A-3. Subroutine AGORTH

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDIR</td>
<td>Direction indicator</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>II</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>IMAX</td>
<td>Orthogonal domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>IMIN</td>
<td>Orthogonal domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>JJ</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>JMAX</td>
<td>Orthogonal domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>JMIN</td>
<td>Orthogonal domain index</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>SLOPE</td>
<td>Slope tolerance to zero</td>
<td>DATA</td>
</tr>
<tr>
<td>SWMF</td>
<td>Streamwise movement factor</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>X</td>
<td>X grid point array</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>Y</td>
<td>Y grid point array</td>
<td>ARGUMENT</td>
</tr>
</tbody>
</table>
### A-4. Subroutine AGRID

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPS</td>
<td>Convergence tolerance</td>
<td>DATA</td>
</tr>
<tr>
<td>IDIR</td>
<td>Direction indicator</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>II</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>ISMT</td>
<td>Smoothing flag</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>ITMAX</td>
<td>Iteration limit</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>JJ</td>
<td>Maximum array dimension</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>MAX</td>
<td>Greater of II or JJ</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>MMAX</td>
<td>Adaptive domain boundary</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>MMIN</td>
<td>Adaptive domain boundary</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>R1</td>
<td>Allocation parameter</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>R2</td>
<td>Allocation parameter</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>V</td>
<td>Dependent variable array</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>X</td>
<td>X grid point array</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>XOLD</td>
<td>X grid point hold array</td>
<td>COMMON</td>
</tr>
<tr>
<td>Y</td>
<td>Y grid point array</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>YOLD</td>
<td>Y grid point hold array</td>
<td>COMMON</td>
</tr>
</tbody>
</table>
APPENDIX B: VARIABLE DEFINITIONS

B-1. Subroutine AGDIR

VARIABLE DEFINITION

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Array index</td>
</tr>
<tr>
<td>IDIR</td>
<td>Direction indicator</td>
</tr>
<tr>
<td>IM</td>
<td>I - 1</td>
</tr>
<tr>
<td>IP</td>
<td>I + 1</td>
</tr>
<tr>
<td>J</td>
<td>Array index</td>
</tr>
<tr>
<td>JM</td>
<td>J - 1</td>
</tr>
<tr>
<td>JP</td>
<td>J + 1</td>
</tr>
<tr>
<td>M</td>
<td>Inner loop counter index</td>
</tr>
<tr>
<td>MMAX</td>
<td>Adaptive domain boundary index</td>
</tr>
<tr>
<td>MMIN</td>
<td>Adaptive domain boundary index</td>
</tr>
<tr>
<td>N</td>
<td>Outer loop counter index</td>
</tr>
</tbody>
</table>
B-2. Subroutine AGFDE

VARIABLE DEFINITION

DPM First order backward difference
DPP First order forward difference
DX Computational space interval
D1 First derivative, second order difference
D2 Second derivative, second order difference
G Second derivative control function
I Array index
IDIR Direction indicator
II Maximum array index
IM1 I - 1
IM2 I - 2
IP1 I + 1
IP2 I + 2
J Array index
JJ Maximum array index
JM1 J - 1
JM2 J - 2
JP1 J + 1
JP2 J + 2
M Inner loop counter index
MMAX Adaptive domain boundary index
MMIN Adaptive domain boundary index
N Outer loop counter index
P Dependent variable array
B-3. Subroutine AGINTC

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINT</td>
<td>Interpolation triangle index array</td>
</tr>
<tr>
<td>EMIN</td>
<td>Minimum area coordinate</td>
</tr>
<tr>
<td>E1</td>
<td>Area coordinate 1</td>
</tr>
<tr>
<td>E2</td>
<td>Area coordinate 2</td>
</tr>
<tr>
<td>E3</td>
<td>Area coordinate 3</td>
</tr>
<tr>
<td>FINT</td>
<td>Interpolation function array</td>
</tr>
<tr>
<td>I</td>
<td>Array index</td>
</tr>
<tr>
<td>II</td>
<td>Maximum array index</td>
</tr>
<tr>
<td>IMAX</td>
<td>I boundary index</td>
</tr>
<tr>
<td>IMIN</td>
<td>I boundary index</td>
</tr>
<tr>
<td>I1</td>
<td>Triangle node index 1</td>
</tr>
<tr>
<td>I2</td>
<td>Triangle node index 2</td>
</tr>
<tr>
<td>I3</td>
<td>Triangle node index 3</td>
</tr>
<tr>
<td>J</td>
<td>Array index</td>
</tr>
<tr>
<td>JJ</td>
<td>Maximum array index</td>
</tr>
<tr>
<td>JMAX</td>
<td>J boundary index</td>
</tr>
<tr>
<td>JMIN</td>
<td>J boundary index</td>
</tr>
<tr>
<td>J1</td>
<td>Triangle node index 1</td>
</tr>
<tr>
<td>J2</td>
<td>Triangle node index 2</td>
</tr>
<tr>
<td>J3</td>
<td>Triangle node index 3</td>
</tr>
<tr>
<td>TA</td>
<td>1/(2 * triangle area)</td>
</tr>
<tr>
<td>X</td>
<td>X grid point array</td>
</tr>
<tr>
<td>XOLD</td>
<td>Old X grid point array</td>
</tr>
<tr>
<td>X1</td>
<td>X location node 1</td>
</tr>
<tr>
<td>X2</td>
<td>X location node 2</td>
</tr>
<tr>
<td>X3</td>
<td>X location node 3</td>
</tr>
<tr>
<td>Y</td>
<td>Y grid point array</td>
</tr>
<tr>
<td>YOLD</td>
<td>Old Y grid point array</td>
</tr>
<tr>
<td>Y1</td>
<td>Y location node 1</td>
</tr>
<tr>
<td>Y2</td>
<td>Y location node 2</td>
</tr>
<tr>
<td>Y3</td>
<td>Y location node 3</td>
</tr>
</tbody>
</table>
### B-4. Subroutine AGINTF

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINT</td>
<td>Interpolation triangle index array</td>
</tr>
<tr>
<td>FINT</td>
<td>Interpolation function array</td>
</tr>
<tr>
<td>F1</td>
<td>Interpolation function, node 1</td>
</tr>
<tr>
<td>F2</td>
<td>Interpolation function, node 2</td>
</tr>
<tr>
<td>F3</td>
<td>Interpolation function, node 3</td>
</tr>
<tr>
<td>I</td>
<td>Array index</td>
</tr>
<tr>
<td>II</td>
<td>Maximum array index</td>
</tr>
<tr>
<td>I1</td>
<td>Node 1 index</td>
</tr>
<tr>
<td>I2</td>
<td>Node 2 index</td>
</tr>
<tr>
<td>I3</td>
<td>Node 3 index</td>
</tr>
<tr>
<td>J</td>
<td>Array index</td>
</tr>
<tr>
<td>JJ</td>
<td>Maximum array index</td>
</tr>
<tr>
<td>J1</td>
<td>Node 1 index</td>
</tr>
<tr>
<td>J2</td>
<td>Node 2 index</td>
</tr>
<tr>
<td>J3</td>
<td>Node 3 index</td>
</tr>
<tr>
<td>VAR</td>
<td>Interpolated variable array</td>
</tr>
<tr>
<td>VOLD</td>
<td>Working array for VAR</td>
</tr>
</tbody>
</table>
### B-5. Subroutine AGORTH

**VARIABLE DEFINITION**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMB</td>
<td>$A - B$, simultaneous system</td>
</tr>
<tr>
<td>C</td>
<td>$C$, simultaneous system</td>
</tr>
<tr>
<td>D</td>
<td>$D$, simultaneous system</td>
</tr>
<tr>
<td>I</td>
<td>Array index</td>
</tr>
<tr>
<td>IBDY</td>
<td>I boundary flag</td>
</tr>
<tr>
<td>IDIR</td>
<td>Direction indicator</td>
</tr>
<tr>
<td>II</td>
<td>Maximum array index</td>
</tr>
<tr>
<td>IMAX</td>
<td>Orthogonalization boundary index</td>
</tr>
<tr>
<td>IMIN</td>
<td>Orthogonalization boundary index</td>
</tr>
<tr>
<td>IM1</td>
<td>$I - 1$</td>
</tr>
<tr>
<td>IP1</td>
<td>$I + 1$</td>
</tr>
<tr>
<td>J</td>
<td>Array index</td>
</tr>
<tr>
<td>JBDY</td>
<td>J boundary flag</td>
</tr>
<tr>
<td>JJ</td>
<td>Maximum array index</td>
</tr>
<tr>
<td>JMAX</td>
<td>Orthogonalization boundary index</td>
</tr>
<tr>
<td>JMIN</td>
<td>Orthogonalization boundary index</td>
</tr>
<tr>
<td>JM1</td>
<td>$J - 1$</td>
</tr>
<tr>
<td>JP1</td>
<td>$J + 1$</td>
</tr>
<tr>
<td>SLOPE</td>
<td>Line slope tolerance to zero</td>
</tr>
<tr>
<td>SL1</td>
<td>Slope, line 1</td>
</tr>
<tr>
<td>SL2</td>
<td>Slope, line 2</td>
</tr>
<tr>
<td>SWMF</td>
<td>Streamwise movement factor</td>
</tr>
<tr>
<td>TR1</td>
<td>Trajectory slope, line 1</td>
</tr>
<tr>
<td>TR2</td>
<td>Trajectory slope, line 2</td>
</tr>
<tr>
<td>X</td>
<td>X grid point array</td>
</tr>
<tr>
<td>XA</td>
<td>X location, point A</td>
</tr>
<tr>
<td>XAP</td>
<td>X location, point A'</td>
</tr>
<tr>
<td>XB</td>
<td>X location, point B</td>
</tr>
<tr>
<td>XBP</td>
<td>X location, point B'</td>
</tr>
<tr>
<td>XC</td>
<td>X location, point C</td>
</tr>
<tr>
<td>XCP</td>
<td>X location, point C'</td>
</tr>
<tr>
<td>Y</td>
<td>Y grid point array</td>
</tr>
<tr>
<td>YA</td>
<td>Y location, point A</td>
</tr>
<tr>
<td>YAP</td>
<td>Y location, point A'</td>
</tr>
<tr>
<td>YB</td>
<td>Y location, point B</td>
</tr>
<tr>
<td>YBP</td>
<td>Y location, point B'</td>
</tr>
<tr>
<td>YC</td>
<td>Y location, point C</td>
</tr>
<tr>
<td>YCP</td>
<td>Y location, point C'</td>
</tr>
</tbody>
</table>
B-6. Subroutine AGRID

VARIABLE DEFINITION

A  Interpolation segment length, X
AL First derivative multiplier
B  Interpolation segment length, Y
BE Second derivative multiplier
C  Interpolation segment length, P
CON Convergence sum, L2 norm
DM Computational space interval
DMM Arc length scan, lower bracket
DMP Arc length scan, upper bracket
DPA Upwind derivative array
DPB Local derivative array
DS Interpolation ratio
DX X interval
DY Y interval
EPS Convergence tolerance
I  Array index
IB Arc length scan index
IDIR Direction indicator
II Maximum array index
IM I - 1
IP I + 1
IS Arc length bracket index
ISM IS - 1
ISMT Smoothing flag
IT Iteration counter
ITMAX Maximum iteration limit
J  Array index
JJ Maximum array index
JM J - 1
JP J + 1
JS Arc length bracket index
JSM JS - 1
M  Inner loop counter index
MAX Greater of II or JJ
MM Inner loop limit
MMAX Adaptive domain boundary index
MMAXM MMAX - 1
MMIN Adaptive domain boundary index
MMINP MMIN + 1
MM1 MMIN - 1
N  Outer loop counter index
NN Outer loop limit
P  Dependent variable working array
R1 Allocation parameter
R2 Allocation parameter
S  Old arc length vector
SN New arc length vector
TF1M  Quadrature sum, first derivative
TF2M  Quadrature sum, second derivative
V     Dependent variable input array
X     X grid point array
XOLD  X grid point hold array
Y     Y grid point array
YOLD  Y grid point hold array
APPENDIX C: PROGRAM LISTINGS

C-1. Subroutine AGDIR

C==================================================================================================================================
C AGDIR SUBROUTINE - GRID DIRECTION INDICES
C
C VARIABLE DEFINITIONS:
C ARGUMENTS: IDIR = DIRECTION INDICATOR (0=I,1=J) (GIV)
C M = DRIVING INDEX, INNER LOOPS (GIV)
C N = DRIVING INDEX, OUTER LOOPS (GIV)
C MMIN = ADAPTATION DOMAIN MIN BOUNDARY (GIV)
C MMAX = ADAPTATION DOMAIN MAX BOUNDARY (GIV)
C I,J = ARRAY ELEMENT INDICES (RET)
C IM = 1-1 IF IDIR=0 AND I>MMIN (RET)
C JM = J-1 IF IDIR=1 AND I>MMIN (RET)
C IP = I+1 IF IDIR=0 AND I<MMAX (RET)
C JP = J+1 IF IDIR=1 AND I<MMAX (RET)
C (IM,IP),(JM,JP) ARE I,J OTHERWISE
C LOCAL: NONE
C==================================================================================================================================
C
C SUBROUTINE AGDIR (IDIR,M,N,MMIN,MMAX,I,J,IM,JM,IP,JP)
C IF (IDIR.EQ.1) GO TO 100
C
C.....I DIRECTION ADAPTATION INDICES
C I = M
C J = N
C IM = M - 1
C JM = N
C IP = M + 1
C JP = N
C
C.....CHECK BOUNDARY
C IF (M.EQ.MMIN) IM = M
C IF (M.EQ.MMAX) IP = M
C GO TO 200
C
C.....J DIRECTION ADAPTATION INDICES
C 100 I = N
C J = M
C IM = N
C JM = M - 1
C IP = N
C JP = M + 1
C
C.....CHECK BOUNDARY
C IF (M.EQ.MMIN) JM = M
C IF (M.EQ.MMAX) JP = M
C
C 200 CONTINUE
C RETURN
C END
C-2. Subroutine AGFDE

SUBROUTINE AGFDE

- DEPENDENT VARIABLE FINITE DIFFERENCES
- WITH RESPECT TO COMPUTATIONAL COORDINATE
- LOCAL FIRST AND SECOND DERIVATIVE ESTIMATION
- SECOND ORDER FORWARD, CENTRAL, BACKWARD FDE
- I OR J DIRECTION
- CALLS SUBROUTINE AGDIR

VARIABLE DEFINITIONS:

ARGUMENTS:
M = DRIVING INDEX, INNER LOOPS (GIV)
N = DRIVING INDEX, OUTER LOOPS (GIV)
D1 = ABS OF FIRST DERIVATIVE (RET)
D2 = ABS OF SECOND DERIVATIVE * G (RET)
MMIN = ADAPTATION DOMAIN MIN BOUNDARY (GIV)
MMAX = ADAPTATION DOMAIN MAX BOUNDARY (GIV)
IDIR = DIRECTION INDICATOR (0=I, 1=J) (GIV)
P = DEPENDENT VARIABLE P(II,JJ) (GIV)
II = ARRAY DIMENSION (ROWS) (PAR)
JJ = ARRAY DIMENSION (COLS) (PAR)

LOCAL:
DPM = BACKWARD DIFFERENCE FOR G
DPP = FORWARD DIFFERENCE FOR G
DX = COMPUTATIONAL COORD INTERVAL
G = CONTROL FUNCTION FOR 2ND DERIVATIVE
I , J = ARRAY INDICES
IM1,JM1 = ARRAY INDICES (FROM AGDIR)
IP1,JP1 = ARRAY INDICES (FROM AGDIR)
IM2,JM2 = ARRAY INDICES (CALC FUNCTION(IDIR))
IP2,JP2 = ARRAY INDICES (CALC FUNCTION(IDIR))

Coded for VS-FORTRAN-77 AS/9000
Remove parentheses on parameter for FORTRAN-V UNIVAC

SUBROUTINE AGFDE (M,N,D1,D2,MMIN,MMAX,IDIR,P)
PARAMETER (II=21,JJ=21)
DIMENSION P(II,JJ)

Determine Computational Line Interval
DX = 1.0 / (II - 1)
IF (IDIR.EQ.1) DX = 1.0 / (JJ - 1)

Assign Index Values
CALL AGDIR (IDIR,M,N,MMIN,MMAX,I,J,IM1,JM1,IP1,JP1)
JP2 = J + 2 * IDIR
JM2 = J - 2 * IDIR
IP2 = I - 2 * IDIR + 2
IM2 = I + 2 * IDIR - 2
IF (M.NE.MMIN) GO TO 1
D1 = (-P(IP2,JP2) + P(IP1,JP1)*4. - 3.*P(I,J))*0.5/DX
\[
D2 = \frac{((P(IP2, JP2) - P(IP1, JP1))/DX - (P(IP1, JP1) - P(I , J))/DX)}{DX}
\]
\[
DPP = \frac{(P(IP1, JP1) - P(I , J))}{DX}
\]
\[
DPM = 0.0
\]

GO TO 10

C.....BACKWARD DIFFERENCES FOR MMAX BOUNDARY

1 IF (M.NE.MMAX) GO TO 2

\[
D1 = \frac{(P(I,J)*3. - P(IM1, JM1)*4. + P(IM2, JM2))*0.5}{DX}
\]
\[
D2 = (\frac{(P(I , J) - P(IM1, JM1))}{DX} - \frac{(P(IM1, JM1) - P(IM2, JM2))}{DX})}{DX}
\]
\[
DPP = \frac{(P(I , J) - P(IM1, JM1))}{DX}
\]
\[
DPM = \frac{(P(I , J))}{DX}
\]

GO TO 10

C.....CENTRAL DIFFERENCES FOR MMIN < M < MMAX

2 \[
D1 = \frac{(P(IP1, JP1) - P(IM1, JM1))*0.5}{DX}
\]
\[
D2 = \frac{((P(IP1, JP1) - P(I , J)))/DX - (P(IM1, JM1))/DX)}{DX}
\]
\[
DPP = \frac{(P(IP1, JP1) - P(I , J))}{DX}
\]
\[
DPM = \frac{(P(I , J))}{DX}
\]

GO TO 10

C.....DETERMINE CONTROL FUNCTION FOR SECOND DERIVATIVE

10 G = \text{ABS}(DPP - DPM)
\[
D1 = \text{ABS}(D1)
\]
\[
D2 = \text{ABS}(D2) \times G
\]
RETURN

END
C-3. Subroutine AGINTC

AGINTC SUBROUTINE - GRID CELLS AND INTERPOLATION FUNCTIONS
- DETERMINES XOLD, YOLD LOCATION OF X, Y
AND CORRESPONDING INTERPOLATION FUNCTIONS
BY AREA COORDINATES
- COMPLEMENTS AGINTF SUBROUTINE

VARIABLE DEFINITIONS:

ARGUMENTS:  II = ARRAY DIMENSION (PAR)
JJ = ARRAY DIMENSION (PAR)
IMIN = MINIMUM I INDEX (GIV)
IMAX = MAXIMUM I INDEX (GIV)
JMIN = MINIMUM J INDEX (GIV)
JMAX = MAXIMUM J INDEX (GIV)
X = NEW X GRID POINTS (COM)
Y = NEW Y GRID POINTS (COM)
XOLD = OLD X GRID POINTS (COM)
YOLD = OLD Y GRID POINTS (COM)
CINT = INTERPOLATION INDICES ARRAY (COM)
FINT = INTERPOLATION FUNCTIONS ARRAY (COM)

LOCAL:  X1, X2, X3, Y1, Y2, Y3 = TRIANGLE NODE POINTS
TA = 1/DET [JACOBIAN]
E1, E2, E3 = AREA COORDINATES

CODED FOR VS-FORTRAN-77 AS/9000
REMOVE PARENTHESES ON PARAMETER FOR FORTRAN-V UNIVAC

SUBROUTINE AGINTC (IMIN, IMAX, JMIN, JMAX, X, Y)
PARAMETER (II=21, JJ=21)
COMMON/ADAPT/XOLD(II, JJ), YOLD(II, JJ), CINT(II, JJ, 4), FINT(II, JJ, 3)
DIMENSION X(II, JJ), Y(II, JJ)

DO 100 I=IMIN, IMAX
I1 = I
I2 = I + 1
IF (I.EQ.II) I2 = I - 1
J1 = JMIN
J3 = JMIN + 1
IF (J1.EQ.JJ) J3 = JJ - 1

DO 100 J=JMIN, JMAX

DO 100 I3=I1
J2 = J1
X1 = XOLD(I1, J1)
X2 = XOLD(I2, J2)
X3 = XOLD(I3, J3)
Y1 = YOLD(I1, J1)
Y2 = YOLD(I2,J2)
Y3 = YOLD(I3,J3)

C......DETERMINE AREA COORDINATES (INTERPOLATION FUNCTIONS)
TA = 1.0/((X1 - X3) * (Y2 - Y3) - (Y1 - Y3) * (X2 - X3))
E1 = TA * ((X2*Y3-X3*Y2) + (Y2-Y3)*X(I,J) + (X3-X2)*Y(I,J))
E2 = TA * ((X3*Y1-X1*Y3) + (Y3-Y1)*X(I,J) + (X1-X3)*Y(I,J))
E3 = TA * ((X1*Y2-X2*Y1) + (Y1-Y2)*X(I,J) + (X2-X1)*Y(I,J))
IF (ABS(E1).LE.1.E-4) E1 = 0.0
IF (ABS(E2).LE.1.E-4) E2 = 0.0
IF (ABS(E3).LE.1.E-4) E3 = 0.0
EMIN = AMIN1(E1,E2,E3)

C......IF E1,E2,E3 GREATER THAN ZERO, POINT IS IN INTERIOR OF TRIANGLE
    IF (EMIN.GE.0.0) GO TO 50
C
C......IF E1,E2, OR E3 IS NEGATIVE, FLIP TRIANGLE OVER CORRESPONDING SIDE
C
    IF (E1.GE.0.0) GO TO 20
C
C......FLIP TRIANGLE OVER SIDE 1
I1 = I2
I2 = I3.
J1 = J3.
J3 = J2
GO TO 40

20 IF (E2.GE.0.0) GO TO 30
C
C......FLIP TRIANGLE OVER SIDE 2
    IF (I1.EQ.1.OR.I1.EQ.II) GO TO 50
        I2 = I2 + ISIGN(2,(I1-I2))
    30 IF (E3.GE.0.0) GO TO 40
C
C......FLIP TRIANGLE OVER SIDE 3
    IF (J1.EQ.1.OR.J1.EQ.JJ) GO TO 50
        J3 = J3 + ISIGN(2,(J1-J3))
C
C......RETEST WITH NEW TRIANGLE
40 GO TO 5
C
C......TRIANGLE CELL FOUND: STORE COORDS AND INTERPOLATION FUNCTIONS
50 CINT(I,J,1) = I1
CINT(I,J,2) = J1
CINT(I,J,3) = I2
CINT(I,J,4) = J3
FINT(I,J,1) = E1
FINT(I,J,2) = E2
FINT(I,J,3) = E3
100 CONTINUE
RETURN
END
C-4. Subroutine AGINTF

AGINTF SUBROUTINE - INTERPOLATE VARIABLES TO NEW GRID
AGINTF SUBROUTINE - DETERMINES VARIABLE VALUES ON X,Y GRID BY
INTERPOLATION FUNCTIONS FROM AGINTC

VARIABLE DEFINITIONS:
ARGUMENTS: II = ARRAY DIMENSION (PAR)
JJ = ARRAY DIMENSION (PAR)
VAR = VARIABLE TO INTERPOLATE (II,JJ) (MOD)
CINT = INTERPOLATION INDICES ARRAY (COM)
FINT = INTERPOLATION FUNCTION ARRAY (COM)

LOCAL: VOLD = HOLD ARRAY FOR VAR
F1 = NODE 1 INTERPOLATION FUNCTION
F2 = NODE 2 INTERPOLATION FUNCTION
F3 = NODE 3 INTERPOLATION FUNCTION
I1,J1,I3,J1,J2,J3 = NODE INDICES FROM AGINTC

CODED FOR VS-FORTRAN-77 AS/9000
REMOVE PARENTHESES ON PARAMETER FOR FORTRAN-V UNIVAC

SUBROUTINE AGINTF (VAR)
PARAMETER (II=21,JJ=21)
COMMON/ADAPT/XOLD(II,JJ),YOLD(II,JJ),CINT(II,JJ,4),FINT(II,JJ,3)
DIMENSION VAR(II,JJ),VOLD(II,JJ)

DO 50 I=1,II
DO 50 J=1,JJ
50 VOLD(I,J) = VAR(I,J)

DO 100 I=1,II
DO 100 J=1,JJ
I1 = CINT(I,J,1)
I2 = CINT(I,J,3)
I3 = I1
J1 = CINT(I,J,2)
J2 = J1
J3 = CINT(I,J,4)
F1 = FINT(I,J,1)
F2 = FINT(I,J,2)
F3 = FINT(I,J,3)
100 VAR(I,J) = F1 * VOLD(I1,J1) + F2 * VOLD(I2,J2) + F3 * VOLD(I3,J3)
RETURN
END
C-5. Subroutine AGORTH

C-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
C AGORTH SUBROUTINE - ORTHOGONALIZATION ROUTINE
C - APPROXIMATE ORTHOGONAL TRAJECTORIES
C - ORTHOGONAL LINES FITTED TO ADAPTED LINES
C - I OR J DIRECTION (CONSTANT ETA, XI)
C
C A) NORMAL CONSTRUCTED FROM LINE 1, POINT A, TO LINE 2, POINT B
C B) NORMAL CONSTRUCTED FROM LINE 2, POINT B, TO LINE 1, POINT C
C C) NORMAL CONSTRUCTED FROM LINE 2, POINT AP, TO LINE 1, POINT BP
C D) NORMAL CONSTRUCTED FROM LINE 1, POINT BP, TO LINE 2, POINT CP
C E) NEW ORTHOGONAL POINTS DETERMINED FROM
C POINT A = (POINT BP + POINT C)/2
C POINT B = (POINT CP + POINT B)/2
C F) POINT MOVEMENT ON ADAPTED LINES SCALED BY SWMF

C VARIABLE DEFINITIONS:
C
C ARGUMENTS: IMIN = I BOUNDARY MINIMUM (GIV)
C IMAX = I BOUNDARY MAXIMUM (GIV)
C JMIN = J BOUNDARY MINIMUM (GIV)
C JMAX = J BOUNDARY MAXIMUM (GIV)
C SWMF = STREAMWISE MOVEMENT FACTOR (0 TO 1) (GIV)
C IDIR = DIRECTION INDICATOR (0=I, 1=J) (GIV)
C II = ARRAY DIMENSION (ROW) (PAR)
C JJ = ARRAY DIMENSION (COL) (PAR)
C SLOPE = SLOPE TOLERANCE TO 0 (PAR)
C X = GRID POINT ARRAY X(II,JJ) (COM)
C Y = GRID POINT ARRAY Y(II,JJ) (COM)
C
C LOCAL : AMB = A - B FOR SIMULTANEOUS EQNS
C C = C FOR SIMULTANEOUS EQNS
C D = D FOR SIMULTANEOUS EQNS
C IBDY = I BOUNDARY FLAG
C JBDY = J BOUNDARY FLAG
C I ,J = ARRAY INDICES
C IP1,JP1 = ARRAY INDICES
C IM1,JM1 = ARRAY INDICES
C SL1 = SLOPE OF LINE 1
C SL2 = SLOPE OF LINE 2
C TR1 = SLOPE OF NORMAL TO LINE 1
C TR2 = SLOPE OF NORMAL TO LINE 2
C XA ,YA = POINT A COORDINATES
C XB ,YB = POINT B COORDINATES
C XC ,YC = POINT C COORDINATES
C XAP,YAP = POINT AP COORDINATES
XBP,YBP = POINT BP COORDINATES
XCP,YCP = POINT CP COORDINATES

CODED FOR VS-FORTRAN-77 AS/9000
REMOVE PARENTHESES ON PARAMETER FOR FORTRAN-V UNIVAC

SUBROUTINE AGORTH (IMIN,IMAX,JMIN,JMAX,SWMF,IDIR,X,Y)
PARAMETER (II=21,JJ=21)
COMMON/ADAPT/XOLD(II,JJ),YOLD(II,JJ),CINT(II,JJ,4),FINT(II,JJ,3)
DIMENSION X(II,JJ),Y(II,JJ)
DATA SLOPE/0.1/
IF (IDIR.EQ.1) GO TO 11

C.....CONSTANT ETA DIRECTION ORTHOGONALIZATION

DO 10 I=IMIN+1,IMAX
IM1 = I - 1
DO 10 J=JMIN,JMAX
JP1 = J + 1
JM1 = J - 1
JBDY= 0

C.....CHECK BOUNDARY
IF (J.EQ.JMAX.OR.J.EQ.JMIN) JBDY=1
IF (J.EQ.JMAX) JP1=J
IF (J.EQ.JMIN) JM1=J
TR1 = 0.0
TR2 = 0.0
SL1 = (X(IM1,JP1) - X(IM1,JM1))/(Y(IM1,JP1) - Y(IM1,JM1))
SL2 = (X( I ,JP1) - X( I ,JM1))/(Y( I ,JP1) - Y( I ,JM1))
IF (ABS(SL1).GT.SLOPE) TR1 = -1.0/SL1
IF (ABS(SL2).GT.SLOPE) TR2 = -1.0/SL2

C.....POINT A,AP
XA = X(IM1,J)
YA = Y(IM1,J)
XAP = X( I ,J)
YAP = Y( I ,J)

C.....POINT B
YB = YA
XB = SL2 * (YB - YAP) + XAP
IF (TR1.EQ.0.0) GO TO 2
AMB = TR1 - SL2
C = -TR1 * Y(IM1,J) + X(IM1,J)
D = -SL2 * Y( I ,J) + X( I ,J)
IF (I.NE.IMAX) XB = (TR1 * D - C * SL2)/AMB
IF (JBDY.EQ.0) YB = (D - C)/AMB

C.....POINT C
2 YC = YB
XC = SL1 * (YC - YA) + XA
IF (TR2.EQ.0.0) GO TO 4
AMB = TR2 - SL1
C = -TR2 * YB + XB
D = -SL1 * Y(I,J) + X(I,J)
IF (IM1.NE.IMIN) XC = (TR2 * D - C * SL1)/AMB
IF (JBDY.EQ.0) YC = (D - C)/AMB

C.....POINT BP

4 YBP = YAP
XBP = SL1 * (YBP - YA) + XA
IF (TR2.EQ.0.0) GO TO 6
AMB = TR2 - SL1
C = -TR2 * Y(I,J) + X(I,J)
D = -SL1 * Y(IM1,J) + X(IM1,J)
IF (IM1.NE.IMIN) XBP = (TR2 * D - C * SL1)/AMB
IF (JBDY.EQ.0) YBP = (D - C)/AMB

C.....POINT CP

6 YCP = YBP
XCP = SL2 * (YCP - YAP) + XA
IF (TR1.EQ.0.0) GO TO 8
AMB = TR1 - SL2
C = -TR1 * YBP + XBP
D = -SL2 * Y(I,J) + X(I,J)
IF (I.NE.IMAX) XCP = (TR1 * D - C * SL2)/AMB
IF (JBDY.EQ.0) YCP = (D - C)/AMB

C.....REASSIGN X,Y ARRAY VALUES

C X DISPLACEMENTS WEIGHTED BY SWMF
8 X(IM1,J) = X(IM1,J) + SWMF*(0.5 * (XBP + XC) - X(IM1,J))
Y(IM1,J) = (YBP + YC) * 0.5
X(I,J) = X(I,J) + SWMF*(0.5 * (XCP + XB) - X(I,J))
Y(I,J) = (YCP + YB) * 0.5
10 CONTINUE
GO TO 100

C C.....CONSTANT XI DIRECTION ORTHOGONALIZATION
C
11 CONTINUE
DO 100 J = JMIN+1,JMAX
JM1 = J - 1
DO 100 I = IMIN,IMAX
IP1 = I + 1
IM1 = I - 1
IBDY = 0.0

C.....CHECK BOUNDARY
 IF (I.EQ.IMIN.OR.I.EQ.IMAX) IBDY = 1
 IF (I.EQ.IMAX) IP1 = I
 IF (I.EQ.IMIN) IM1 = I
 TR1 = 0.0
 TR2 = 0.0
 SL1 = (Y(IP1,JM1) - Y(IM1,JM1))/(X(IP1,JM1) - X(IM1,JM1))
 SL2 = (Y(IP1, J ) - Y(IM1, J ))/(X(IP1, J ) - X(IM1, J ))
 IF (ABS(SL1).GT.SLOPE) TR1 = -1.0/SL1
 IF (ABS(SL2).GT.SLOPE) TR2 = -1.0/SL2

C.....POINT A,AP
XA = X(I,JM1)
YA = Y(I,JM1)
XAP = X(I, J )
YAP = Y(I, J )

C.....POINT B
XB = XA
YB = SL2 * (XB - XAP) + YAP
IF (TR1.EQ.0.0) GO TO 20
AMB = TR1 - SL2
C = -TR1 * X(I,JM1) + Y(I,JM1)
D = -SL2 * X(I, J ) + Y(I, J )
IF (J.NE.JMAX) YB = (TR1 * D - C * SL2)/AMB
IF (IBDY.EQ.0) XB = (D - C)/AMB

C.....POINT C
20XC = XB
YC = SL1 * (XC - XA) + YA
IF (TR2.EQ.0.0) GO TO 40
AMB = TR2 - SL1
C = -TR2 * XB + YB
D = -SL1 * X(I,J) + Y(I,J)
IF (JM1.NE.JMIN) YC = (TR2 * D - C * SL1)/AMB
IF (IBDY.EQ.0) XC = (D - C)/AMB

C.....POINT BP
40XBP = XAP
YBP = SL1 * (XBP - XA) + YA
IF (TR2.EQ.0.0) GO TO 60
AMB = TR2 - SL1
C = -TR2 * X(I, J ) + Y(I, J )
D = -SL1 * X(I,JM1) + Y(I,JM1)
IF (JM1.NE.JMIN) YBP = (TR2 * D - C * SL1)/AMB
IF (IBDY.EQ.0) XBP = (D - C)/AMB

C.....POINT CP
60XCP = XBP
YCP = SL2 * (XCP - XAP) + YAP
IF (TR1.EQ.0.0) GO TO 80
AMB = TR1 - SL2
C = -TR1 * XBP + YBP
D = -SL2 * X(I,J) + Y(I,J)
IF (J.NE.JMAX) YCP = (TR1 * D - C * SL2)/AMB
IF (IBDY.EQ.0) XCP = (D - C)/AMB

C.....REASSIGN X,Y ARRAY VALUES
C
Y DISPLACEMENTS WEIGHTED BY SWMF
80 X(I,JM1) = (XBP + XC) * 0.5
Y(I,JM1) = Y(I,JM1) + SWMF * (0.5 * (YBP + YC) - Y(I,JM1))
X(I, J ) = (XCP + XB) * 0.5
Y(I, J ) = Y(I, J ) + SWMF * (0.5 * (YCP + YB) - Y(I, J ))

100 CONTINUE
RETURN
END
C-6. Subroutine AGRID

C--------------------------------------------------------------------------------------
C AGRID SUBROUTINE - ADAPTIVE GRID ROUTINE
C - ADAPTATION ALONG FIXED COMPUTATIONAL COORD
C - VARIATIONAL EQUIDISTRIBUTION SCHEME
C - 2 DIMENSIONAL GRIDS, 1 DIMENSION ADAPTATION
C - I OR J DIRECTION (CONSTANT XI, ETA)
C - CALLS SUBROUTINES AGFDE, AGDIR

C VARIABLE DEFINITIONS:
C ARGUMENTS : R1 = 1ST DERIVATIVE ALLOCATION (GIV)
C R2 = 2ND DERIVATIVE ALLOCATION (GIV)
C MMIN = ADAPTATION DOMAIN BOUNDARY (MIN) (GIV)
C MMAX = ADAPTATION DOMAIN BOUNDARY (MAX) (GIV)
C IDIR = DIRECTION INDICATOR (0=I,1=J) (GIV)
C ISMT = SMOOTHING FLAG (0=NO SMOOTHING) (GIV)
C V = DEPENDENT VARIABLE FOR ADAPTATION (GIV)
C II = ARRAY DIMENSION (PAR)
C JJ = ARRAY DIMENSION (PAR)
C MAX = MAXIMUM ARRAY DIMENSION (II OR JJ)(PAR)
C EPS = CONVERGENCE TOLERANCE (PAR)
C ITMAX = MAXIMUM ITERATIONS (PAR)
C X = NEW GRID POINT LOCATIONS X (COM)
C Y = NEW GRID POINT LOCATIONS Y (COM)
C XOLD = OLD GRID POINT LOCATIONS X (COM)
C YOLD = OLD GRID POINT LOCATIONS Y (COM)

C LOCAL : DPA = DERIVATIVE ARRAY (1=1ST,2=2ND) (M-1)
C DPB = DERIVATIVE ARRAY (1=1ST,2=2ND) ( M )
C S = OLD ARC LENGTH ARRAY (MAX)
C SN = NEW ARC LENGTH ARRAY (MAX)
C AL = 1ST DERIVATIVE WEIGHT FUNCTION PARAM
C BE = 2ND DERIVATIVE WEIGHT FUNCTION PARAM
C CON = CONVERGENCE TOLERANCE
C DM = COMPUTATIONAL COORDINATE INTERVAL
C I,J = ARRAY INDICES
C IM,JM = ARRAY INDICES (FROM AGDIR)
C IP,JP = ARRAY INDICES (FROM AGDIR)
C IS,JS = ARRAY INDICES (FROM AGDIR)
C IT = ITERATION COUNTER
C M = INNER LOOP INDEX (DIRECTION DEPENDENT)
C N = OUTER LOOP INDEX (DIRECTION DEPENDENT)
C MM = MAX INNER LOOP (DIRECTION DEPENDENT)
C NN = MAX OUTER LOOP (DIRECTION DEPENDENT)
C MM1 = M - 1
C TF1M = QUADRATURE SUM 1ST DERIVATIVE
C TF2M = QUADRATURE SUM 2ND DERIVATIVE
C XM = NEW X LOCATION
C YM = NEW Y LOCATION
C
SUBROUTINE AGRID (R1,R2,MIN,MMA,DIR,ISMT,X,Y,V)
PARAMETER (II=21,JJ=21,MAX=21,ITMAX=10)
DIMENSION DPA(MAX,2),DPB(MAX,2),S(MAX),SN(MAX),V(II,JJ),P(II,JJ),
   *X(II,JJ),Y(II,JJ)
COMMON/ADAPT/XOLD(II,JJ),YOLD(II,JJ),CINT(II,JJ,4),FINT(II,JJ,3)
DATA EPS/0.001/
NN = JJ
MM = II
IF (IDIR.EQ.1) NN = II
IF (IDIR.EQ.1) MM = JJ

C.....DUMP CURRENT GRID ARRAYS TO OLD GRID ARRAYS (FOR INTERPOLATION)
C-->
DO 20 I=1,II
DO 20 J=1,JJ
  P (I,J) = V(I,J)
  XOLD(I,J) = X(I,J)
  YOLD(I,J) = Y(I,J)
20 CONTINUE
IF (ISMT.EQ.0) GO TO 30
C.....SMOOTH DEPENDENT VARIABLE STREAMWISE GRADIENTS
C-->
25 DO 30 N=1,NN
   DO 30 M=MIN,MMA
      CALL AGRID (IDIR,M,N,MIN,MMA,I,J,IM,JM,IP,JP)
      P(I,J)= (P(IM,JM) + 2.0 * P(I,J) + P(IP,JP)) * 0.25
30 CONTINUE
C.....SET COMPUTATIONAL SPACE INTERVAL AND LOOP BOUNDS
DM = 1.0/(MM - 1.0)
MINP = MIN + 1
MAXM = MAX - 1
C
C.....OUTER LOOP: FIX CONSTANT COMPUTATIONAL GRID LINE
C-->
DO 100 N=1,NN
   IT = 0
   S (MIN)= 0.0
   SN(MIN)= 0.0
C.....ASSIGN GUESS VALUES TO SN
C-->
   DO 1 M = MINP,MMA
      S(M) = 0.0
      MM1 = M-1
      CALL AGRID (IDIR,M,N,MIN,MMA,I,J,IM,JM,IP,JP)
      DX = (X(I,J) - X(IM,JM)) * (X(I,J) - X(IM,JM))
      DY = (Y(I,J) - Y(IM,JM)) * (Y(I,J) - Y(IM,JM))
      1 SN(M)= SN(MM1) + (DX + DY)**0.5
C
C>>> ITERATION RETURN POINT
  2 IT = IT + 1
  CON = 0.0
C.....SCALE SN VECTOR, DETERMINE TOTAL ERROR, REASSIGN S VECTOR
C-->
  DO 4 M = MMINP, MMAX
  SN(M) = SN(M)/SN(MMAX)
  CON = CON+(SN(M)-S(M))*(SN(M)-S(M))
  4 S(M) = SN(M)
C.....CHECK SOLUTION CONVERGENCE
  IF (CON.LE.EPS) GO TO 100
C.....CLEAR QUADRATURE SUMS
  TF1M = 0.0
  TF2M = 0.0
C.....SET QUADRATURE SUMS
C-->
  DO 50 M = MMINP, MMAX
C.....DETERMINE LOCAL DERIVATIVE VALUES
  CALL AGFDE (M-1,N,DPA(M,1),DPA(M,2),MMIN,MMAX,IDIR,P)
  CALL AGFDE (M,N,DPB(M,1),DPB(M,2),MMIN,MMAX,IDIR,P)
C.....DETERMINE LOCAL QUADRATURE SUM
  TF1M = TF1M + 0.5 * DM * (DPA(M,1) + DPB(M,1))
  TF2M = TF2M + 0.5 * DM * (DPA(M,2) + DPB(M,2))
  50 CONTINUE
C.....DETERMINE GRADIENT WEIGHT PARAMETERS AL, BE
  AL = 0.0
  BE = 0.0
  IF (TF1M.GT.0.0) AL = R1 / (1.0 - R1) / TF1M
  IF (TF2M.GT.0.0) BE = R2 / (1.0 - R2) / TF2M * (1.0 + AL * TF1M)
C.....QUADRATURE LOOP: DETERMINE NEW ARC LENGTH VECTOR
C-->
  DO 60 M = MMINP, MMAX
  MM1 = M-1
  DSN = 0.5*((1+AL*DPA(M,1)+BE*DPB(M,1)) +
             (1+AL*DPB(M,1)+BE*DPA(M,2)))
  SN(M) = SN(M) + DM/DSN
  60 CONTINUE
C.....SCALE ARC LENGTH VECTOR, INTERPOLATE NEW X,Y,P
C-->
  DO 80 M = MMINP, MMAX
  SN(M) = SN(M)/SN(MMAX)
  CALL AGDIR(IDIR,M,N,MMIN,MMAX,IS,JS,ISM,JSM,IP,JP)
C.....FIND COMPUTATIONAL SPACE LINE SEGMENT
  62 DMM = (IB - MMINP) * 1.0/(MMAX - MMIN)
  DMP = (IB - MMIN) * 1.0/(MMAX - MMIN)
  IF (SN(M).GE.DMM.AND.SN(M).LT.DMP) GO TO 63
  IB = IB + ISIGN(1,INT(SN(M) - DMM))
  GO TO 62
C.....PIECEWISE LINEAR INTERPOLATION
  63 CALL AGDIR(IDIR,IB,N,MMIN,MMAX,IS,JS,ISM,JSM,IP,JP)
DS = (SN(M) - DMM) / (DMP - DMM)
A = XOLD(IS, JS) - XOLD(ISM, JSM)
B = YOLD(IS, JS) - YOLD(ISM, JSM)
C = P(IS, JS) - P(ISM, JSM)
X(I,J) = XOLD(ISM, JSM) + A * DS
Y(I,J) = YOLD(ISM, JSM) + B * DS
P(I,J) = P(ISM, JSM) + C * DS

CONTINUE

IF (IT.LT.ITMAX) GO TO 2

CONTINUE
RETURN
END