NASA Technical Memorandum 103905

The Multidimensional Self-Adaptive Grid Code, SAGE

Carol B. Davies, Sterling Software, Palo Alto, California
Ethiraj Venkatapathy, Eloret Institute, Sunnyvale, California

July 1992

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

TABLE OF CONTENTS

THE MULTIDIMENSIONAL SELF-ADAPTIVE GRID CODE, SAGE

Carol B. Davies* and Ethiraj Venkatapathy†
Ames Research Center

SUMMARY

This report describes the multidimensional self-adaptive grid code SAGE. A two-dimensional version of this code was described in an earlier report by the authors. The formulation of the multidimensional version is described in the first section of this document. The second section is presented in the form of a user guide that explains the input and execution of the code and provides many examples. Successful application of the SAGE code in both two and three dimensions for the solution of various flow problems has proven the code to be robust, portable, and simple to use. Although the basic formulation follows the method of Nakahashi and Deiwert, many modifications have been made to facilitate the use of the self-adaptive grid method for complex grid structures. Modifications to the method and the simplified input options make this a flexible and user-friendly code. The new SAGE code can accommodate both two-dimensional and three-dimensional flow problems.

1. METHOD

1.1 Introduction

Solution-adaptive grid methods have become useful tools for efficient and accurate flow predictions. In supersonic and hypersonic flows, strong gradient regions such as shocks, contact discontinuities, and shear layers require careful distribution of grid points to minimize grid error and thus produce accurate flow-field predictions. Frequently, the generation of the first grid topology does not adequately capture these flow structures. It has been shown that an effective way of obtaining accurate solutions is by intelligently redistributing (i.e., adapting) the original grid points based on the initial flow-field solution and then computing a new solution using the adapted grid. The cost efficiency of grid adaptions in terms of CPU time depends on the basic formulation of the adaptive-grid solver. A short historical review and a list of references on the adaptive grid procedure used in the SAGE code is given in Davies and Venkatapathy (1991).

The self-adaptive grid procedure outlined by Nakahashi and Deiwert (1985) has evolved into a flexible tool for adapting and restructuring both two-dimensional (2-D) and three-dimensional (3-D) grids. The adaptive-grid method is based on grid-point redistribution through local error minimization. The procedure is analogous to applying tension and torsion spring forces proportional to the local flow gradient at every grid point and finding the equilibrium position of the resulting system of grid points. The multidimensional problem of grid adaption is split into a series of one-dimensional (1-D) problems along the computational coordinate lines. The reduced 1-D problem then requires a tridiagonal solver to find the location of grid points along a given

* Sterling Software, Palo Alto, Ca.
† Eloret Institute, Sunnyvale, Ca.

coordinate line. Multidirectional adaption is achieved by the sequential application of the method in each coordinate direction.

The tension forces direct the redistribution of points to the strong gradient regions. The torsion forces relate information between the families of lines adjacent to one another, to maintain smoothness and a measure of orthogonality of grid lines. These smoothness and orthogonality constraints are direction-dependent, since they relate only the coordinate lines that are being adapted to the neighboring lines that have already been adapted. This implies that the solutions are nonunique and depend on the order and direction of adaption. Nonuniqueness of the adapted grid is acceptable since it makes possible an overall and local error reduction through grid redistribution.

The Self-Adaptive Grid codE (SAGE) code has been built with many flexible elements that make it user-friendly. The second section of this report is a user guide, which is independent of the first section and can be used as a separate document. However, the nomenclature and details of the grid adaption procedure within the code consistently follow the analysis, allowing the user to understand the code and implement any individual changes, if desired. The user guide describes the input parameters and their effects as well as the adaption procedure, execution commands, and subroutines, and provides many examples.

1.2 Formulation of Adaptive-Grid Scheme

As stated in the introduction, adaption takes place as a series of one-dimensional adaptions. Figure 1 illustrates this concept: three constant k planes of an initial grid are shown in figure 1(a) and a flow-field solution has been obtained using this unadapted grid. The points in this grid are then adapted to the flow solution, starting on the first line ($j = 1$) on the lower plane ($k = 1$). In figure 1(b), the first plane has already been adapted and the second plane is the current adaption plane. The current adaption line (j) is shown, with previous lines already adapted and subsequent lines awaiting adaption. The third plane is still in its original form. Adaption is performed in this line-by-line, plane-by-plane manner until all requested planes are complete. It is then possible to perform an adaption in a second direction, "adapting" on top of the already adapted grid. The number and order of adaptions are arbitrary and depend on the type of flow problem and the purpose of the adaption.

Figure 2 shows a segment of the current adaption line in more detail. The lower plane has already been adapted and the upper plane is currently being adapted. Four forces control the redistribution of points along each line: the two tension springs that act on each side of a node, and the two torsion springs that control the smoothness of the grid. The tension forces (ω) cluster the redistributed points into the high-gradient regions. The torsion forces (τ and ψ) restrain the tension forces to maintain continuity between sequentially adapted lines. As shown in figure 2, the τ force acts from the previously adapted line within the current plane and the ψ force acts from the previously adapted line in the preceding plane.

A 3-D grid is described in terms of its computational coordinate directions (i, j, k) and its physical coordinates (x, y, z). An adaption can take place in any or all of the computational coordinate directions and the SAGE code permits any combination. However, for clarity, this analysis assumes that all adaptions are performed in the i direction and that stepping (within a plane surface) occurs in the j direction. Thus k is the marching direction for plane stepping. The current adaption line is thus j in the plane k where lines $j - 1$, $j - 2$, etc., have already been adapted. In addition, this report uses the term "plane" to mean any 3-D coordinate surface.

The first step in the formulation of the adaption algorithm is to consider the adaption of a single line, with no torsional constraints (i.e., no smoothness control). Figure 3 shows a 2-D surface, where the arc length at A (i.e., $s_{i,j,k}$) along the current adaption line, j, is defined as

$$s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1}) + (z_i - z_{i-1})^2} \tag{1}$$

A tension force, $f(\omega)$, is defined to act between each $i-1$ and i node such that

$$\omega_i \Delta s_i = K \tag{2}$$

where ω_i, the tension constant, is a weighting function based on the flow gradient, and K is the resultant force. To redistribute the points along a line with the minimum solution error, the adaptive procedure defines the weighting function as proportional to the derivative of the flow variable (Nakahashi and Deiwert 1985). In this formulation, ω is defined as a function of the normalized flow gradient, $\bar{f}$, such that

$$\omega_i = 1 + A\bar{f}_i^{\,B} \tag{3}$$

where A and B are constants directly related to the grid spacing and are chosen to maintain the grid intervals to within the limits (Δs_{MIN} and Δs_{MAX}) set by the user on input. They are defined in appendix I of this section.

Equation (2) is written for each node on the line, giving a 1-D formulation that can be solved directly for Δs_i. Taking the sum of both sides of equation (2) gives

$$\sum_{l=1}^{n_i} \Delta s_l = s_{max} = K \sum_{l=1}^{n_i} \frac{1}{\omega_l}$$

giving

$$K = s_{max} \Big/ \sum_{l=1}^{n_i} \frac{1}{\omega_l} \tag{4}$$

Substituting back in equation (2), we obtain

$$\Delta s_i = s_{max} \Big/ \left(\omega_i \sum_{l=1}^{n_i} \frac{1}{\omega_l} \right) \tag{5}$$

In the SAGE code, this 1-D solution technique is used along the initial adaption line of the initial adaption plane, where no directional information is available. Continuing this approach for successive line-by-line adaptions will not create a mesh that is sufficiently smooth for input into computational flow-field codes. To ensure a more reasonable grid, the redistribution of points (driven by the tension springs) is constrained by torsion terms defined on two adjacent adapted lines: one on the current plane and one on the preceding plane. Within the current plane, the torsion parameter τ represents the magnitude of the torsion force that maintains smoothness and orthogonality between the node (i, j, k) and the nodes $(i, j-1, k)$ and $(i, j-2, k)$. This is the only torsion parameter available on the initial surface. For subsequent surfaces, the torsion

parameter ψ constrains the movement between the same node (i,j,k) and the nodes $(i,j,k-1)$ and $(i,j,k-2)$ on adjacent computational planes. The torsion terms are evaluated as $\tau_i(s'_i - s_i)$ and $\psi_i(s^*_i - s_i)$, where the torsion parameters τ and ψ define the magnitude of the torsion forces and s' and s^* define their inclination (i.e., orthogonality and smoothness).

To introduce the torsion forces to the system of equations, we first rewrite equation (2) to represent the force balance,

$$\omega_i(s_{i+1} - s_i) - \omega_{i-1}(s_i - s_{i-1}) = 0 \tag{6}$$

and then add the torsion terms to obtain

$$\omega_i(s_{i+1} - s_i) - \omega_{i-1}(s_i - s_{i-1}) + \tau_i(s'_i - s_i) + \psi_i(s^*_i - s_i) = 0 \tag{7}$$

which is rearranged to produce the coefficients of the tridiagonal matrix used to solve for s_i,

$$\omega_{i-1}s_{i-1} - (\omega_i + \omega_{i-1} + \tau_i + \psi_i)s_i + \omega_i s_{i+1} = -\tau_i s'_i - \psi_i s^*_i \tag{8}$$

This equation is written for each interval along the adaption line, producing a system of $(n_i - 1)$ equations. Since s_1 and s_n are known, there are $n_i - 2$ rows in the matrix. This equation is solved iteratively (updating ω_i at new s_i) until $\sum_{i=1}^{n_i} \left| s_i^{(n)} - s_i^{(n-1)} \right| < 10^{-3} \times s_{max}$.

This system of equations is central to the adaption technique used in the SAGE code and most of the description that follows pertains to deriving the coefficients of this equation.

1.3 Auxiliary Equations

1.3.1 Tension constant, ω_i. As described in the formulation, ω_i acts as a tension force in the interval $(s_i - s_{i-1})$ and can be imagined to be a spring (aligned with the grid line) connecting the two grid points, as shown in figure 3. This tension parameter is defined in equation (3) as

$$\omega_i = 1 + A\bar{f}_i^B$$

where $\bar{f}_i$ is a function of the gradient of the flow variable, q:

$$\bar{f}_i = \frac{f_i - f_{min}}{f_{max} - f_{min}} \text{ and } f_i = \frac{\partial q_i}{\partial s} \tag{9}$$

The standard format of the user's input flow-field file contains the conservative flow variables Q, as defined in the plotting software package PLOT3D, which (for a 3-D dataset) assumes these flow variables are $\rho, \rho u, \rho v, \rho w$, and e. Since the adaption is based on a scalar function q, this function can be evaluated as a user-specified combination of flow variables Q. Pressure, Mach number and temperature ratio are also included as adaption variables and are internally computed, assuming perfect gas relations. However, Q need not be restricted to conservative flow variables, but can be any combination of user-specified flow variables that represent the flow field.

To remove the unwanted oscillations from the discrete evaluation, f_i is smoothed by adding a second derivative term, i.e., $f_i = .75f_i + .125(f_{i+1} - f_{i-1})$. By default, two smoothing passes are made, but this can be overridden by changing the filter input control parameter. The tension parameter, ω_i, is smoothed in the same way.

1.3.1.1 A and B. As seen in equation (3), ω is also a function of A and B. These variables are the important elements of the self-adaptive nature of the scheme because they control the size limits of the grid spacing. A and B are computed from the user-supplied maximum and minimum allowable grid spacings ($\Delta s_{MAX}, \Delta s_{MIN}$). These are relative values of mesh size and allow the user to specify the proportion of largest to smallest Δs in the final adapted grid.

The value of A is constant throughout the grid adaption and is given by

$$A = \frac{\Delta s_{MAX}}{\Delta s_{MIN}} - 1 \tag{10}$$

The value of B is computed (by an iterative process) for each j line to provide

$$input\ \Delta s_{MIN} = computed\ \Delta s_{min}$$

That is, the computed minimum grid spacing is equal to the user-requested value. Appendix I gives a detailed description of the derivation of these two parameters.

1.3.1.2 Torsional effect on the evaluation of ω. As just noted, the function of variables A and B is to control the size limits of the adapted-grid mesh spacing and, as described in appendix I, B is computed from the 1-D equation (2). Thus the addition of the torsion terms is somewhat inconsistent with the value of B, and the resulting grid spacings may give values slightly outside the requested limits. To provide for additional control, a weighting factor is applied to ω such that $\omega_i = (1 + A\bar{f}^B)\omega_{t_i}$, where ω_{t_i} is the weighted tightness factor. When equation (8) is solved, each Δs_i is tested and if it does not lie within the requested spacing limits, ω_{t_i} will be computed; otherwise $\omega_{t_i} = 1.0$. If $\Delta s_i < \Delta s_{MIN}$, we need to relax (decrease) the value of ω_i to produce a larger Δs_i in the next iteration. If Δs_i is too large, ω_i should be increased. We therefore use the modifier

$$\omega_{t_i} = \frac{1}{2}\left(\frac{\Delta s_i}{\Delta s_{MIN}} + 1\right) \text{ or } \omega_{t_i} = \frac{1}{2}\left(\frac{\Delta s_i}{\Delta s_{MAX}} + 1\right)$$

depending on whether $\Delta s_i < \Delta s_{MIN}$ or $\Delta s_i > \Delta s_{MAX}$. Equation (8) is therefore solved using the modified values of ω. It should be noted that $\omega_t = 1.0$ in the regions of side-spacing control (see section 1.4.3) where Δs is permitted to be outside the specified limits.

1.3.2 Torsion terms. As shown in equation (7), the torsion terms, $\tau(s_i' - s_i)$ and $\psi(s_i^* - s_i)$ are added to the 1-D equation to preserve the required smoothness and orthogonality of the grid. They are constructed as being analogous to the behavior of torsion springs. Consider first the torsion term within the plane, $f(\tau)$. A torsion force, T, relates the node at $(i, j-1, k)$ to the node (i, j, k), is proportional to the turning angle, θ, shown in figure 3, and is defined as

$$T_i = \kappa\theta_{i,j-1,k} \tag{11}$$

where κ is a torsion-related constant. The next step is to relate this torsional force to the previously defined parameters on the adaption line. Since θ is generally small, we can approximate θ_i by $(s_i' - s_i)/|DA'|$ (see figure 3), where $s_i' - s_i$ is a function of the inclination angle computed from the proportioned orthogonality and straightness vectors, as described in section 1.3.2.3. In

addition, we assume that κ is proportional to the maximum ω_i and the local aspect ratio of the grid cell. Hence we can write

$$\kappa = \frac{\lambda \omega_{max}(s_{i+1,j-1} - s_{i-1,j-1})}{2|DA'|} \tag{12}$$

where λ is the proportionality coefficient and is an input quantity. The variable τ used in equation (8) can thus be defined as

$$\tau_i = \frac{\lambda \omega_{max}(s_{i+1,j-1} - s_{i-1,j-1})}{2|DA'|^2} \tag{13}$$

The evaluation of ψ is similar, with the node $(i, j-1, k)$ replaced by $(i, j, k-1)$ and a proportionality coefficient λ^* defined that is analagous to λ such that:

$$\psi_i = \frac{\lambda^* \omega_{max}(s_{i+1,j,k-1} - s_{i-1,j,k-1})}{2|BA'|^2} \tag{14}$$

where $|BA'|$ is the distance from $(i, j, k-1)$ to the location of $s^*_{i,j,k}$.

The evaluations of $|DA'|$ and $|BA'|$ are given in appendix II.

1.3.2.1 Evaluation of s' and s^*. These two variables, seen on the right-hand side of equation (8), are calculated from the torsion vectors and provide the direction of the smoothness and orthogonality constraints. The location of s'_i lies at the intersection of the projection of the within-plane torsion vector $\vec{t}_i$ with the arclength vector $\vec{s}_i$ along the current adaption line. The location of s^*_i is similarly determined from the between-plane torsion vector $\hat{t}^*_i$. Figure 4 shows the two unit torsion vectors: $\hat{t}$, acting from point D and intersecting the line AC at A', and $\hat{t}^*$, acting from point B and intersecting the same line at A*, such that

$$s'_i = s_i + AA' \text{ and } s^*_i = s_i + AA^* \tag{15}$$

In general, neither $\vec{t}$ nor $\vec{t}^*$ will directly intersect the line AC, and the projection of the torsion vectors onto $\vec{s}$ is required. An example of the projection vectors is shown in figure 5 where the projection of $\vec{t}$ onto the k plane intersects $\vec{s}$ at A'. A full description of the calculation of the intersection is given in appendix II.

1.3.2.2 Torsion vectors $\vec{t}$ and $\vec{t}^*$. Figure 6 shows in detail the within-plane torsion vector $\hat{t}$, and the associated base vectors ($\hat{n} = f(\hat{u}, \hat{b})$, and $\hat{e}$). The vectors in these figures are represented on a 2-D surface, and it should be remembered that the shown $\hat{t}$ is actually the projection of $\hat{t}$ seen in figure 4. The unit vector $\hat{t}$, associated with the nodal point (i, j, k) [but acting from the point $(i, j-1, k)$] is defined as

$$\hat{t}_{i,j,k} = \frac{1}{|t|}\left[C_t \hat{e}_i + (1 - C_t)\hat{n}_i\right] \tag{16}$$

where

C_t is the input parameter that defines the percentage of straightness to orthogonality

$\hat{e}_i$ is an average straightness vector from $(i, j-2, k)$ to $(i, j-1, k)$

$\hat{n}_i$ represents the orthogonality vector between the $j-1$ line and the node (i,j,k) and is a function of $\hat{b}$ and $\hat{u}$ vectors described below

In figures 5 and 6, all vectors are shown to intersect the j line in the interval $(i, i+1)$; however, this is not necessarily the case. In general, we assume that the projection of $\hat{t}_i$ intersects the j line in the interval $(l, l+1)$. Appendix II describes a general method used to compute the intersection of a vector with the vector $\vec{s}$.

The between-plane torsion vector $\vec{t}^*$ which acts from the node $(i,j,k-1)$, is similarly defined as

$$\hat{t}^*_{i,j,k} = \frac{1}{|t^*|}[C^*_t \hat{e}^*_i + (1 - C^*_t)\hat{n}^*_i] \tag{17}$$

where C^*_t is the input proportion parameter, and $\hat{e}^*$ and $\hat{n}^*$ are the equivalent straightness and normal vectors.

1.3.2.3 Evaluation of vectors $\hat{s}$, $\hat{e}$, and $\hat{n}$. The unit vector $\hat{s}_i$ (direction AC in figure 7) is the arclength vector from s_i to s_{i+1} along a j line. It is defined as

$$\hat{s}_{i,j} = s_{x_i}\hat{\imath} + s_{y_i}\hat{\jmath} + s_{z_i}\hat{k}$$

where

$$s_{x_i} = \frac{1}{|s_i|}(x_{i+1,j,k} - x_{i,j,k})$$

$$s_{y_i} = \frac{1}{|s_i|}(y_{i+1,j,k} - y_{i,j,k})$$

$$s_{z_i} = \frac{1}{|s_i|}(z_{i+1,j,k} - z_{i,j,k})$$

The unit vector $\hat{e}_i$ (direction ED in figure 7) is defined as the sum of three straightness vectors, $\hat{d}_{i-1}, \hat{d}_i$, and $\hat{d}_{i+1}$, where

$$\hat{d}_i = d_{x_i}\hat{\imath} + d_{y_i}\hat{\jmath} + d_{z_i}\hat{k}$$

and

$$d_{x_i} = \frac{1}{|d_i|}(x_{i,j-1,k} - x_{i,j-2,k})$$

$$d_{y_i} = \frac{1}{|d_i|}(y_{i,j-1,k} - y_{i,j-2,k})$$

$$d_{z_i} = \frac{1}{|d_i|}(z_{i,j-1,k} - z_{i,j-2,k})$$

Therefore,

$$\hat{e}_i = \frac{1}{\sqrt{3}}(\hat{d}_{i-1} + \hat{d}_i + \hat{d}_{i+1})$$

If the line $j-2$ does not exist (such as at the second line from a physical boundary), it is assumed that $\hat{e} = \hat{n}$.

The straightness vector between planes, $\hat{e}_i^*$, is shown as $\overrightarrow{PB}$ in figure 7. It is computed like $\hat{e}$ above, with points $(i, j-1, k)$ and $(i, j-2, k)$ replaced with $(i, j, k-1)$ and $(i, j, k-2)$ respectively.

The vector $\hat{n}$ is a combination of two vectors:

$$\hat{n} = \frac{1}{|n_i|}[t_n \hat{b} + (1 - t_n)\hat{u}] \tag{18}$$

where $\hat{u}_i$ represents orthogonality to the ik plane through the $j-1$ line, and $\hat{b}_i$ the orthogonality to the ik plane through the j line, as shown in figure 8. The parameter t_n proportions $\hat{b}$ and $\hat{u}$ and is defaulted to 0.5, changing only at the upper and lower marching boundaries, as discussed in the next section. The calculation of a normal vector to a point in a 3-D surface needs defining: for example, in figure 8, the vector $\hat{u}$ acts at D, perpendicular to the ik surface at $j-1$ (the surface that contains G,F,M, and Q). In this case, there are four possible planes to choose from (MDF,MDQ,GDF, and GDQ) and thus the definition of the $\hat{u}$ and $\hat{b}$ vectors is the average of the normals to each of the four planes, when the planes exist. Since two vectors define the plane in which they lie, the normal to that plane is obtained by taking their cross product. The vector $\hat{b}$ would also act at D but is perpendicular to the ik plane at j. Appendix III describes the general calculation of the normal to a given plane at a given point.

In addition to the within-plane orthogonal constraints, orthogonality is also controlled between planes by a function of the vector $\hat{n}^*$, defined as

$$\hat{n}_i^* = \frac{1}{|n^*|}(t_n^* \hat{b}^* + (1 - t_n^*)\hat{u}^*) \tag{19}$$

The normal $\hat{u}^*$ acts at point B, perpendicular to the ij plane at $k-1$, and $\hat{b}^*$ acts at A, but perpendicular to the plane at k.

1.4 Enforcement of Boundary Conditions

The ability to modify the adaption techniques in boundary regions substantially improves the flexibility of the adaptive scheme. The adaption domain is defined by the user and may be equal to, or a subset of, the physical grid. A boundary occurs at the limits of the adaption domain defined by the user. Within a plane, there are two types of boundaries: marching boundaries (all points along the initial and final adaption lines) and edge boundaries (at $i = i_{st}$ and $i = i_{end}$). There are also initial and final surface boundaries to take into consideration. Figure 9 shows a 2-D example of an adaption domain as a subset of the physical grid, and illustrates the two types of boundary lines when marching in the j direction. Note that if marching had been in the i direction, the boundary definitions would have been reversed.

By internally amending the previously defined variables of C_t, t_n, and λ in the boundary regions, it is possible to maintain physical characteristics at boundaries, allow for multiple passes, provide continuity across grid boundaries, and simplify multiple grid adaptions. Specialised boundary conditions, such as wall shape preservation and periodic boundaries, are discussed in separate sections.

1.4.1 Treatment of initial marching boundary line. If the initial adaption line is within the physical domain, a smooth transition will be required across the starting internal

boundary. For example, this situation occurs when adapting in zones; each zone has different flow features and the user may wish to march up to a certain line using one set of parameters, then continue marching using a new set. The common boundary between the two zones must remain unchanged when starting the second adaption pass. This feature is controlled by the input parameter m_g. When $m_g > 0$, a smooth transition from external grid lines (e.g., those in the already adapted zone) to internal lines is created by maintaining the same grid points along the initial line and then incrementally introducing the input adaption parameters. For example, when stepping to the second adaption line, we maintain as much straightness as possible by setting $C_t = 1$ (see eq. (16)). At each subsequent line, C_t is gradually decreased until it equals the user-supplied input value. The number of lines stepped before the full effect of the new parameters is felt depends on m_g; C_t will linearly decrease until m_g lines have been adapted. An example of this can be seen in figure 9, in which $j_{st} = 4$ and $m_g = 5$. The grid points along $j = 4$ will remain unchanged while those along $j = 8$ will be fully adapted to the input control parameters. The code controls the adaption parameters for lines in between. Consequently, we define a variable n_m as

$$n_m = \frac{m_g - j}{m_g - j_{st}} \tag{20}$$

and replace the value of C_t used in equation (16) by

$$C_{t_m} = C_t(1 - n_m) + n_m$$

At the same time, the value of λ is increased to $\lambda(1 + 5n_m)$ (thus increasing the magnitude of the torsion term) so that this amendment to C_t is effective. After m_g lines, $n_m = 0$, thus returning C_t and λ to their original values. Note that the computation of $\hat{e}$ along the $j_{st} + 1$ line is a function of the $j_{st} - 1$ line (if it exists), and this will also help in the merging process. A similar parameter, m_g^*, is used when the first plane must remain unchanged (e.g., at a multigrid boundary) and subsequent planes are gradually adapted.

1.4.2 Treatment of final boundary line. Another discontinuity will occur between the final adaption line and any subsequent lines external to the adaption domain. Since the adaption process is a marching scheme, it is not possible to use the same merging concept described earlier. On request (through the MARCH input parameter), the grid points on the remaining external lines will be redistributed with the same proportions as the points on the final adapted line. In addition, nonadapted planes can be proportioned with respect to lines on previously adapted planes. This is not an adaption to the flow field, but provides a more acceptable interface between the computational and physical domains.

1.4.3 Treatment of side-edge boundary. A smooth transition will also be needed at the side-edge boundaries if the adaption domain is internal to the given grid, i.e., if $i_{st} > 1$ or $i_{iend} < imax$. Figure 10(a) shows an example in which the fixed external grid spacing is denser than the first redistributed points, giving a discontinuous effect across the boundary. If the user requests continuity of mesh spacing across the side-edge boundaries (by setting the input parameter NEDGE, $(n_g) \neq 0$), a modification is made to the tension parameter, ω. Consider the start-edge boundary at $i = i_{st}$ along line j. We wish to enforce some value on ω_1 that will give a value of Δs_1 close to the value of $\Delta s_{i_{st-1}}$. To do this, we find the average $\omega\Delta s$ along the converged $j - 1$ line and replace ω_1 by $\overline{\omega\Delta s}/\Delta s_{i_{st-1},j}$. This value remains fixed during the iteration, but is

merged into the updated values of ω close to the boundary. In general, this implies that ω_2, ω_3, and ω_4 will be amended, however an additional option is available (using input parameters MG1 and MG2) to spread the effect further into the adaption domain. Figure 10(b) shows the effect of this process, giving a more appropriate spacing in the vicinity of the *ist* boundary. The end-edge boundary is handled in a similar manner.

The side-edge spacing often needs to be improved even when the adaption domain coincides with the physical domain. If an adaption pass generates inappropriate side-edge spacing, the code can be rerun with n_g set to nonzero in an attempt to improve the spacing by using the above technique. In this case we do not have an external Δs, and $\Delta s_{2,j}$ is used instead of $\Delta s_{i_{st-1},j}$. This will usually prevent the spring constants from pulling the lines too far off the boundary.

The variable n_g can be set to initiate computation for either or both side edges.

1.4.4 Treatment of orthogonality at boundaries. The code provides the choice of constructing grid lines that are as orthogonal as possible to a marching boundary, either to the final line in a plane or to the entire final plane. To accomplish this, the normal vector $\hat{n}$ (and/or $\hat{n}^*$) is emphasized over the straightness vector (by decreasing C_t) when the adaption line is close to a marching boundary line. For even greater control, the coefficient t_n is modified to emphasize either $\hat{u}$ or $\hat{b}$, depending on whether the initial-line or end-line boundary is being considered. To ensure that the modifications to these torsion coefficients sufficiently affect the computation, the value of λ is simultaneously increased to accentuate the torsion term. Since not all marching boundaries are physical boundaries, an input option is available that will override this emphasis on orthogonality for either boundary.

1.5 Preservation of Initial Wall Boundary Shape

In applications for which the shape of the wall boundary is defined by large geometric gradients, such as for turbine blades, sharp corners, and the leading edges of airfoils and wings, sufficient points need to be placed in the appropriate regions of the initial grid to accurately define the geometry. If flow-field gradients are weak in these regions, the standard grid-redistribution algorithm will cause points to be dispersed, leaving insufficient points to properly describe the boundary. To maintain necessary clustering in these regions, on user request a new variable is introduced that is a function of the geometry gradients that define the boundary shape. The weighting parameter now becomes a function of both flow-field gradients and geometry gradients. The solution procedure will therefore redistribute the points into regions of high geometry gradients as well as into regions of high flow-field gradients. Both the start and end wall boundaries are treated. A merging technique is used to integrate the boundary and internal redistribution, so that the internal flow is controlled only by the flow-field gradients.

The original weighting function was defined in equation (3) as

$$\omega = 1 + A\bar{f}^B$$

where $\bar{f}$ is a function of $\partial q/\partial s$. When the geometry option is invoked, $\bar{f}$ becomes a function of both $\partial q/\partial s$ and $\partial g/\partial s$, where $f(\partial g/\partial s)$ is computed as the radius of curvature at the wall boundaries. The radius of curvature is defined as

$$R = \left(\frac{\partial x}{\partial s}\frac{\partial^2 y}{\partial s^2} - \frac{\partial y}{\partial s}\frac{\partial^2 x}{\partial s^2}\right) \Big/ \left(\frac{\partial x}{\partial s}^2 + \frac{\partial y}{\partial s}^2\right)^2$$

where the derivatives are computed from the spline coefficients.

The full effect of the geometry function should be felt at the wall boundaries; it should not be a factor in the internal grid redistribution. Hence the final weighting function takes the form

$$\omega = 1 + A\bar{f}^B$$

where $\bar{f} = C_q f(q) + C_g f(g)$ and the constant C_q normally equals 1. The constant C_g equals 1 when the upper and lower wall boundaries are being adapted, but it must be decreased away from the walls until $C_g = 0$ internally. The value of the aspect ratio was chosen to drive the rate of decrease of C_g; i.e., $C_g = 1 - 4A_s$, where A_s is the aspect ratio at the maximum radius of curvature. Regardless of the value of A_s, C_g remains positive for a minimum of four steps from the boundaries and decreases smoothly. When the upper boundary is approached, C_g must be turned on when necessary and increased with each step, and $f(g)$ becomes a function of the upper wall boundary only.

It is also possible to adapt only to the geometry gradient. If this is requested, $C_q = 0$ and $C_g = 1$ throughout the flow, and $f(g)$ is computed for each adaption line, based on the local geometry of that line. This option has proven to be a useful tool for improving the starting grid.

1.6 Cyclical and Periodic Boundaries

Certain classes of grids used by flow-field solvers are not suitable for adaption using the basic grid formulation described thus far. The self-adaptive grid procedure is a marching scheme, i.e., the solution along each line is influenced only by previously adapted lines. The adaption of the first grid line is based on flow gradients only, but as marching proceeds, the redistribution of points is dampened by the torsion effect. As a result, the final adaption line will be somewhat less adapted to the flow-field gradient than the initial line. This implies that the scheme cannot be directly applied to cyclical and periodic grid structures, since common and/or matching boundaries will show as discontinuities. For 2-D problems, these difficulties have been overcome by including both an iterative and a rearrangement procedure that can be requested by the user. However, these options are not provided in this version of SAGE, and interested users should contact the authors to obtain a copy of SAGE2D and its documentation. For 3-D grid structures, maintaining periodicity at matching planes is a more difficult task. Since the initial grid and flow-field on matching first and last planes are identical, setting the plane torsion parameter (λ^*) to zero produces matching adapted grids on these planes. In this case, the adaption of intermediary planes has no influence on subsequent planes.

1.7 Discussion

1.7.1 Flexibility and segmentation. The vector approach used in the analysis provides for great flexibility in the use of the SAGE code. The user can choose the adaption direction and order of sequential adaptions without concern for the computational data structure. Multiple passes are available with no restraint on stepping directions: for each adaptive pass (or sweep), the user can choose a completely new set of adaptive parameters. This facility, combined with the capability of edge-boundary control, enables the code to handle multidimensional zonal grids without losing continuity along the common boundaries. For patched grids, the multiple-pass

capability allows complete adaption. It was mentioned in the introduction that the adaptive technique does not produce a unique grid. This nonuniqueness enables the user to choose the most appropriate solution to the grid adaption.

1.7.2 Multiple grids. The SAGE code currently accommodates files in single-grid format only. Multiple grids must therefore be separated before adaption. The plane merging capability enables the user to retain the grid distribution at a common (or zonal) boundary. However the user is responsible for passing the adapted plane from one grid to the matching plane in other grid(s). A new version of SAGE will soon be available that will read multiple-grid files. It will also permit the user to direct the transfer of common planes by using new options in the SAGE input-control parameter file. It is suggested that users interested in this capability contact the authors.

1.7.3 Blanked grids. Certain types of complex multiple-grid structures utilize the facility in PLOT3D that blanks out regions of overlapping grids. When this option is invoked, the number of points in each grid line need not be constant, i.e., IMAX can vary for each line. One version of SAGE has been modified to handle this situation, but in most applications it is not practical. The major drawback occurs when the number of points on a line decreases with each step. Depending on the flow-field gradients and the currently adapted line, the "dropped" points can produce a discontinuous grid structure. The starting location of the adaption can be chosen to ensure only an increasing point count, but this removes flexibility. This blanking feature is therefore not included in the standard version of SAGE since it detracts from the simplicity of the code. Nevertheless, there are cases for which the blanking option is useful, and it is suggested that the authors be contacted for further information.

1.8 Appendices

1.8.1 Appendix I: Derivation of A and B.

A and B provide self-adaptiveness to the adaption scheme. These two parameters ensure that all repositioned grid nodes maintain grid spacing to within user-requested mesh-size limits (Δs_{MIN} and Δs_{MAX}).

1.8.1.1 Calculation of A. We wish to relate A to the input values of Δs_{MIN} and Δs_{MAX}. A is constant throughout the entire mesh, and hence the 1-D relationship given in equation (2) holds. From the original definition of $\bar{f}$ in equation (9), $\bar{f}_{max} = 1$ and $\bar{f}_{min} = 0$; therefore, from equation (3) we have $\omega_{max} = 1 + A$ and $\omega_{min} = 1$. From equation (2) (rewritten as $\Delta s_i = K/\omega_i$) we can see that the minimum Δs will occur at K/ω_{max}. Similarly, the maximum Δs occurs at K/ω_{min}. We therefore wish to set $\Delta s_{MIN} = K/(1 + A)$ and $\Delta s_{MAX} = K$. These can be solved simultaneously: by eliminating K we get the expression given in equation (10)

$$A = \frac{\Delta s_{MAX}}{\Delta s_{MIN}} - 1$$

1.8.1.2 Calculation of B. This calculation is more complex. B is found by an iterative procedure and will change for each j line. The objective is to determine which value of B will give the minimum computed Δs_i equal to the requested minimum, Δs_{MIN}. For each value of B there exists a minimum Δs_i. (An example of a plot of B vs. Δs_{min} is shown in figure 11).

We need to find B at the requested Δs_{MIN}. To do this, we assume an initial value of $B(=1.0)$, evaluate ω, and then solve for the new Δs using equation (5). Although equation (5) is true only for the initial line, it is assumed here to hold for all j in order to simplify the B calculation. If $|\Delta s_{MIN} - \min \Delta s_i^{(n)}|$ is small, an acceptable value of B has been found. If not, a new B is computed from $B^{(n+1)} = B^{(n)} + \Delta B^{(n)}$, ω is reevaluated, and the procedure repeated.

$\Delta B^{(n)}$ can be found from the definition

$$\frac{\partial}{\partial B}\min(\Delta s_i) = \lim_{\partial B \to 0} \frac{(\Delta s_{MIN} - \min \Delta s_i^{(n)})}{\Delta B^{(n)}} \tag{21}$$

As mentioned in the calculation of A, we know that Δs_i is a minimum when $\omega_i = 1 + A$, and by substituting this in equation (5) and differentiating, we obtain

$$\frac{\partial}{\partial B}\min(\Delta s_i) = \frac{s_{max}}{1+A}\frac{\partial}{\partial B}\left(\frac{1}{\psi}\right) \tag{22}$$

where

$$\psi = \sum_{l=1}^{n_i} \frac{1}{\omega_l}$$

We know that

$$\frac{\partial}{\partial B}\left(\frac{1}{\psi}\right) = -\frac{1}{\psi^2}\frac{\partial \psi}{\partial B}$$

Hence, the next step is to evaluate $\partial\psi/\partial B$. With this definition of ψ, we can take the summation sign out of the differential and obtain

$$\begin{aligned}
\frac{\partial}{\partial B}\left(\sum_{l=1}^{n_i}\frac{1}{\omega_l}\right) &= \sum_{l=1}^{n_i}\frac{\partial}{\partial B}\left(\frac{1}{\omega_l}\right) \\
&= \sum_{l=1}^{n_i}\frac{\partial}{\partial B}\left(\frac{1}{1+A\bar{f}_l^B}\right) \\
&= -A\sum_{l=1}^{n_i}\frac{\bar{f}_l{}^B log \bar{f}_l}{\omega_l^2}
\end{aligned}$$

Equation (22) can now be solved, and after substituting for

$$\sum_{l=1}^{n_i}\left(\frac{1}{\omega_l}\right) = \frac{s_{max}}{\Delta s_{min}(1+A)}$$

from equation (5), we obtain

$$\frac{\partial}{\partial B}\min(\Delta s_i) = \frac{A(1+A)}{s_{max}}[\min(\Delta s_i)]^2 \sum_{l=1}^{n_i}\frac{\bar{f}_l^B log \bar{f}}{\omega_l^2}$$

Finally ΔB can be obtained from equation (21), giving B.

1.8.2 Appendix II: The intersection of a 3-D vector, $\vec{v}$, with the arclength vector, $\vec{s}$. This technique is used to obtain the $\hat{b}$ vectors used in the orthogonality terms and to find the location of s_i' and s_i^*. For example, to determine s_i' we need the segment in which the torsion vector $\hat{t}$ from the grid point $(i, j-1, k)$ intersects the j line. To evaluate the direction cosines of the $\hat{b}$ vector, we need to find the segment $(l \rightarrow l+1)$ along the j line that contains the intersection of a vector acting from $(i, j-1, k)$ that is normal to the j segment. For s_i^* and $\hat{b}^*$, we are concerned with the line j in the plane $k-1$. In all cases, the technique for finding l is the same.

Figure 5 shows the case of the torsion vector $\hat{t}$, computed for the point (i, j, k) and emanating from the point $(i, j-1, k)$ at D. This vector $(\hat{t})$ most likely will not lie in the plane described by ADC (or the equivalent l plane) and thus its projection onto the plane is required. The value of s' in the solution equation lies at A', the intersection of the projection of $\hat{t}$ onto the ADC plane and the arclength vector, $\vec{s}$. The vector $\hat{n}$ $(= n_x\hat{\imath} + n_y\hat{\jmath} + n_z\hat{k})$ is the unit normal to the plane ADC and is computed by the technique described in Appendix III. Both $|AA'|$ and $|DA'|$ are required in the code.

From vector addition, we can write

$$|AA'|\hat{s} = \overrightarrow{AD} + |DN|\hat{t} - |NA'|\hat{n} \tag{23}$$

Since the coordinates of A and D are known, $\overrightarrow{AD}(= a_x\hat{\imath} + a_y\hat{\jmath} + a_z\hat{k})$ can be evaluated, hence $|AA'|, |DN|$ and $|NA'|$ are the three unknowns. By equating coefficients of $\hat{\imath}, \hat{\jmath}$, and $\hat{k}$ in the vector equation (23), we can obtain a set of three equations with three unknowns:

$$|AA'|s_x = a_x + |DN|t_x - |NA'|n_x$$

$$|AA'|s_y = a_y + |DN|t_y - |NA'|n_y$$

$$|AA'|s_z = a_z + |DN|t_z - |NA'|n_z$$

These equations can be readily solved for $|AA'|$ by computing the relevant determinants.

$|DA'|$ is also found by utilizing vector addition. We have

$$\overrightarrow{DA'}(= |DA'|\hat{DA'}) = |AA'|\hat{s} - \overrightarrow{AD}$$

and, since the lengths of the left- and right-hand sides are equal,

$$|DA'|^2 = (|AA'|s_x - a_x)^2 + (|AA'|s_y - a_y)^2 + (|AA'|s_z - a_z)^2$$

1.8.3 Appendix III: Definition of normal vectors. The vector normal to a plane is required for the calculation of vectors $\hat{u}, \hat{u}^*, \hat{b}, \hat{b}^*$. This appendix describes the derivation of a general unit-vector normal, $\hat{n}$, at a given point (i, j, k) and normal to the plane ik passing through the constant $j-1$ line. This specific orientation is analogous to $\hat{u}$. Figure 8 shows such a vector acting at D, that represents the normal to the shaded surface. In the figure it can be seen that there are four possible planes passing through $(i, j-1, k)$: GDF, FDM, MDQ, and QDG. Any vector normal required in the code is calculated as the average of the normals to each of these

four planes. (When the node $(i, j-1, k)$ is on or close to a boundary, only one or two of these planes will exist.)

The normal to a plane is derived from the cross product of two vectors lying within the plane, with care taken to ensure the correct order of the operation (it is assumed that the input grid is organized as a right-handed system). In the example shown in figure 8, the four cross products are $\overrightarrow{DF} \times \overrightarrow{DM}$, $\overrightarrow{DF} \times \overrightarrow{GD}$, $\overrightarrow{QD} \times \overrightarrow{GD}$, and $\overrightarrow{QD} \times \overrightarrow{DM}$. The required unit normal is thus the unit vector representing the sum of these four vectors.

This picture gives an idealized view of the defined planes. In reality, some of the required lines have already been adapted and some have not. In figure 8, the point F could be significantly different from D, since D is an already-adapted point and F is still part of the initial grid. The code solves this problem by forming a block of data around the current j line that includes all i for lines $j-1 \Rightarrow j+1$ for planes $k-1 \Rightarrow k+1$. Lines that have not yet been adapted (e.g., $j+1$ at k or $j-1$ at $k+1$) are proportioned with respect to the s_i at j; this relocates the points for a smoother computational effect.

2. SAGE USER GUIDE

2.1 Overview

The SAGE code is based on the self-adaptive grid method developed by Nakahashi and Deiwert (1985). This guide contains information that will enable the user to run the code with little knowledge of the mathematical concepts employed in the development of the adaption technique. Included is a detailed description of the input-control parameters, along with routine descriptions and nomenclature. The code stands alone and has been run on many computer systems. Users wishing to understand and/or amend the code can find the details of the mathematical background in the first section of this document.

The first step in the code is to read two data files: one that contains the coordinates of the grid (x, y, z) and another that contains the corresponding flow-field variables, q. It is assumed that both these files are in the binary format associated with the plotting software package PLOT3D. The 3-D format is also assumed unless 2-D is specified by the user. The next step is to adapt the grid with respect to the user-supplied input-control parameters. Adaption takes place as a sequence of one-directional adaptions, with the input-control parameters determining the order of adaption. Each plane (i.e., 3-D surface) is adapted in one direction only before stepping to the next plane. Figure 1 shows a small section of a 3-D grid: adaption in the i direction (stepping in j) has already been performed on the first plane. The code then steps to the next plane and performs the same directional adaption on this plane. A 3-D adaption is performed by a sequence of adaptions: once the entire grid has been adapted in one direction, the adaption can be repeated with a new parameter-input file, in another direction. (It should be noted that different orders of the adaption direction will produce different adapted-grid solutions). When all adaptions are complete, the code sends the redistributed grid points and the corresponding interpolated flow variables to two new data files, also in PLOT3D format.

The analysis used to generate the algorithms used in the code is given in detail in the first section of this report. Briefly, the redistribution of points is controlled by parameters related to torsion and tension springs, and by the maximum and minimum allowable grid spacings. Along each grid line, (x, y, z) is transformed to a 1-D arc-length variable, s. If we assume that adaption (within a plane, k) steps in the j direction, the tension spring constants, ω, are evaluated at each point (i, j, k); i.e., $\omega_i = f(s_{i,j,k})$ and are a function of the gradient of the user-chosen flow-field variables. The torsion parameters (shown in figure 2) $\tau_i = f(s_{i,j,k}, s_{i,j-1,k}, s_{i,j-2,k})$ and $\psi_i = f(s_{i,j,k}, s_{i,j,k-1}, s_{i,j,k-2})$ are the link between the current line and the previously adapted lines. These are the parameters that maintain the integrity of the grid by controlling straightness and orthogonality within the plane and between planes. With the evaluation of these variables, a system of $(n_i - 2)$ equations (as given in equation (8)) is developed for the current j line and solved as a tridiagonal system. Once the adaption is complete for the current j line, the code steps to the next j line (either forward or backward) and repeats the same procedure. At the end of the current k plane, the code moves to the next plane and the process is repeated.

There are eight major steps taken in the code:

1. Input of three data files: initial grid, flow-field solution, and user-control parameters
2. Initialization and reorganization of data for computational purposes
3. Adaption along the start line on the start plane with the 1-D technique
4. For each subsequent j line on the initial plane;

a. Computation of variables that define torsion and tension constants (including ω and τ), and hence coefficients of the tridiagonal matrix defined by equation (8)
b. iteration to find new values of s, and hence (x, y, z)

5. Repetition of step 4 until all lines are complete on this plane
6. For each subsequent k plane, computation of the torsion parameter ψ for inclusion in the coefficients of the solution equation
7. Repetition of steps 4 through 6 until all planes have been adapted
8. Output of new grid and interpolated flow-field files in the original format

2.2 Execution of the 3-D SAGE Code

The following is an example of the command file to run the SAGE code on a UNIX system. SAGE is written in FORTRAN, and *sage* is the executable module.

```
cp xyz.grd fort.7      !copy grid file to unit 7
cp q.fun fort.8      ! copy solution file to unit 8
sage < sage.inp      ! run SAGE code with sage.inp containing user control parameters
cp fort.10 xyz.out      ! name the output adapted grid file
cp fort.11 q.out      ! name the interpolated function file
```

where *sage.inp* is in namelist format ($NAMEL). The remaining files are in PLOT3D format:

xyz.grd contains the initial grid points (stored as a right-handed coordinate system)
q.fun contains the flow-field variables to which the grid is to be adapted
xyz.out contains the adapted grid points
q.out contains the flow-field variables interpolated on the adapted grid

In addition, an output message file is associated with unit 6, which may or may not be defaulted to the user's output device. For other operating systems, the user must appropriately assign the six input/output files.

The following are the read statements for the PLOT3D binary input files:

xyz.grd:

```
READ(7) IMAX,JMAX,KMAX
READ(7) (((X(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
        (((Y(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
        (((Z(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX)
```

q.fun:

```
READ(8) IMAX,JMAX,KMAX
READ(8) FSMACH,ALP,RE,TIME
READ(8) ((((Q(I,J,K,N),I=1,IMAX),J=1,JMAX),K=1,KMAX),N=1,NDIM)
```

where

IMAX=number of points in the i direction of the grid file
JMAX=number of points in the j direction
KMAX=number of points in the k direction
NDIM=number of Q variables (default=5)

As seen in the above format, five flow-field variables are expected in the Q file. PLOT3D preassigns $\rho, \rho u, \rho v, \rho w$, and e, but since the SAGE code requests only the index of the function, any variables may be stored. Note that it is possible to handle any number of flow-field variables by changing the value of NDIM in the parameter statement at the beginning of each subroutine.

Also contained in the parameter statement is the maximum dimension of the grid arrays, currently set at 85. Because of the internal switching of data, this dimension parameter must be at least as large as the maximum of IMAX, JMAX, and KMAX. If the assigned code dimensions are too small, a message will be sent to the user stating the minimum dimension requirements for the current application.

FSMACH, ALP, RE, and TIME are not used in the code and may contain dummy values. They are part of the PLOT3D package and are displayed on the output plots.

2.3 User Input Parameters

sage.inp is the user-supplied, input-parameter control file. The grid adaption is based on the user's choice of these input parameters, which are listed and briefly described below. A more complete explanation of each parameter is given in the next section. *sage.inp* uses the namelist format, since generally only a few of the input parameters need to be changed from the default set in the code. These default values are shown in parentheses below and must be input by those using a nonstandard version of Fortran. If more than one adaption pass is to be made (for example, a two- or three-directional adaption), the subsequent adaptions can be made on the adapted grid by linking up to 10 sets of namelist inputs within the same *sage.inp* file.

Before describing the input parameters, some terminology needs to be clarified:

The term physical domain is used to reference the complete grid defined by the input grid file, i.e., the grid bounded by IMAX, JMAX, and KMAX. The adaption domain is the part of the grid, as defined by the input-control file, that is to be adapted, i.e., (IST,IEND), (JST,JEND), (KST,KEND). These two domains can be equivalent.

The direction i, j, or k refers to the direction of the grid coordinates as defined by the order in which they are stored in the grid file. The first index (normally containing x) is named the i direction, the second index is the j direction, and the third index the k direction. This implies that if data happened to be stored as (z, x, y) instead of (x, y, z), i would represent the z direction. Regardless of the order, the data must be stored as a right-handed coordinate system.

The adaption direction is used to define the 1-D line along which adaption (redistribution of points) takes place. In the analysis and in the descriptions below, this direction is always i for convenience, but the user may request i, j, or k. There are two stepping directions: one within the plane, defined as j in the analysis, and the other in the direction of plane marching, defined as k. Although the analysis and descriptions in this report assume a particular order of adaption and stepping, the code makes no such a priori assumptions, since the order is controlled by the user input-parameter file.

Reference is also made to grid boundaries. Side-edge boundaries refer to the start and end *points* at the edge of the adaption line (assumed to be i in the analysis). These are the edges that are controlled by the NEDGE parameter. Marching boundaries are the start and end *lines* of the stepping (within the plane) direction (assumed to be j). Adaption close to these lines are affected by the parameters ORTHS, ORTHE, and MGSTEPS, as well as by some internal controls. Planes juxtaposed to start and end *planes* are affected by ORTHS, ORTHE, and MGPLS.

2.3.1 Parameter control file, *sage.inp*

The input parameters, with their default values and short descriptions, are

$NAMEL		
IST	(1)	first adaption line in i direction
IEND	(IMAX)	last adaption line in i direction
JST	(1)	first adaption line in j direction
JEND	(JMAX)	last adaption line in j direction
KST	(1)	first adaption line in k direction
KEND	(KMAX)	last adaption line in k direction
IJPLANE	(TRUE)	the adaption surface lies in the (i, j) plane, with plane stepping occurring in the k direction
JKPLANE	(FALSE)	the adaption surface lies in the (j, k) plane, with plane stepping occurring in the i direction
IKPLANE	(FALSE)	the adaption surface lies in the (i, k) plane, with plane stepping occurring in the j direction
ISTEP	(FALSE)	=.true. for stepping in the i direction within the plane: it **cannot** =.true. if JKPLANE =.true.
JSTEP	(TRUE)	=.true. for stepping in the j direction within the plane
KSTEP	(FALSE)	=.true. for stepping in the k direction within the plane
INDQ	(1)	index of adaption flow-field variable, q; default (1) $\rightarrow \rho$
IQ(8)	(0)	enables a combination of q variables to drive the adaption
RDSMAX	(2.0)	relative maximum allowed Δs: $\Delta s_{MAX} \geq 1.0$
RDSMIN	(.5)	relative minimum allowed Δs: $\Delta s_{MIN} \leq 1.0$
CLAM(1)	(.01)	$=\lambda$, coefficient of torsion parameter τ, $10^{-1} \leq \lambda \leq 10^{-6}$
CLAM(2)	(.0001)	$=\lambda^*$, coefficient of plane torsion parameter, ψ; same range as λ
CT(1)	(.5)	proportion of "straightness" to "orthogonal" for torsion vector $\vec{t}$
CT(2)	(.5)	as CT(1), but proportions plane torsion vector, $\vec{t}^*$
NEDGE	(0)	override control on side-edge adaption: 1=both edges, 2=start edge, 3=end edge
MG1	(0 or 4)	used with NEDGE: number of points to merge start-side spacing
MG2	(0 or 4)	used with NEDGE: number of points to merge end-side spacing
INTER	(2)	order of interpolation: 2, 3, or 4
NFILT	(2)	number of passes to filter q and ω
MGSTEPS	(0)	number of adaption lines (from no adaption to full adaption) for merging of torsion control
MGPLS	(0)	number of planes before full adaption, using plane torsion control
MARCH	(FALSE)	=.true. to extrapolate end adaption *line* throughout remaining flow on this plane
MARCHPL	(FALSE)	=.true. to extrapolate last adapted *plane* throughout remaining planes
ADD	(0)	$= n$ to add n points between each node in selected range
SUB	(0)	$= n$ to delete n points between each node in selected range
LSTADD	(IST)	lower limit of range for adding points (i.e., ADD > 0)
LENDADD	(IEND)	upper limit (if ADD=0, value is ignored)

LSTSUB	(IST)	lower limit of range for point deletion (i.e., SUB > 0)
LENDSUB	(IEND)	upper limit (if SUB=0, value is ignored)
REMOVE	(0)	removes requested number of points from outer grid region
NOUP	(FALSE)	=.true. if no adaption required (e.g., for adding points only)
SAVE	(TRUE)	=.false. to suppress output of data files
ORTHS(2)	(TRUE)	=.false. to remove orthogonal constraint at start boundaries
ORTHE(2)	(TRUE)	=.false. to remove orthogonal constraint at end boundaries
GEOM	(FALSE)	=.true. to include geometry at wall boundaries in adaption variable
QFUN	(TRUE)	=.false. to remove $f(q)$ from adaption variable (used with GEOM=.true. only)
LNSING	(0)	$=n$ if this line is common to all adaption planes
PLSING	(0)	$=n$ if adaption plane n is collapsed to a line
TWOD	(FALSE)	=.true. if data sets are 2-D (see section 2.3.3)
$		

First adaption The best technique, even for experienced users, is to retain as many of the default parameters as possible. For some 3-D problems, it is clear which plane should be the stepping plane so the PLANE parameter, along with the STEP parameter, can be initially chosen. Note that only one PLANE parameter and one STEP parameter are needed for one adaption pass. Also, a flow-field variable should be chosen for INDQ that shows the flow features most clearly. To find appropriate within-plane parameters, adapt on only one plane and/or turn off the connectivity between planes (i.e., CLAM(2)=0) to see the set of 2-D adaption planes. If there are any lines common to all planes or if a plane is collapsed to a line, PLSING and/or LNSING must be set. If the first adaption pass is not acceptable, try increasing the ratio between RDSMAX and RDSMIN, decreasing CLAM(1), and/or setting NEDGE=1. Since many of the parameters have interdependent effects, it is better to change only one or two of the parameters at a time.

2.3.2 Explanation of user-supplied input parameters

The following is a detailed explanation of each of the input parameters; they are listed in alphabetical order.

ADD When this is set, points are added between adjacent mesh points within the requested range (see LSTADD, LENDADD). For example, if ADD=2, two points will be added between each consecutive grid point. Adding occurs only in the adaption direction and not in either of the stepping directions. Ensure that the added points do not cause the coded array dimensions to be exceeded.

CLAM CLAM(1)=λ and CLAM(2)=λ^* define the magnitude of the torsion parameters τ and ψ, respectively. As the values of these parameters decrease, more points will be pulled into the high-gradient regions, at the possible expense of grid smoothness.

CLAM(1) controls τ, the torsion parameter within the plane; its order of magnitude can lie between 10^{-6} and 10^{-1}. A value of zero produces a set of independently adapted lines, possibly generating crossed grid lines. As λ increases, the grid becomes smoother but less adapted.

CLAM(2) controls the magnitude of ψ, the torsion between planes. It has the same range of values as CLAM(1); however, if it is zero, the adapted grid may still be acceptable. For periodic planes (i.e., if the first and last planes are the same), setting CLAM(2)=0 is necessary to prevent a discontinuous grid at the juncture.

CT CT(1)=C_t and CT(2)=C_t^*; they represent the direction of the torsion vectors $\vec{t}$ and $\vec{t}^*$ (whereas λ is its magnitude in this direction). They have the values of $0 \leq C_t$ and $C_t^* \leq 1.0$, where a value of zero emphasizes orthogonality and a value of one emphasizes straightness. The default of .5 places the torsion vectors halfway between. This value is suitable in most cases, but it may cause problems when side boundaries are concave or when adapting on already adapted planes.

GEOM This parameter should be used when a wall boundary is defined by high surface gradients and the standard grid redistribution has moved points in such a way that the original shape has been deformed. When GEOM is set to .true., the code will add the surface curvature function to the flow-field gradient function in the wall boundary regions. Points will thus be maintained in or redistributed into regions of high surface curvature as well as into regions of high flow-field gradients. The contribution of the geometry function will proportionally decrease away from both boundaries, with the internal lines controlled by the flow-field only. If GEOM=.true. **and** QFUN=.false., adaption will be to geometry gradients only, for all grid lines.

IJPLANE, IKPLANE, JKPLANE These parameters define the plane on which adaption takes place. By default, they also imply the direction of the stepping plane. The input parameter file needs only one of these parameters to be set to .true., and the code will automatically assign .false. to the other two. IJPLANE=.true. indicates that the plane represented by (i, j) will be the adaption plane and that plane stepping will occur in the k direction. Whether k is a forward or backward step will depend on KST and KEND. Similarly, IKPLANE=.true. indicates that the adaption plane contains the (i, k) directions and that j is the plane-stepping direction. Finally, JKPLANE =.true. refers to the plane containing the points (j, k) and i is the plane stepping direction.

ISTEP, JSTEP, KSTEP These are used in conjunction with the PLANE parameters described above and define the marching and adaption directions within the defined plane. Only one of these three parameters should be input and set to .true., and the code will assign .false. to the other two. If IJPLANE=.true., then only ISTEP or JSTEP can be true (KSTEP must be false). If ISTEP=.true., stepping occurs in the i direction and thus the adaption direction will be j. Points will be adapted along each constant i line and stepping will occcur to the next i line, forward or backward, depending on IST and IEND. If JKPLANE=.true., only JSTEP or KSTEP can be true and similarly, for IKPLANE=.true., only ISTEP or KSTEP =.true. will have any meaning. The code puts out an error message if these inputs are inconsistent.

INDQ This parameter indicates which of eight possible flow-field variable(s) will drive the redistribution of grid points. From the standard PLOT3D format, five options are available:
1 → density, ρ
2 → x-momentum, ρu
3 → y-momentum, ρv
4 → z-momentum, ρw
5 → stagnation energy, e

Three more options are available by setting INDQ=6,7 or 8

6 → pressure, p

7 → Mach number, M

8 → temperature ratio, T

Pressure, Mach number, and temperature ratio are computed using the ideal gas relationship and assuming the Q file contains the standard variables. (The code actually assigns pressure to NDIM+1, Mach number to NDIM+2, and temperature to NDIM+3, so if the user has changed the value of NDIM to accommodate extra flow-field variables, INDQ must reflect this change.) Note that INDQ=4 is not available for 2-D data sets.

The user will normally choose to adapt to the flow-field variable that most represents the flow features. However, if different variables demonstrate different features, it may be advantageous to combine them to bring out all the features on the adapted grid. In this case, set INDQ=0 and input a value for IQ.

IQ is an array of eight (NDIM+3) integer values that are used only when INDQ=0. They allow the user to modify the adaption variable to a combination of variables. The order of IQ is consistent with the order of the flow-field variables (i.e., IQ(2) represents ρu and IQ(7) represents M). The value of an index is the proportion that the corresponding variable will contribute to the final adaption variable. For example, IQ(1)=1,IQ(7)=1 [i.e., (1,0,0,0,0,0,1,0)] will produce an adaptive function of $\frac{1}{2}(\frac{\partial \rho}{\partial s} + \frac{\partial M}{\partial s})$, and IQ(1)=1,IQ(3)=1, IQ(6)=3 will produce $\frac{1}{5}\frac{\partial \rho}{\partial s} + \frac{1}{5}\frac{\partial \rho v}{\partial s} + \frac{3}{5}\frac{\partial p}{\partial s}$. Obviously, (1,0,0,0,0,0,0,0) is the same as INDQ=1.

INTER indicates whether to use a linear (INTER=2), a quadratic Lagrange polynomial (INTER=3), or a cubic spline (INTER=4) scheme for interpolations. Interpolation is used throughout the code; for example, the q values in the output function file are interpolated at the new adapted grid points. Linear interpolation will usually provide the appropriate result.

IST, IEND contain the indices defining the first and last boundary lines of the adaptive domain in the i direction. Similarly, **JST, JEND** define the domain in the j direction and **KST, KEND** define the domain in the k direction. These variables define the limits of the adaption domain and must lie within the input grid boundaries defined for the physical domain, i.e., $1 \leq IST, IEND \leq IMAX$, $1 \leq JST, JEND \leq JMAX$ and $1 \leq KST, KEND \leq KMAX$.

Forward and backward stepping is also controlled by these parameters. If plane stepping is in the k direction, setting KST>KEND will produce backward stepping. Similarly, if stepping within the plane is in the j direction, setting JST>JEND will produce backward stepping. The reversing of the data is handled internally and is imperceptible to the user. Reversing either of the stepping directions will completely change the resulting grid, since the redistribution along the initial line and plane will be different, as will the connecting torsion springs. Reversing the order of the adaption direction (i.e., i in the default case) should have no effect on the solution, since the solution along a line is independent of the order of points. However, for meshes that contain very large and very small Δs_i, numerical accuracy may be influenced by the adaption direction.

LNSING This is a parameter that is unique to the 3-D grid adaption. In some grid types, planes emanate from a common grid line, generally on a wall surface. If the chosen set of adaption planes includes this common line, then this line should be adapted only on the first plane and not

on subsequent planes. This adapted line is then placed in the first line of all planes. Although the input option is LNSING=n, it is likely that $n = 1$.

LSTADD, LENDADD These are input only if ADD $\neq 0$ and if only a portion of the grid is to be expanded. If they are not input and ADD$\neq 0$, the entire adaption domain (not physical domain) is assumed. If ADD=0, their value is ignored.

LSTSUB, LENDSUB These parameters are input only if SUB$\neq 0$; they define the limits in which points are to be removed. If they are not input, then the entire adaption domain is assumed.

MARCH This parameter refers to stepping within the plane. If the last line to be adapted (j_{end}) is within the physical grid boundary (i.e., $j_{end} < j_{max}$), a sharp discontinuity will occur between the last adapted line, j_{end}, and the nonadapted line, j_{end+1}. Setting MARCH=.true. causes the remaining lines within the plane (i.e., $j_{end+1} \rightarrow j_{max}$) to be realigned so that they are proportional to the last adapted line. This realignment will be performed for every plane. The process is not an adaption but has proved to be a useful tool.

MARCHPL This parameter refers to the plane-stepping direction, and can be used independently or in conjunction with MARCH. If the last adaption plane is within the physical boundary (i.e., $k_{end} < k_{max}$), each line in each subsequent plane will be proportioned with respect to the adapted lines in the k_{end} plane.

MG1, MG2 When NEDGE is nonzero, an override mesh spacing is computed at the requested boundaries (either first, last, or both). This edge-point spacing, which is not a function of the adaption but of the initial grid, is merged into the three adjacent points, to produce a smooth transition. The default value of MG1 and MG2 is zero when NEDGE=0, but 4 when NEDGE is requested. The user has the option of overriding this value, setting it to any other integer value. This can be used when adapting a boundary layer; when MG is increased, the dense edge spacing is maintained over a larger region.

MGPLS This input variable is analogous to MGSTEPS described below, but applies to planes. If the first adaption plane (k_{st}) is internal to the physical domain, MGPLS= n permits the user to gradually bring in the effect of the adaption parameters to produce a smooth transition across the start plane. No adaption will take place on the first plane (k_{st}), and after n planes, full adaption occurs with the adaption parameters C_t^*, λ^*, etc. equaling their input value. This is an especially important feature for 3-D grids, since the initial distribution on the wall boundary frequently defines physical shapes and must not be changed. It is also useful for retaining matching multiple grid boundaries.

MGSTEPS (m_g) provides continuity when the first adaption line on each plane is internal to the physical boundary. Inputting MGSTEPS= n tells the code to start with no adaption on the initial line (i.e., retain the original distribution on the j_{st} line) and to linearly increase the adaption effect until, after n lines, full adaption occurs. At this point, C_t, λ, etc., will coincide with their input values. If MGSTEPS=1, no adaption will be performed on the first line, but full adaption will occur on the second line. Figure 9 shows an example with $m_g = 5$, and section 1.4.1 describes the parameter in detail.

NEDGE is a flag that requests an override on the computed side-edge boundary spacing. Side edges occur at i_{st} and i_{end} and frequently need special handling. If there are no flow gradients

near the edge of the domain, the standard adaption algorithm will pull points away from the edge. This may not be a satisfactory result, as, for example, when the side edge is internal to the physical grid boundary. Figure 10(a) shows a side-edge adaption with NEDGE=0. The flow-field gradients are concentrated in the center of the grid, and the first adaption point ($i = 4$) has been pulled far from the boundary line at $ist = 3$. It is clear that it is preferable for the adapted side-edge spacing to be continuous with the juxtaposed spacing in the external region. Even if the two boundaries coincide (i.e., $ist = 1$), the user may prefer a different spacing than that computed by the adaption algorithm. In either case, NEDGE can be set and the code will try to improve the side-edge spacing. The computed mesh-size override is merged into the next four points, but this number can be changed by MG1 and MG2. The result of setting NEDGE=1 is shown in figure 10(b). Depending on the case, both or only one of the side spacings may need improving. NEDGE=1 requests both edges, NEDGE=2 requests start edge only, and NEDGE=3 requests end edge only.

NFILT is a "filtering" variable that sets the number of passes used to smooth the gradient of the input q data and the computed tension parameter, ω. The default value of two will generally suffice, but if the flow-field variables are discontinuous, it may be helpful to increase the value of NFILT.

NOUP This variable is used to increase or decrease the number of points in the grid (by using ADD or SUB) without performing an adaption. NOUP stands for NOUPdate.

ORTHS, ORTHE The code assumes that orthogonality to the marching boundaries is desirable. This may not be the case, as, for example, in outgoing flow where the shape of the outer boundary is arbitrary. Setting ORTHS(1)=.false. will turn off orthogonality from the first to second adaption lines, and ORTHE(1)=.false. performs the same function when approaching the end adaption line. ORTHS(2) and ORTHE(2) will similarly affect the plane boundaries.

PLSING This is a parameter unique to 3-D grid adaption. It stands for plane singularity, and implies that a plane n is actually a single line, but is stored as a set of identical lines. The code will not be able to compute normals and will certainly "blow up" unless informed of this condition. It is only relevant when the adaption plane direction coincides with this collapsed plane. Do not use it when adapting in other plane directions.

QFUN This parameter permits the user to adapt to the geometry function only. When QFUN=.false. **and** GEOM=.true. are input, the coefficient of the flow-field gradient (C_q) is set to zero for all grid lines (see section 1.5). In this case, the geometry function is computed throughout the grid and drives the adaption for all grid lines.

RDSMAX, RDSMIN control the density of the redistributed points and are the maximum and minimum allowable grid spacings. They are input as proportioned values and are changed to physical variables internally, i.e., $\Delta s_{MAX} \times s_{max}/(n_i - 1)$ and $\Delta s_{MIN} \times s_{max}/(n_i - 1)$, where n_i is a constant equal to the total number of points along the adapted line, and s_{max} is the length of the current adaption line. This implies that RDSMAX $\geq$ 1.0 and RDSMIN $\leq$ 1.0. (If both are set to 1.0, a totally uniform grid will result.) As an example, RDSMIN=0.5 will prevent a converged Δs from being less than half the average step size. (Since many factors influence the distribution of grid points, this control is not absolute.) Note that since the adaption along the first line is not influenced by the torsion parameters, this initial line will present a clearer picture of the effect of RDSMAX and RDSMIN.

REMOVE It is possible that an initial grid has unnecessary grid points in the outer region. Once an initial solution has been obtained, the user can see that these points are wasted. REMOVE=n will remove n points from the end of the adaption line, before performing the adaption. Use NOUP is only removing points is required.

SAVE This is useful when more than one adaption pass is made in the same program run, for example, an adaption stepping in the j direction followed by one stepping in the i direction. The output grid and function files are large, and setting SAVE=.false. on a set of $NAMEL will suppress the output for that particular adaption. If SAVE=.true. (default), each subsequent output set of XYZ.OUT and Q.OUT files will be assigned to different unit numbers. As stated in the execution section, the first output set is assigned to units 10 and 11. The second output set will therefore be units 12 and 13, and so on.

SUB When SUB=n ($\neq 0$), points are removed from the adaption line. As an example, if $n = 1$, every other point is deleted. If n=2, two consecutive points are deleted between the points retained. Note that the number of stepping lines remains constant: points are only removed from the adaption line. To remove points from both directions, two passes are required. See LSTSUB, LENDSUB if only a selected range of deleted points is to be deleted.

TWOD If the input grid and function files are stored as 2-D PLOT3D files, this parameter must be set to .true. The code will assume that IJPLANE is the adaption plane and JSTEP the stepping direction. The user must specify ISTEP=.true. if required. The next section discusses the adaption of 2-D problems.

2.3.3 Two-dimensional Adaption.

The SAGE code can accommodate 2-D data sets, and will adapt the single plane in the same manner as in the original 2-D SAGE code (Davies and Venkatapathy 1989). When the input parameter TWOD is set to .true., the code will read the grid (assuming (x, y)) and function files as 2-D files. These data sets will then be internally converted to a 3-D format; the number of k planes will be set equal to one, creating a constant z coordinate, and the 4th q function will be shifted to index 5. (Note that indices INDQ and IQ retain their 3-D relationship) Because of this reorganization, no special handling of 2-D data sets is required within the body of the code. Data sets are reconverted to 2-D form before output.

Periodic and cyclical boundaries are not an option in this version of SAGE, and the current 2-D code must still be used for these applications. Another difficulty is size: 2-D data sets tend to have larger dimensions, and the maximum dimension available for 3-D datasets (currently 85) may be too small. The parameter statement at the beginning of each routine contains ID, JD and KD. Since 2-D datasets require only KD=1, ID and JD may be increased when KD is decreased. Note that this cannot be done for the 3-D case unless IJPLANE is the adaption plane; space must be available for data swapping.

2.4 Output Message File

The *sage.out* file contains messages that help to explain what has happened during program execution. At the end of each adaption, "ADAPTION n COMPLETE" indicates that the program was able to run to completion and that *xy.out* and *q.out* files have been created. The following are other messages that may be seen, along with a short description of their meaning.

NUMBER OF POINTS INCREASED FROM n_1 TO n_2. (Informational)
If the ADD option has been input, this message gives the new grid dimensions.

NUMBER OF POINTS DECREASED FROM n_1 TO n_2. (Informational)
If the SUB option has been input, this message gives the new grid dimensions.

ADD OPTION EXCEEDS DIMENSION. (Critical)
Self-explanatory. Increase array dimensions.

GRID SIZE TOO LARGE FOR DIMENSION STATEMENT
CHANGE PARAMETER STATEMENTS TO ID=n_1, JD=n_2, KD=n_3, IMX=n_4
THESE ARE THE MINIMUM DIMENSIONS FOR THIS SET OF CONTROL PARAMETERS (Critical)
Increase array dimensions to size suggested. Note that these values are appropriate for this set of input-control parameters only.

CAUTION: SUB OPTION PRODUCES TOO FEW POINTS FOR ADAPTION (Critical)
Fewer than 10 points remain in the adaption direction.

INCONSISTENT PLANE AND STEP (Critical)
A stepping direction has been requested that is not available for the requested plane. For example, IKPLANE and JSTEP.

NO CONVERGENCE ALONG INITIAL LINE, ERRMIN=a_1. (Warning)
The initial line is a 1-D adaption only. This may not be a catastrophic error, especially if a_1 is small; however, the adaption may not be completely satisfactory. The only control parameters that affect the initial line are RDSMAX, RDSMIN, and NEDGE.

NO CONVERGENCE ON LINE j AND PLANE k, ERR=a_2. (Warning)
This message is only a warning and adaption continues. Even many of these messages may of be no concern as long as a_2 is small. If adaption is successfully completed, check the new mesh to see if it is acceptable.

S IS NON-MONOTONIC ON LINE j AND PLANE k. (Critical)
This message will terminate the program. It indicates that the values of s_i at the completion of the iteration on line j are not monotonically increasing, thus implying crossover of points. Since this is unacceptable, the program outputs the data. It is then possible to see the plots and reevaluate the control parameters.

2.5 Outline of Each Subroutine

The *MAIN* routine is a driver routine whose task is to call the relevant subroutines. A loop (using NADS) is set to provide for multiple adaption passes. The first routine to be called is *INITIAL*, which reads and organizes the data. A loop is then set up for the adaption of each plane, and

the constant planar variables are computed. Finally, a loop is set up for each line in the plane in which all the coefficients of the adaption equation are computed, followed by the solution process. When each pass is complete, the *OUTPUT* routine is called.
Along with the *MAIN* routine, the SAGE code consists of the following subroutines, listed here in alphabetical order. Arguments shown in boldface are computed within the subroutine.

ADDPTS
This routine is called by *MAIN* if ADD$\neq$ 0. Extra points (depending on ADD) are inserted between every two input nodal points within the range LSTADD to LENDADD, by a linear interpolation.

ADDV(COSX1,COSY1,COSZ1,A1,COSX2,COSY2,COSZ2,A2,**COSX,COSY,COSZ**)
This is a utility routine. The direction cosines of two unit vectors (COSX1,COSY1,COSZ1) and (COSX2,COSY2,COSZ2) are input arguments. The routine computes the direction cosines (COSX,COSY,COSZ) of the unit vector that represents the sum of the input vectors, proportioned by the coefficients A1 and A2.

BLOCK(J,K)
Initially, all grid coordinates are stored in the X, Y, and Z arrays. In this routine, a block of data around the current j line is stored in arrays XJ, YJ, ZJ to prevent the original grid from being overwritten during interim calculations. Only the converged adaption line is replaced into the original grid arrays. The XJ, YJ, and ZJ are made up of all i points on the current j line and all i points on the $j-1$ and $j+1$ lines in the $k-1, k$, and $k+1$ planes, giving $(n_i, 3, 3)$ dimensioned arrays. At boundaries, nonexistent data points are filled with 999. Most calculations within the code, but especially the computation of the normal vectors, are performed on this block.

CROSSV(XT,YT,ZT,XT1,YT1,ZT1,DST,COSV,**AAP,DAP,ICROSS**,J)
Appendix II, section 1.8.2 contains the analysis used to develop this routine. COSV is the array containing the direction cosines of the vector $\vec{v}_i$, defined in that appendix. (XT,YT,ZT) are the coordinates of a j line, (XT1,YT1,ZT1) are the coordinates of a juxtaposed line, and DST is Δs along j. As an example, this routine is used to compute $s - s'$ (i.e., AA') and DA', the distance between s' and the corresponding node at $(i, j-1, k)$, as shown in figure 4. ICROSS is the array containing l and indicates the intersecting segment for each COSV.

CSPLIN(NT,S,V,**SPF**)
The cubic spline coefficients, SPF, are computed for NT points. S is the streamwise location of the function V. These coefficients are used by the routine *SPEVAL*.

DETERM(A1,B1,C1,A2,B2,C2,A3,B3,C3,**DET**)
This routine computes the determinant of the three vectors whose direction cosines are given in the argument list.

DLENG(JL,K)
When NEDGE is set, the tension parameter ω is amended at the edges (i.e., at IST and IEND) to improve edge-boundary spacing. This routine computes DLENGS and DLENGE, which are used in the edge ω calculation (by the routine WTEDGE). The values of DLENGS and DLENGE depend on whether the grid is defined outside of the adaption domain. JL indicates which j line is needed.

EDGEMG(VAR)
This is a utility routine. For various reasons, the values of some variables at the IST and/or IEND edges are overridden. To blend these different values into the calculations, this routine will merge the new values (given at the two boundaries of VAR) into the next three grid locations of VAR.

FBAR(J)
This routine is called once for every j line. The Δs and the gradients $\partial q/\partial s$ are computed at the input grid points and stored in FQ. If GEOM=.true., the wall gradients $\partial g/\partial s$ are also computed by calling *WALLS*, and stored in FG. In addition, the coefficient C_g is computed and FG is added to FQ and stored in F. The routine *INTF* is called to interpolate the value of F at these new grid points and to compute the normalized form of F, i.e., FB. The routine *GETB* is called to find the value of B for this j line. For lines other than the first, initial guesses for the s distribution are extrapolated from the converged s_i along the $j-1$ line. Since these do not correspond to the input points, new local values of x, y, and z are interpolated by calling *PROPS*.

FILTER(VAR,NIPTS,NFILT)
This routine smooths the parameter contained in VAR by adding a second derivative term, $v_i = .75v_i + .125(v_{i+1} - v_{i-1})$. NFILT is the number of smoothing passes (default=2). The parameters smoothed are $f_i = f(\frac{\partial q}{\partial s})$ and ω_i; they are returned to the calling routine via VAR.

GETB(J,K)
This routine computes the value of B used to evaluate ω. B is found by an iterative process and is said to converge when the minimum requested Δs equals the computed minimum Δs. The analysis for this routine is given in appendix I.

GETWT
This routine computes ω_t, the modifier of ω that is set when any computed Δs lies outside the requested range.

INITIAL(**NOMORE,NADS**)
This routine sets all the default values, reads the input parameter file and calls the grid and function file input routine. When necessary, the grid points and corresponding flow-field data are rearranged to correspond to the data organization assumed by the analysis. This routine also controls the addition or removal of points. If no more adaptions are requested, NOMORE is set and control is returned to the main routine.

INTF(F1,**F2**,S1,SMID,NPTS)
This routine interpolates to find F2 at the new s_i (S1), based on the input values of F1 and the midpoints (SMID) of the input s array. The interpolation routine chosen is based on the value of INTER.

INTXYZQ(J,K,J1,K1,SS,SN,**QJ**)
This routine interpolates for X, Y, Z, and Q at the new SN, given the corresponding values at SS. The new (x, y, z) coordinates are stored in the block of data defined by XJ, YJ, and ZJ. The Q data is stored in QJ. The appropriate interpolation routine is called, its choice based on INTER.

LAGCOF(SNEW,SARR,NPTS,**M,P1,P2,P3**)
This routine computes the Lagrange coefficients P1, P2, and P3 for a point SNEW, with respect to the input s array, SARR. These are used by the calling routine to interpolate for the variable

at SNEW. First or second order is available; the choice depends on INTER. M is the associated index for P1 and will reflect a forward or backward interpolation, depending on the location of SNEW within the interval.

LINE1
This routine solves for the adapted values of s_i along the initial line $j = j_{st}$ on the initial plane k_{st}. Since both torsion terms are zero on this line, the Δs_i are computed from the 1-D approach.

MARCHJ(K)
If the final adapted line within plane k is internal to the physical end-boundary line, this routine will redistribute the points on the remaining j lines, based on the distribution along the j_{end} line. The routine is called for each plane. This action is performed only on request by the user and is not an adaption to the flow field. However, it will prevent the discontinuity between the last adapted line and the remaining nonadapted grid lines.

MARCHK
This routine is called on user request if the final adaption plane k_{end} is internal to the physical end plane k_{max}. It will redistribute points on all lines on the remaining planes to be proportional to the corresponding line on the last adapted plane.

MGWALLS(J)
This routine is called when GEOM=.true. It computes the coefficient of the geometry function, C_g (FGW), based on the local aspect ratio.

NOADAPT(J,K)
This routine updates variables and/or places them in appropriate arrays when no adaption is to be performed on the current j line, but adaption is to be performed on the next line, where the code expects certain variables to be available at $j - 1$. This occurs, for example, in merging situations (MGSTEPS> 0) and for common lines (LNSING> 0).

NORM(F1,NPTS)
The function F1 is normalized as $(F1 - F1_{min})/(F1_{max} - F1_{min})$. If maximum and minimum values are equal or nearly equal, the normalized variable is set to $O(10^{-5})$

NORMPT(IP,JP,KP,INDPL,**PA,PB,PC,SING**)
This routine finds the vector at the point (IP,JP,KP) normal to the plane defined by INDPL. Although three direction planes exist through the point, only two are needed by the calling routines: INDPL=1 is the (i, k) plane and INDPL=2 is the (i, j) plane. The analysis to describe this routine is given in Appendix III of this report. Four normals are computed and the average is found, with the direction cosines returned in (PA,PB,PC). SING is set to 1 if the normal does not exist.

OUTPUT(NADS)
OUTPUT is called at the conclusion of each adaption set. Proportioning of any remaining planes is performed if requested by MARCHPL, and then the data files are returned to their original order, to conform with the input mesh structure. Output unit numbers are chosen and WRITOUT is called to output the grid and flow-field files. NADS indicates which adaption number has been completed (maximum of 10 sets). This routine is also called if s becomes nonmonotonic (OK=.false.). In this case, the new mesh points that have been computed are output to help the user choose more appropriate control parameters.

PROPS(J,K)
After a new solution of s_i has been obtained on a line j, the code stores this data in line $j-1$ and steps to the next line. This routine proportions the new s_i throughout the non-adapted regions of the block of data defined by XJ,YJ,ZJ. Essentially, this provides a first guess for the current j line and also produces smoother planes for the vector normal calculations. (See appendix III)

PURPLE(A1,A2,A3,B1,B2,B3,**V1,V2,V3**,NFL)
A and B are direction cosines of two vectors defining an enclosed plane. This routine takes their cross-product and normalizes the result to give the direction cosines (V1,V2,V3) of the unit normal to the enclosed plane. NFL is the direction sign of the normal.

READAT
This routine reads in the grid and function files in PLOT3D binary format. Input datasets may be in 2-D or 3-D form.

SETUPJ(J,K)
SETUPJ computes the direction cosines of the vectors $\vec{u}, \vec{b}$ and $\vec{e}$ used to evaluate the torsion vector $\vec{t}$. These vectors are associated with the (i,j) points within the constant k plane.

SETUPK(J,K)
This routine is similar to SETUPJ, but the direction cosines of $\vec{u}^*, \vec{b}^*$ and $\vec{e}^*$ are computed, to evaluate the plane torsion vector, $\vec{t}^*$. These vectors are associated with the (i,k) within the constant j plane.

SINGPLN
When PLSING is set, *SINGPLN* stores the result of the first adapted line into every line of the same plane.

SIZING(IER)
The parameter statement at the beginning of each subroutine presets the dimensions (ID,JD,KD) of the grid and q files. This routine compares the input grid dimensions (IMAX,JMAX,KMAX) to these preset values. If insufficient space has been allocated, the minimum possible values of ID, JD, and KD are computed for this adaption to proceed. (These values are not obvious since space must be allocated for data swapping.) IMX, the maximum of ID, JD and KD is also evaluated. Finally, IER is set to 1. This informs the *INITIAL* routine to terminate the code and to send a message to the user to recompile SAGE with the suggested dimensions.

SOLUT(J,K)
By the time *SOLUT* is called, all variables have been computed that are needed to obtain the new distribution of s_i. The coefficients of s_i (see eq. (8) in the first section of this report) are set up in a tri-diagonal matrix and solved for s_i. Interpolated values of ω_i are found at these new values of s_i and iterations are performed until $\sum|(s_i^{(n)} - s_i^{(n-1)}|$ is small or too many iterations have been performed. In both cases, a check is made to see if all the s_i are monotonic. If so, the program continues; if not, the flag OK is set to false, causing the program to output the current grid and terminate.

SPEVAL(NT,S,V,SPF,SI,**VI,VPI,VPPI**)
This is a cubic spline interpolation routine. It uses the coefficients (SPF) computed in *CSPLIN*. In addition to interpolating for the variable V at SI, the corresponding first (VPI) and second (VPPI) derivatives are also evaluated.

SUBPTS
When SUB $\neq$ 0, n points are deleted between each retained mesh point. If LSTSUB and/or LENDSUB are nonzero, the range of the deletion is restricted. Deletion occurs only in the adaption direction and does not decrease the number of stepping lines or planes.

SWAPINV
This routine is called if $k_{st} > k_{end}$, $j_{st} > j_{end}$ or $i_{st} > i_{end}$. The order of (i, j, k) in the x, y and z input matrices are reorganized to ensure that internal computations have monotonically increasing indices. The flag for the handedness of the coordinate system is amended accordingly.

SWAPXYZ(RSWAP)
Since the internal computation assumes that j is the stepping direction within the plane and that k is the stepping direction of planes, this routine is called to interchange x, y, z, and q when the input requests alternative stepping directions. RSWAP is a flag that states whether this data exchange is at the start of the computations or is the reverse process required for output.

SWAP2D(IO)
When TWOD=.true., the input data sets are formatted in two dimensions. This 2-D plane is reorganized to appear in the code as a 3-D surface. Every z is given the value of zero, each Q(4) is moved to Q(5), and Q(4) is zeroed. IO indicates whether the translation is from 2-D to 3-D, which occurs on input, or from 3-D back to 2-D for output.

TORCOF(L,JK,JKST,JKEND,MGNOS,MARCHJK)
TORCOF is called twice, once with all the arguments representing the k plane passing through (i, j, k) and once with the arguments representing the j plane also passing through (i, j, k). This routine amends the coefficients (C_t, λ) etc. of the torsion vectors $\hat{t}$ and $\hat{t}^*$ based on the current line location. For example, C_t is decreased when leaving or approaching a boundary to emphasize orthogonality.

TORSION(J,K)
This routine first chooses the appropriate $\hat{b}$ and adds to $\hat{u}$ to obtain $\hat{n}$. The torsion vector $\hat{t}$ is then obtained by adding $\hat{n}$ and $\hat{e}$. $\hat{t}^*$ is computed in a similar manner. The routine *CROSSV* is then called to find the intersection of the torsion vectors with the j line from which both s'_i and s^*_i can now be evaluated. Finally, a check is made to ensure that s' and s^*_i monotonically increase. If they do not, the code attempts a correction, but any major problems will cause the code to terminate in the *SOLUT* routine.

UNITV(X1,Y1,Z1,X2,Y2,Z2,**DIRCX,DIRCY,DIRCZ**)
This is a utility routine that finds the unit vector from (X1,Y1,Z1) to (X2,Y2,Z2). DIRCX, DIRCY and DIRCZ are the direction cosines of this vector.

UPDATE(J,K)
UPDATE is the last routine called in the iteration loop for the current j. Newly adapted values of s_i have been found. The values of x, y, z and q at this new distribution are interpolated and replaced into the matrices containing the physical mesh.

VMERGE(DIRV,LST,LEND)
This routine performs the same function as *EDGEMG*, but with a vector quantity (DIRV) in place of a scalar value. LST and LEND indicate which value of DIRV must be merged into the next three points.

WALLS(JW,K)
Along line j (JW), the geometry gradient, as defined by the radius of curvature, is computed for each grid segment. Normally, j will equal j_{st} or j_{end}. However, for cases when geometry is the only adaption variable (QFUN=.false.), *WALLS* is called for every j line.

WRITOUT
This routine writes the adapted grid and interpolated function file on units NITG and NITQ, in PLOT3D binary format.

WTEDGE(J,K)
This routine is called by *MAIN* when NEDGE modification is requested. The edge values of the tension parameter at the next j line are a function of the average $\omega\Delta s$ along the just completed adapted line. This routine calls *DLENG* to obtain the appropriate value of edge Δs on the next line, computes the average $\omega\Delta s$ on this line, and evaluates WDS and WDE to be used in routines *LINE1* and *SOLUT*.

2.6 Nomenclature

The following is a list of variables used in the formulation of the SAGE method. When applicable, the corresponding FORTRAN name used in the code is shown in boldface.

A, B	constants used to compute ω, **(A,B)**
A_s	aspect ratio, used to control C_g, **FGASP**
C_g	coefficient of $f(g)$ in ω calculation, **FGW**
C_q	coefficient of $f(q)$ in ω calculation, **FQW**
C_t	input proportion coefficient for torsion, **CT(1)**
C_t^*	input proportion for torsion between planes, **CT(2)**
$C_{t_m}, C_{t_m}^*$	modified values of C_t, C_t^*, **CTM(1),CTM(2)**
$\vec{b}$; $(b_{x_l}, b_{y_l}, b_{z_l})$	normal vector to j line within k plane; direction cosines of $\vec{b}$, **COSB**
$\vec{b}^*$	normal vector to j line within j plane, **COSBK**
$\vec{d}$; $(d_{x_i}, d_{y_i}, d_{z_i})$	straightness vector, **COSD**
$\vec{e}$; $(e_{x_i}, e_{y_i}, e_{z_i})$	average straightness vector within k plane, **COSE**
$\vec{e}^*$	average straightness vector within j plane, from $k-2 \rightarrow k-1$, **COSEK**
f	gradient of q (and g if necessary), **F**
f_{min}, f_{max}	minimum and maximum f used to normalize f, **FMIN,FMAX**
$\bar{f}$	normalized function of f, **FB**
g	geometry function
i	subscript indicating the current node in adaption direction, **I**
$imax, jmax,$ $kmax$	total number of points in i, j and k directions of input grid file, **IMAX, JMAX, KMAX**
i_{st}, j_{st}, k_{ist}	indices indicating start of adaption domain in i, j and k directions, **IST, JST, KST**
$i_{end}, j_{end}, k_{end}$	indices indicating end of adaption domain, **IEND, JEND, KEND**
j	subscript of the current stepping line, **J**
k	subscript of current adaption plane, **K**
κ	torsion-related constant

l	local subscript relating to node i, **L**
m_g	input value indicating number of stepping lines before full adaption, **MGSTEPS**
m_g^*	the plane equivalent of m_g, **MGPLS**
n_g	flag for edge control, **NEDGE**
n_i	number of points in the adaption line, **NIPTS**
n_m	value of j at full adaption line $f(m_g)$, **CNM(1)**
n_m^*	value of k at full adaption plane, **CNM(2)**
$\vec{n}$; $(n_{x_i}, n_{y_i}, n_{z_i})$	orthogonality vector within plane, **COSN**
$\vec{n}^*$	orthogonality vector between planes, also stored in **COSN**
q	input flow-field variable $(\rho, \rho u, \rho v, \rho w, e)$, **Q**
R	radius of curvature for geometry function, **FGS**
s	streamwise length, **SS or SN**
$\vec{s}$; $(s_{x_i}, s_{y_i}, s_{z_i})$	vector representing s, **SSX, SSY, SSZ**
s_{max}	maximum value of s on line j, **SMAX**
s'	value of streamwise length used for torsion within planes, **SP**
s^*	value of streamwise length used for torsion between planes, **SPP**
Δs_i	$s_i - s_{i-1}$, **DS**
$\Delta s_{MIN}, \Delta s_{MAX}$	requested minimum and maximum grid spacings, **DSMIN,DSMAX**
$\Delta s_{min}, \Delta s_{max}$	computed minimum and maximum grid spacings
T	torsion force
$\vec{t}$; $(t_{x_i}, t_{y_i}, t_{z_i})$	within-plane torsion vector, **COST**
$\vec{t}^*$	torsion vector betwen planes, also **COST**
t_n	proportion of $\vec{b}$ and $\vec{u}$ used to compute $\vec{n}$, **TN(1)**
t_n^*	proportion of $\vec{b}^*$ and $\vec{u}^*$ used to compute $\vec{n}^*$, **TN(2)**
$\vec{u}$; $(u_{x_i}, u_{y_i}, u_{z_i})$	vector normal to $j-1$ line in k plane, **COSU**
$\vec{u}^*$	vector normal to j line on $k-1$ plane, **COSUK**
x, y, z	input grid mesh, **(X,Y,Z)** globally and **(XJ,YJ,ZJ)** locally
θ	angle for torsion computation
λ	input magnitude of torsion control parameter, **CLAM(1)**
λ^*	input magnitude of plane torsion control parameter, **CLAM(2)**
ω	tension spring force, **WEIGHT**
ω_t	computed weighting on ω, **WT**
ψ	between-plane torsion-related parameter, **TAUPL**
τ	within-plane torsion-related parameter, **TAU**

2.7 List of Major Variables

This section contains the list of variables (in alphabetical order) used in the SAGE code. Local variables that contain only intermediary values are not listed. The format is:
Variable name(dimension) $/r_1/r_2/$ brief description
where r_1 describes what type of variable—input, local, parameter, or name of common block and r_2 lists routine(s) where the variable is initialized.

A	/COM2/INITIAL/ A used to compute ω
AA(IMX)	/local/SOLUT/ coefficient of s_{i-1} in solution matrix
AAP(IMX)	/argument/CROSSV/ $s - s'_i$ or $s - s^*_i$
ACT	/local/TORSION/ final modified C_t
ADD	/COM9/input/ if set, add grid points
ALPHA	/COM12/input/ information only for PLOT3D
AMACH(IMX)	/local/FBAR/ computed Mach number
B	/COM2/GETB/ B used to compute ω
BB(IMX)	/local/SOLUT/ coefficient of s_i in solution matrix
BCONV	/local/GETB/ convergence criteria for B iteration
BJ1	/local/GETB/ value of B along $j-1$ line in the k plane
BK1	/COM15/GETB/ value of B along j line in the $k-1$ plane
CC(IMX)	/local/SOLUT/ coefficient of s_{i+1} in solution matrix
CLAM(2)	/COM10/input/ λ and λ^*, magnitude of torsion terms
CLAMW(2)	/COM10/TORCOF/ modified CLAM
CNM(2)	/COM10/TORCOF/ λ modifiers for MGSTEPS$\neq$ 0 and MGPLS$\neq$ 0
CONV	/COM15/INITIAL/ general convergence criteria
COSB(IMX,3)	/COM7/TORSION/ direction cosines of $\vec{b}$, in (i,j) plane
COSBK(IMX,3)	/COM20/TORSION/ direction cosines of $\vec{b}^*$, in (i,k) plane
COSD(IMX,3)	/local/SETUPJ,SETUPK/ temporary straightness vector, $\vec{d}$
COSE(IMX,3)	/COM7/SETUPJ/ straightness vector, $\vec{e}$, in (i,j) plane
COSEK(IMX,3)	/COM20/SETUPK/ straightness vector, $\vec{e}^*$, in (i,k) plane
COST(IMX,3)	/local/TORSION/ torsion vector, $\vec{t}$ or $\vec{t}^*$
COSU(IMX,3)	/COM7/SETUPJ/ $\vec{u}$, normal to $j-1$ line, in (i,j) plane
COSUK(IMX,3)	/COM20/SETUPK/ $\vec{u}^*$, normal to j line in $k-1$ plane
CT(2)	/COM10/input/ C_t and C^*_t, directions for torsion vectors
CTM(2)	/COM10/TORCOF/ modified C_t and C^*_t
DAP(IMX)	/COM9/CROSSV/ DA' for s' calculation
DAPPL(IMX)	/COM9/CROSSV/ DA' for s^* calculation
DET	/argument/DETERM/ value of three-order determinant
DLENGE	/COM6/DLENG/ Δs computed for end-edge modification
DLENGS	/COM6/DLENG/ Δs computed for start-edge modification
DMINSDB	/local/GETB/ $\partial\ min(\Delta s)/\partial B$
DS(IMX)	/COM3/FBAR,LINE1,SOLUT/ $s_i - s_{i-1}$
DSMAX	/COM6/FBAR/ Δs_{max} from RDSMAX
DSMIN	/COM6/FBAR/ Δs_{min} from RDSMAX
F(IMX)	/COM2/FBAR/ flow gradient, $f = \partial q/\partial s$, at input s
FB(IMX)	/COM2/INTF/ $\bar{f}$, normalized f at current s
FG(IMX)	/COM17/FBAR/ normalized $\partial g/\partial s$
FGASP	/local/MGWALLS/ function of aspect ratio
FGS(IMX)	/COM17/WALLS/ $\partial g/\partial s$ at SMSS

FGW	/COM17/MGWALLS/ coefficient of FG for ω calculation
FF(IMX)	/local/SOLUT/ coefficient of RHS of solution matrix
FQ(IMX)	/COM17/FBAR/ normalized $\partial q/\partial s$
FQW	/COM17/MGWALLS/ coefficient of FQ for ω calculation
FSMACH	/COM12/input/ information only, for PLOT3D
GEOM	/COM11/input/ request for geometry function
ICROSS	/argument/CROSSV/ index for interval location of s' or s^*
ID	/dimension/parameter statement/ currently set at 85
IEND	/COM4/input/ last node along i in adaption domain
IINVERSE	/COM13/INITIAL/ adaption requested in backward i steps
IJPLANE	/COM14/input/ true if (i, j) is adaption plane
IKPLANE	/COM14/input/ true if (i, k) is adaption plane
IMAX	/COM4/input/ number of points in physical domain, i direction
IMX	/dimension/parameter statement/ currently set at 85
INDQ	/COM5/input/ index for q for adaption variable
INTER	/COM11/input/ order of interpolation, 2,3 or 4
IQ(8)	/COM5/input/ related to INDQ, combines q adaption function
IST	/COM4/input/ first node along i in adaption domain
ISTEP	/COM13/input/ true for stepping in i direction within the plane
JD	/dimension/parameter statement/ currently set at 85
JEND	/COM4/input/ last node in j adaption domain
JINVERSE	/COM13/INITIAL/ adaption requested in backward j steps
JKPLANE	/COM14/input/ true if (j, k) is adaption plane
JMAX	/COM4/input/ number of points in physical domain, j direction
JST	/COM4/input/ first node in j adaption domain
JSTEP	/COM13/input/ true for stepping in j direction within the plane
KD	/dimension/parameter statement/ currently set at 85
KEND	/COM4/input/ last node in k adaption domain
KINVERSE	/COM13/ adaption requested in backward k steps
KMAX	/COM4/input/ number of points in physical domain, k direction
KST	/COM4/input/ first node in k adaption domain
KSTEP	/COM13/ true for stepping in k direction within the plane
LENDADD	/COM19/input/ start of add points range
LENDSUB	/COM18/input/ start of delete points range
LNSING	/COM15/input/ flag to indicate a common line for each plane
LSTADD	/COM19/input/ end of add points range
LSTSUB	/COM18/input/ end of delete points range
MARCH	/COM14/input/ true to interpolate up to physical boundary
MARCHPL	/COM14/input/ true to interpolate to final plane

MAXITS	/COM15/INITIAL/ maximum number of iterations for convergence
MG1	/COM9/INITIAL/ number of merging points for NEDGE at i_{st}
MG2	/COM9/INITIAL/ number of merging points for NEDGE at i_{end}
MGPLS	/COM11/input/ number of planes before full adaption of plane parameters
MGSTEPS	/COM11/input/ number of lines before full adaption within plane
MXFG	/COM17/MGWALLS/ location of maximum $\partial g/\partial s$
NADS	/argument/MAIN/ index on number of adaptions
NDIM	/dimension/parameter statement/ used for input q, currently set to 5
NEDGE	/COM9/input/ initiates side-edge boundary override
NFILT	/COM11/input/ number of passes for smoothing data
NFLAG	/COM11/INITIAL/ indicates direction sign for normal vectors
NITG	/COM12/OUTPUT/ unit number for output grid file
NIPTS	/COM4/INITIAL/ total number of i points in computation domain
NITQ	/COM12/OUTPUT/ unit number for output q file
NOUP	/COM15/input/ prevents adaption. i.e., no update
NOMORE	/argument/INITIAL/ no more input data sets remain; end program
OK	/COM14/SOLUT/ indicates normal termination
ORTHE	/COM16/input/ false removes orthogonality at outer boundary
ORTHS	/COM16/input/ false removes orthogonality at initial boundary
P1,P2,P3	/argument/LAGCOF/ coefficients of Lagrange polynomials
PLSING	/COM15/input/ flag to define a singular plane
PRES(IMX)	/local/FBAR/ computed pressure
Q(ID,JD,KD,NDIM)	/COM1/input,UPDATE/ flow-field variables
QFUN	/COM11/input/ false to allow geometry function only
RDSMAX	/COM5/input/ relative value of Δs_{max}
RDSMIN	/COM5/input/ relative value of Δs_{min}
RE	/COM12/input/ not used, PLOT3D variable
REMOVE	/local/input/ number of points to remove at outer boundary
RSWAP	/argument/INITIAL,OUTPUT/ flag for swapping data
SAVE	/COM14/input/ flag to suppress output
SING	/argument/NORMPT/ flag to indicate non-existant normal
SMS(IMX)	/COM3/FBAR/ midpoints of Δs
SMSS(IMX)	/COM17/WALLS/ s_i for geometry calculation
SN(IMX)	/COM3/FBAR/ current s array along j line
SNM(IMX)	/COM3/UPDATE/ converged s array along $j-1$ line
SNMK(IMX)	/COM3/FBAR/ converged s along j line on $k-1$ plane
SP(IMX)	/COM9/TORSION/ s', intersection of torsion vector and j line
SPPL(IMX)	/COM9/TORSION/ s^*, intersection of plane torsion vector and j line

SPF(IMX)	/argument/CSPLIN/ cubic spline coefficients
SS(IMX)	/COM3/FBAR/ initial value of s along j line
SUB	/COM18/input/ request to delete points
TAU	/local/SOLUT/ τ, torsion parameter within plane
TAUPL	/local/SOLUT/ ψ, torsion parameter between planes
TIME	/COM12/input/ not used, PLOT3D variable
TN(2)	/COM10/INITIAL/ proportion of $\vec{b}$ to $\vec{u}$ for torsion vectors
TNM(2)	/COM10/TORCOF/ TN modified for boundaries
TRAT(IMX)	/local/FBAR/ computed temperature ratio
TWOD	/COM13/input/ indicates 2-D input files
WDE	/COM2/WTEDGE/ weighting factor used when NEDGE set
WDES	/COM6/WTEDGE/ retained WDE
WDS	/COM2/WTEDGE/ as WDE, but at initial boundary
WDSS	/COM6/WTEDGE/ retained WDS
WEIGHT(IMX)	/COM2/LINE1,SOLUT/ ω, tension parameter
WT(IMX)	/COM6/GETWT/ ω_t, correction to ω
WTSUM	/local/GETB,LINE1,WTEDGE/ $\sum 1/\omega$
X(ID,JD,KD)	/COM1/input,UPDATE/ input grid points, i direction
XJ(IMX,3,3)	/COM8/BLOCK,INTXYZQ/ X data stored in block
Y(ID,JD,KD)	/COM1/input,UPDATE/ input grid points, j direction
YJ(IMX,3,3)	/COM8/BLOCK,INTXYZQ/ Y data stored in block
Z(ID,JD,KD)	/COM1/input,UPDATE/ input grid points, k direction
ZJ(IMX,3,3)	/COM8/BLOCK,INTXYZQ/ Z data stored in block

3. EXAMPLES of the SAGE CODE

This section contains many examples to familiarize the user with the adaptive-grid process. Each example includes plots of the initial grid and flow-field contours, the input-control-parameter file used to adapt the grid, the resultant adapted grid and a discussion on the choice of control parameters. In addition, several cases show the improved flow solution obtained from the flow solver using the adapted grid as input. The 2-D examples show the effect of within-plane parameters that are also needed for 3-D problems, so 3-D users should also study these examples.

3.1 Two-Dimensional Examples

The first set of examples are for 2-D problems. Although the code adapts the data as if it were a single plane in a 3-D environment, this is imperceptable to the user. When TWOD=.true. on the input parameter file, SAGE will expect the data to be in the 2-D format of PLOT3D. The index of (1) on the CLAM and CT parameters refers to within-plane variables. Index (2) is not needed for these 2-D cases.

Case 1. Flow in a Supersonic Inlet

Figure 12(a) shows the 101 × 79 initial grid (assigned to unit 7) for an inlet flow-field problem. Flow is from left to right and a shock emanates from the upper-wall corner, reflecting off the lower wall. In addition, an expansion fan originates from the downstream upper-wall corner and interacts with the reflected shock. The computed solution (input on unit 8), generated by the flow solver using this initial grid, is shown as density contours in figure 12(b). The SAGE code is now run to create a new grid that is more adapted to this solution, i.e., the grid points are redistributed with respect to a chosen flow-field variable (in this case, density) and output to the file assigned to unit 10. SAGE also interpolates the complete flow-field solution onto these new grid points and outputs this file to unit 11. The updated files will subsequently be input into the flow solver to produce more accurate solutions. Several examples of grid adaption are given for this inlet problem to demonstrate the effect of the control parameters.

Example 1: Single pass, stepping in j direction. Until the user is familiar with the basic parameters, the first attempt should be an input-control file with no parameters; i.e., all default values are used (with the exception of TWOD=.true.). The choice of all default values for this example case is shown in figure 12(c). The grid has been more evenly spaced and there is a slight clustering of points around the shock on the lower-wall boundary but the remaining grid lines are not adequately adapted to the flow features. This implies that the torsion parameter is too large and is overriding the local tension term, preventing the tension forces from pulling the points to the high-gradient regions. In addition, the minimum grid spacing is too large. The input-control file was therefore chosen to contain:

$namel $twod = .t., rdsmin = .2, clam(1) = .001, nedge = 1$

Only three control parameters are input: the minimum allowable grid spacing, the magnitude of the torsion term (an order of magnitude less than the default value), and a request for edge control (which is frequently used). The remaining parameters retain their default values, signifying that the full grid will be adapted, density is the adaption variable, and stepping is in the j direction. On completion of the execution, the message to unit 6 contains "ADAPTION 1 COMPLETE".

The result of this adaption is shown in figure 13(a). It can be seen that, since adaption began along the lower surface ($jst = 1$ is the default), the redistributed points cluster around the point of reflection on this line. However, points do not become sufficiently clustered at the corner shock and at the start of the expansion wave region on the upper surface. This is to be expected, since the adaption on the initial line is a 1-D solution and will pick up features quite clearly. However, as stepping continues, the features on subsequent lines get "dampened" by the torsion (i.e., smoothness) control.

Example 2: Adaption with backward j steps. This example maintains the same parameters as example 1, except that the upper surface is chosen as the initial adaption line, i.e., $jst = 79, jend = 1$. Figure 13(b) shows the result of this adaption. In this case, the upper surface is more clearly adapted, and the point of reflection on the lower surface is more spread out. Comparing these two examples indicates quite clearly that the starting line has a strong effect on the resultant adapted grid, and for some applications it should be chosen carefully.

Example 3: Adaption in i direction. This example shows the very different grid generated when the adaption is performed by stepping in the i direction ($jstep = .false.$). Based on the experience obtained from the first two examples, adapting from right to left will produce a better grid since there are no gradients on line $ist = 1$ to adapt to. Hence, the input control file was chosen to contain:

\$namel $twod = .t., jstep = .f., ist = 101, iend = 1, rdsmin = .25, clam(1) = .001, nedge = 1$\$

Figure 13(c) shows the result of this adaption. Not surprisingly, the reflected shock region on the lower surface is not "captured" by this adaption, and thus this grid is less suitable than those shown above. This demonstrates that the choice of stepping direction is important in producing a desirable grid.

Example 4: Effect of changing control parameters. This example is actually a collection of examples that show the effect of varying the control parameters $clam(1)$, $Ct(1)$, $rdsmax$, and $rdsmin$. Figure 14(a) shows an adapted grid using "baseline" values. These are the same as the default values in the code with the exceptions of $clam(1) = .001$ and adaption proceeds from top to bottom (i.e., $jst = 79, jend = 1$). Figures 14(b) through 14(f) are grids adapted by changing just one of the baseline parameters. Figure 14(b) shows the result with $clam(1) = .01$, and it is immediately obvious why this code default value was not chosen as the baseline value for these comparison cases: $clam$ is too large to allow any of the flow features to be "captured" by the adaption. Figure 14(c) shows the case for $clam(1) = .0001$, and this smaller value produces a grid with more points clustered around the shocks. Figures 14(d) and 14(e) show the effect of the $Ct(1)$ parameter: $Ct(1) = 1.0$ in figure 14(d), emphasizing straightness, and $Ct = .0$ in figure 11(e), emphasizing orthogonality. The latter shows how the orthogonality term dampens the adaption for this case: we are requesting orthogonality to the already parallel i lines. Finally, the effect of changing the minimum and maximum allowable grid spacing is shown in figure 14(f), where $rdsmax = 4.0$ and $rdsmin = .25$ are used. This mesh size control is only a factor of 2 different from that in the baseline example (figure 14(a)), but significantly densifies the spacing in the shock regions.

Example 5: Two-directional adaption. Two-directional adaption is created by adapting in one direction (e.g., stepping in j) and then adapting in the other direction (i.e., i), using the first adapted grid as input. The resultant grid will depend on the order of the stepping direction.

Two passes (i.e., two sets of input control parameters) were made to produce the adapted grid shown in figure 15(a). If both these passes are made during the same execution run (not to be recommended for a first attempt), then additional output files will be created on units 12 and 13. Data will be output for both passes, unless the SAVE parameter is set to false on the first set of control parameters. The control file for this example contains:

$namel $twod = .t., clam(1) = .001, rdsmin = .2, nedge = 1$$

$namel $twod = .t., jstep = .f., ist = 101, iend = 1, clam(1) = .001, rdsmin = .25, nedge = 1$$

The first pass steps in the j direction and uses the same input control parameters as example 1, and thus produces the adapted grid shown in figure 13(a). The second pass steps in i, and uses the same parameters as example 3. Note that any parameters that remain unchanged for the two passes still need to be redefined in the control file, since the code restores all default parameters at the conclusion of each pass.

The control file used to produce figure 15(b) contains the same parameters, but the adaption order is reversed. That is, the first pass steps in the i direction and the second pass in the j direction. Although the controls for the second pass are the same as for example 1, the input grid is different, giving a quite different adapted grid. The difference between figures 15(a) and 15(b) is obvious. Although this example shows that both adaptive sequences do adapt the grid, it should be noted that it is also possible that one order of adaption will produce a completely unacceptable result, while the reverse is quite suitable.

Example 6: Flow solution using adapted grid. Figures 16(a) and 16(b) demonstrate the improved flow-field solution obtained when the flow solver is rerun using an adapted grid as input. The adapted grid shown in figure 16(a) was obtained by requesting a significant amount of clustering in the high gradient regions: $rdsmax = 4.0, rdsmin = .25$, and $clam(1) = .0001$. In addition, adaption begins at the upper-wall surface. This grid and the interpolated flow-field variables were input to the flow solver, producing the flow solution (shown as density contours) seen in figure 16(b). The improvement in the resolution of the incident and reflected shock is obvious when compared to the initial solution shown in figure 12(b).

Case 2. Hypersonic Blunt-Body Flow

The second case contains two examples that demonstrate the effect of the CLAM and CT parameters. Figures 17(a) and 17(b) show an initial grid (32×32) and corresponding density flow-field contours for a hypersonic blunt body. The j direction marches out from the surface to the free-stream boundary, and the i direction marches along the body starting at the lowest point. This is a simple one-directional problem with the shock shape aligned with the grid. The outer-side boundary is curved (when marching in the i direction) and the default values of $clam = .01$ and $Ct = .5$ will prevent the adapted grid lines from "turning" sufficiently at this edge, giving either a false densing of points at the outer side boundary or, possibly, a critical error message. In this situation, two remedies are available. CLAM can be decreased, hence de-emphasizing the effect of torsion and thereby allowing the tension force to "pull" the nodes toward the shock wave. Alternatively, the orthogonality restraint can be removed by setting $Ct = 1.0$. Figure 17(c) shows the result of setting $clam = .001$ and retaining all other parameters at their default value. The adapted grid obtained by setting $Ct = 1.0$ is very similar and is not shown here. The following are the two input-control files:

$namel $twod = .t., jstep = .f., clam(1) = .001$ and $namel $twod = .t., jstep = .f., ct(1) = 1.0$

Both the adapted grids show the required result, i.e., the clustering of grid points across the shock region.

Case 3. Blunt-Body Shock Impingement Problem

The third case introduces the use of IQ and ORTHE parameters. Figure 18(a) shows the input grid of a cowl/lip shock interaction problem. The j direction marches from the body surface to the outer free stream and the i direction marches from the lower point of the sphere. The flow-field contours of both density (figure 18(b)) and Mach number (figure 18(c)) are given, since the adaption flow-field parameter is a combination of the two. The blunt body shocks, impinging shock, and shear layers represent a complex flow that requires adaption in both directions to adequately capture all the flow features. Figure 18(d) shows the adapted grid produced by the following 2-pass control file:

$namel $twod = .t., jstep = .f., ist = 91, iend = 1, indq = 0, iq(1) = 2, iq(7) = 1,$
$rdsmin = .25, ct = .7, clam(1) = .005, nedge = 2$
$namel $twod = .t., indq = 0, iq(1) = 2, iq(7) = 1, rdsmin = .3, orthe(1) = .f.$

The first pass steps in the i direction, starting from the topmost boundary. The parameter $indq = 0$ indicates that IQ will be input, and in this case two-thirds of the density function and one-third of the Mach function will be combined to become the adaption variable. As explained in case 2, the curvature of the outer boundary requires increasing CT. Setting $nedge = 2$ requests that only the side-edge boundary at $j = 1$ be treated. The second pass introduces the use of the ORTHE variable: stepping occurs from the sphere wall to the outer curved boundary, where the default would turn the grid to be normal to this outer boundary. ORTHE overrides this and allows the adaption to occur naturally.

Case 4. Hypersonic Inlet (Zonal Adaption)

Figures 19(a) and 19(b) show the initial grid and density contours for a hypersonic cowl/lip inlet problem. This is a more complex case and requires dividing the adaption domain into zones—the blunt body region and the rectangular inlet region. The upper wall is the $j = 1$ line and the outgoing channel region (on the right side of the diagram) is the $i = 1$ line. Five adaption passes are required to create the final adapted grid shown in figure 19(c). The input control file consists of

$namel $twod = .t., rdsmin = 1.0, rdsmax = 1.0$
$namel $twod = .t., jstep = .f., rdsmin = .2, nedge = 1$
$namel $twod = .t., ist = 70$
$namel $twod = .t., jst = 32, jend = 1, iend = 71, rdsmin = .25, clam(1) = .02, nedge = 1$
$namel $twod = .t., iend = 85, jst = 19, jend = 1, clam(1) = .002, nedge = 1, mgsteps = 5$

The first pass, with equal maximum and minimum spacing, spreads out the i grid lines evenly; the original grid points were more densely distributed in the curved section, leaving fewer grid lines in the inlet area. This is a good example of using the program to improve an initial grid, with no flow-field adaption involved. The second pass steps in the i direction and adapts easily throughout the entire grid. The third, fourth, and fifth passes perform the adaption in the j direction. The

very different flow features in the blunt-body region (blunt-body shock) compared to the features in the inlet region (Mach stem and reflected shocks) indicate dividing the adaption domain into these two zones: $i < 70$ and $i > 70$. Only the blunt-body section ($i > 70$) is adapted in pass three, with all default-control parameters. The adaption in the inlet domain starts at $jst = 32$ in order to pick up the Mach stem along the lower wall. After stepping through the triple-point region (the intersection of the cowl shock and the reflected normal shock), the redistributed points do not spread out sufficiently, so the adaption is stopped at line 19 and a fifth pass is started at line 19 with a decreased value of CLAM. Since this pass starts internally to the original adaption, it is necessary to set MGSTEPS in order to merge smoothly from the already adapted region.

Case 5. Axisymmetric Plume Flow

This case is presented to indicate the powerful effect of the adaptive grid process on the final flow-field solution. Figure 20(a) shows the initial grid and computed solution (in Mach contours) of an axisymmetric nozzle-plume flow. This initial solution has not developed sufficiently to capture the final flow features: the outer shear layer, barrel shock, Mach stem, reflected shock, and the triple-point shear layer. Three iterations (through both adaption and flow solver) were made to produce the final adapted grid and solution shown in figure 20(b). The definition of the flow features is greatly improved. Figure 20(c) shows the accuracy of the final solution: the lower picture is a shadowgraph of the actual experiment and is almost mirrored by the computed solution.

Case 6. Subsonic Impinging Jet

This example is to show the use of the SUB parameter. The original grid, with 231 × 100 grid points, and the corresponding Mach contours are shown in figures 21(a) and (b). To remove points from both coordinate directions, the input-control file requires two passes for the adaption:

$\$namel\ sub = 1, rdsmax = 3.0, rdsmin = .2, clam(1) = .005, jst = 100, jend = 1,$
$nedge = 1, indq = 6, twod = .t.\$$
$\$namel\ sub = 1, jstep = .f., noup = .t., twod = .t.\$$

Both passes use the SUB parameter, reducing the number of points to a 116 × 51 grid as shown in figure 21(c). The example shows that the SUB (as with the ADD parameter) can be used in conjunction with an adaption (as in the first pass) or simply as a method to reduce the number of points in the original grid (as in the second pass). It should be noted that if the two passes had been in reverse order, the input value of jst would need to be modified by the user to $jst = 51$.

3.2 Three-Dimensional Examples

The choice of input parameters when adapting 3-D grids is more complicated, not only because of the additional torsion control, but because of the increased choice of stepping and marching directions. For 2-D adaption, four choices of adaption are available: stepping in the i direction, the j direction, or both (in either order). Each of these options will produce a different adapted grid. For 3-D adaption, it is also necessary to choose a plane-stepping direction and, in theory, to consider up to three possible passes. In practice, it has been shown that a one-directional pass

will frequently provide a sufficiently adapted grid, and that a two-directional pass is the maximum required. The following examples show the effect of single- and multi-directional passes.

Case 7: Tandem Fuel Injectors in a Supersonic Combustor (Rockwell Model).

Figure 22 shows a two-slot, tandem fuel injector arranged behind a backward facing step in a supersonic stream that models the Rockwell supersonic combustor. The fuel injection through the slot nozzle creates a complicated three-dimensional flow pattern. Since the outflow is supersonic, fuel injection normal to the main flow produces strong shock waves and streamwise separation in the vicinity of the slots. In addition, the backward facing step locally creates a subsonic flow ahead of the slots. In this example, grid adaption is limited to the region downstream of the backward facing step. The grid and initial solutions for the fuel injection problem were provided by J. Wang.

Example 1. One adaption pass showing 3-D effect. This example is used to demonstrate how a one-directional adaption (i.e., one pass of the program) changes the grid-point distribution, not only on the chosen adaption plane, but also on the two cross planes. The initial grid ($80 \times 31 \times 61$), given in figure 23(a), shows three selected planes from Wang's initial grid, one in each of the coordinate directions, where $i = 40$, $j = 1$ and $k = 30$. For clarity, these planes are separated (but with their original orientations) and shown in figure 23(b). Figure 23(c) presents the Mach contours obtained from the solution on the initial grid by the flow-field code, corresponding to each plane. The adaption is performed by redistributing the points on the first j plane and then marching in j planes (obtained by setting $ikplane = .t.$). Within each j plane, adaption is performed along k lines, i.e., stepping in the i direction by setting $istep = .t.$. The input parameter file used to create the adapted grids shown in figure 23(d) is:

$$\$\text{namel } ikplane = .t., istep = .t., rdsmax = 4.0, rdsmin = .1, indq = 7, clam(1) = .0001,$$
$$clam(2) = .0005, nedge = 1\$$$

As requested, the j plane has clearly adapted to the Mach number gradients in the k direction, but so has the i plane, since this was the stepping direction within planes. However, the curvature of the planes themselves has not changed. The k plane shows a different effect: points have not moved within the plane, but only up and down (again, the k direction), thus changing the curvature of the plane. This example clearly shows, that although the adaption is in only one direction, all planes are affected. It is thus quite possible for one pass to adequately adapt the grid for re-input to the flow-field code.

Example 2. Two-directional pass This combustor problem is one that benefits from a two-directional pass. By looking at the constant j plane in figure 23(d), it can be seen that a second adaption would be appropriate, by adapting in the i direction and stepping in k lines. The second set of adaption parameters are

$$\$\text{namel } ikplane = .t., kstep = .t., rdsmax = 4.0, rdsmin = .1, indq = 7, clam(1) = .0001,$$
$$clam(2) = .0005, nedge = 1\$$$

Figure 24(a) shows the result of performing this adaption "on top" of the adaption already shown in figure 23(d); points have now been redistibuted in the i direction as well as the k direction. However, it should be noted that the i direction can also be adapted by marching in j within constant k planes by using:

$namel $ijplane = .t., jstep = .t., rdsmax = 4.0, rdsmin = .1, indq = 7, clam(1) = .0001,$
$clam(2) = .0005, nedge = 1$$

This adapted grid is shown in figure 24(b) and shows a very similar redistribution on the j plane, even though this is not the adaption plane.

Example 3. Torsion parameter between planes, λ^*. Users familiar with the 2-D code know how the torsion parameter (λ) affects the adaption within a plane: this parameter controls the magnitude of the torsion term and the larger the value of λ, the smoother the resulting grid, but with less grid redistribution. This effect is demonstrated in case 1 in the 2-D section. Unfortunately, the "base" value of λ varies for each problem: the default value in the code is .01 but the user may have to change this value by as much as one, two or even more orders of magnitude. λ^* is the analogous torsion parameter between planes, and acts in a similar manner to λ, but restricts the movement of points from plane to plane to maintain smoothness in the cross-planes. If $\lambda^* = 0$, there is no torsion control between planes, and planes will be adapted as independent entities. (In fact, setting $\lambda^* = 0$ is one way of handling periodic planes.) In the code, λ^* is defaulted to .0001 and this example illustrates the effect of this parameter.
The same combustor-flow problem from example 1 is used with different input control parameters. This time, we are adapting k planes. Figure 25(a) shows the initial distribution of two k planes (reflecting the upper and lower surface) joined by the first j plane. For problem-related adaption, the first k plane would not be adapted, since the maintenance of its initial spacing is a high priority. However, to illustrate the adaption procedure, in this example the first plane is adapted. The input parameter file giving the result shown in figure 25(b) is

$namel $ijplane = .t., jstep = .t., clam(1) = .001, clam(2) = .001, rdsmax = 4.0,$
$rdsmin = .15, nedge = 1$$

This input requests marching in k planes (since $ijplane = .t.$) and stepping in the j direction within the k plane. The only parameter with any control of the j plane is λ^* ($clam(2)$) and the j plane shows some redistribution of points, but insufficient to reflect the flow features. Compare this plane to that in figure 25(c), where the adaption was performed with the identical input parameters except for a decrease in λ^* to .0001. The effect on the smoothing between planes is very evident and indicates the influence of the torsion parameter between planes.

Case 8: NASP 3-D Nozzle Simulation.

Example 1. Two-directional adaption. Figure 26 shows the geometry of a wind-tunnel-experimental model for the National AeroSpace Plane (NASP) called the Single Expansion Ramp Nozzle (SERN). The model is inserted into a hypersonic test section with cold air injected at supersonic speed through the nozzle. Superimposed on the ramp section is an outline of the computational grid which is detailed in figure 27(a). This 3-D flow-field grid (41x60x90) defines the nozzle and after-body region of the model that is used in the computational experiments.
The scramjet nozzle is in the lower left corner of the nozzle-exit plane at $i = 1$. Also shown in the figure is the downstream outflow plane at $i = 41$, the lower grid at $k = 31$ (part of which is the upper surface of the after-body ramp) and the symmetry plane at $j = 1$. Additional patched grids are not shown, but there is a grid boundary that must be matched from $j = 41$ to $j = 60$ along the $k = 31$ plane with the grid that is stored in $k = 1$ to $k = 30$, and also a matching $i = 1$ plane. Adaptions were performed from plane $i = 2$ to $i = 41$, stepping in both j and k directions

within the plane. This complex case uses many of the available input options and the adaption parameters used to produce the adapted grid in figure 27(b) were:

$namel $jkplane = .t., kstep = .t., rdsmax = 2.5, rdsmin = .25, kst = 32, kend = 83,$
$indq = 6, nedge = 1, clam(1) = .1, clam(2) = .01, mgsteps = 4, mgpls = 4,$
$march = .t., ct(1) = .75, save = .f.$$

$namel $jkplane = .t., jstep = .t., rdsmax = 1.25, rdsmin = .25, kst = 35, indq = 6,$
$nedge = 2, mg1 = 8, clam(1) = .1, clam(2) = .01, mgpls = 4, ct(1) = .75$$

The first set of parameters requests the adaption of each i plane ($jkplane = .t.$), stepping in the k direction ($kstep = .t.$) within the plane. Pressure ($indq = 6$) is the adaption parameter. Adaption begins on line $k = 32$ with the merging technique invoked ($mgsteps = 4$). This will retain the first two lines to match existing lower boundaries and ensure that the adaption parameters are filtered for the next four adaption lines. Both torsion parameters and the directional parameter, $ct(1)$ are larger than the default values since a previous test run showed that too much adaption (and thus loss of smoothness) occured using the default values. Another parameter used in this example is MARCH. This was invoked from line $kend = 83$: after this line, the flow has no gradient features and is mostly numerical "noise", and MARCH simply presents a more attractive grid. Since the adaption algorithm normalizes the weighting function, "noise" can produce unnecessary redistribution. Since the $i = 1$ plane was not to be adapted, a smooth transition was required between the first plane and subsequent planes. This was controlled by $mgpls = 4$: the default start plane of $ist = 1$ was not adapted and the next four planes were more gradually adapted than if MGPLS was not requested. Example 2 of this case gives a detailed account of this plane merging process. The SAVE parameter was set to false to prevent the output datasets from being written. This should be done only when more than one adaption pass is made in the same computer run.

The second adaption set requests adaption of the same i planes, but stepping in the j direction within planes. The first adaption point is $k = 35$. This was chosen to retain the first few points in the boundary layer. In addition, the parameters $nedge = 2$ and $mg1 = 8$ are used to filter this boundary layer spacing into the next eight grid points on the line. The value of $nedge$ was changed from 1 to 2 for this pass because there was no need to maintain any spacing at the outer edge, where no gradients are found: these points are better used within the body of the grid. The final adapted grid given in figure 27(b) shows the unadapted $i = 1$ and $k = 31$ planes, but all other planes have been adapted although only the outer planes are shown in the figure. Since it is difficult to portray the 3-D grid, the downstream outflow plane ($i = 41$) is shown in more detail in figure 28. Figure 28(a) shows the initial grid at $i = 41$. Figure 28(b) gives the corresponding pressure gradients that were computed by the flow-field code. The flow features indicate that adaption in both the k and j directions are necessary. The adapted grid computed by using the input parameters given previously is shown in figure 28(c). It must be noted that this is the last plane to be adapted and spacing control has been passed from the previously adapted planes to maintain grid smoothness.

Example 2. Merging from non-adapted planes to adapted planes. The nature of three-dimensional problems frequently necessitates the maintenace of boundaries: either for multiple grids or to preserve solid geometry walls and even for maintaining boundary layers. For the NASP nozzle case described here, the plane at $i = 1$ must match another plane on the adjacent (not shown) grid. There is also a dense grid spacing around the nozzle exit for defining the bound-

ary layer region that should also be retained. The simplest way to handle these two situations is to not adapt the first plane, but to adapt the contiguous planes in a merging manner. This merging is needed to create a smooth transition in the cross plane, not in the adaption plane itself (which is handled by MGSTEPS). Figure 29(a) shows the initial $i = 1$ and the identical $i = 2$ plane for the 3-D grid shown in figure 27 and figure 29(b) contains the corresponding pressure gradients for the $i = 2$ plane. If the input adaption parameter *ist* is set equal to 2, (i.e., the first plane is ignored) the resultant adapted ($i = 2$) plane is given in figure 29(c). Although this grid is smooth and nicely adapted to the flow, the cross plane (i.e., j) is not (compare this with figure 29(a)). In addition, points have been drastically pulled away from the nozzle region. By invoking the MGPLS option, this cross plane unevenness can be dampened. The technique is analogous to that used by MGSTEPS: both λ^* and C_{t_2} are modified to increase the restraint of movement of points between planes. These values will return to their original input values within a certain number of planes, as requested by MGPLS. The following is the input parameter file used to create the adapted grid ($i = 2$ plane) shown in figure 29(d):

$$\text{\$namel } jkplane = .t., jstep = .t., rdsmax = 1.25, rdsmin = .25, kst = 35,$$
$$indq = 6, nedge = 2, mg1 = 8, clam(1) = .1, clam(2) = .01, ct(1) = .75, mgpls = 4\$$$

The difference between figures 29(c) and (d) is striking: the boundary layer spacing is now maintained and the adaption process is just beginning. Although *mgpls* equals four in this example, setting $mgpls = 1$ will also have considerable effect. By setting MGPLS at all, a request is made to not adapt the first requested plane (in this case, $ist = 1$, by default) but to use the grid on that plane as a control for the subsequent planes. Thus, $mgpls = 1, ist = 1$ is not the same as setting $ist = 2$.

Case 9. Aeroassist Flight Experiment (AFE) vehicle

This example is the hypersonic, non-equilibrium flow around the forebody of the aeroassisted flight experiment (AFE) vehicle shown in figure 30(a). In figure 30(b) the initial grid configuration (35x23x49) around the body is outlined from a different viewpoint, in the form of j planes 2,17 and 22 and the outflow plane at $i = 35$. The pressure contours seen in figure 30(c) were computed by the flow-field solver of Palmer (1990) using the non-adapted grid. This grid was then adapted with respect to these pressure contours, and the resulting grid is seen in figure 31(a). The redistribution of points within the blunt-body shock region is clearly shown. Due to the smooth shape of the shock, adaption was performed only in one direction: marching in j planes and stepping in the i direction within planes. The input data set used to create the adapted grid in figure 31(a) was

$$\text{\$namel } lnsing = 1, ikplane = .t., istep = .t., indq = 6, jst = 2, jend = 22,$$
$$rdsmax = 4.0, rdsmin = .1, kst = 15, orthe(1) = .f., nedge = 1\$$$

The LNSING parameter is new. It is set since all j planes emanate from a single line at $i = 1$. (This forebody singular line is used in grid mapping.) However, the dataset still contains the grid points for line $i = 1$ as if it were a separate line in each plane. $lnsing = 1$ ensures that the first line in the first plane is adapted, and that all $i = 1$ lines on subsequent planes are set identical to this adapted line. The parameter $orthe(1)$ was set to $.false.$, which removes the orthogonality constraint at the outflow line of $i = 35$. The default value of *.true.* would have created a false turning of the adaption at this location. This adapted grid was then re-input to Palmer's (1990) flow-solver and the resultant pressure contours are given in figure 31(b). By comparing this figure with figure 30(c), it can be seen that the blunt-body shock feature has sharpened considerably.

REFERENCES

Davies, C.B.; and Venkatapathy, E.: Application of a Solution Adaptive Grid Scheme, SAGE, to Complex Three-Dimensional Flows. AIAA Paper 91-1594, June 1991.

Nakahashi, K.; and Deiwert, G.S.: A Self-Adaptive-Grid Method with Application to Airfoil Flows. AIAA Paper 85-1525, July 1985.

Davies, C.B.; and Venkatapathy, E.: A Simplified Self-Adaptive Grid Method, SAGE. NASA TM-102198, Oct. 1989

Palmer, G.: Enhanced Thermochemical Nonequilibrium Computations of Flow Around the Aeroassist Flight Experiment Vehicle. AIAA Paper 90-1702.

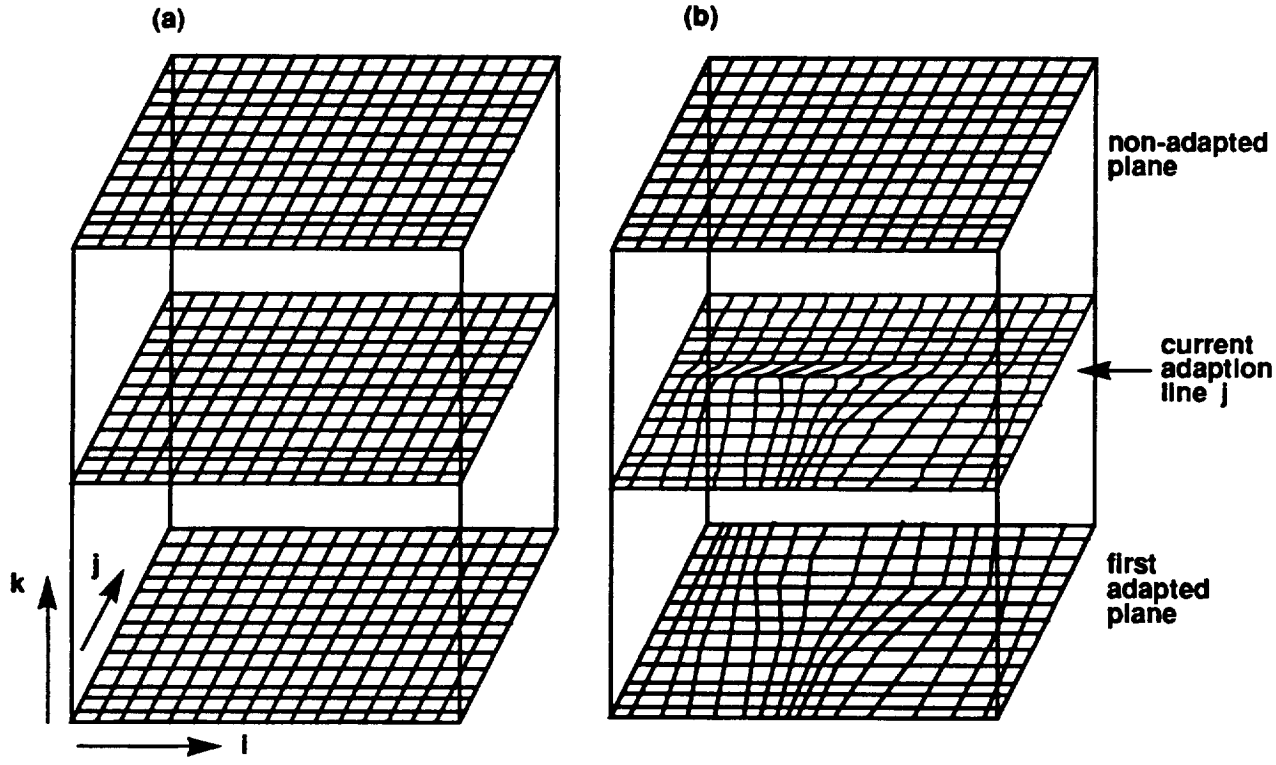


Figure 1. 3-D adaption. (a) Initial grid; (b) first directional adaption.

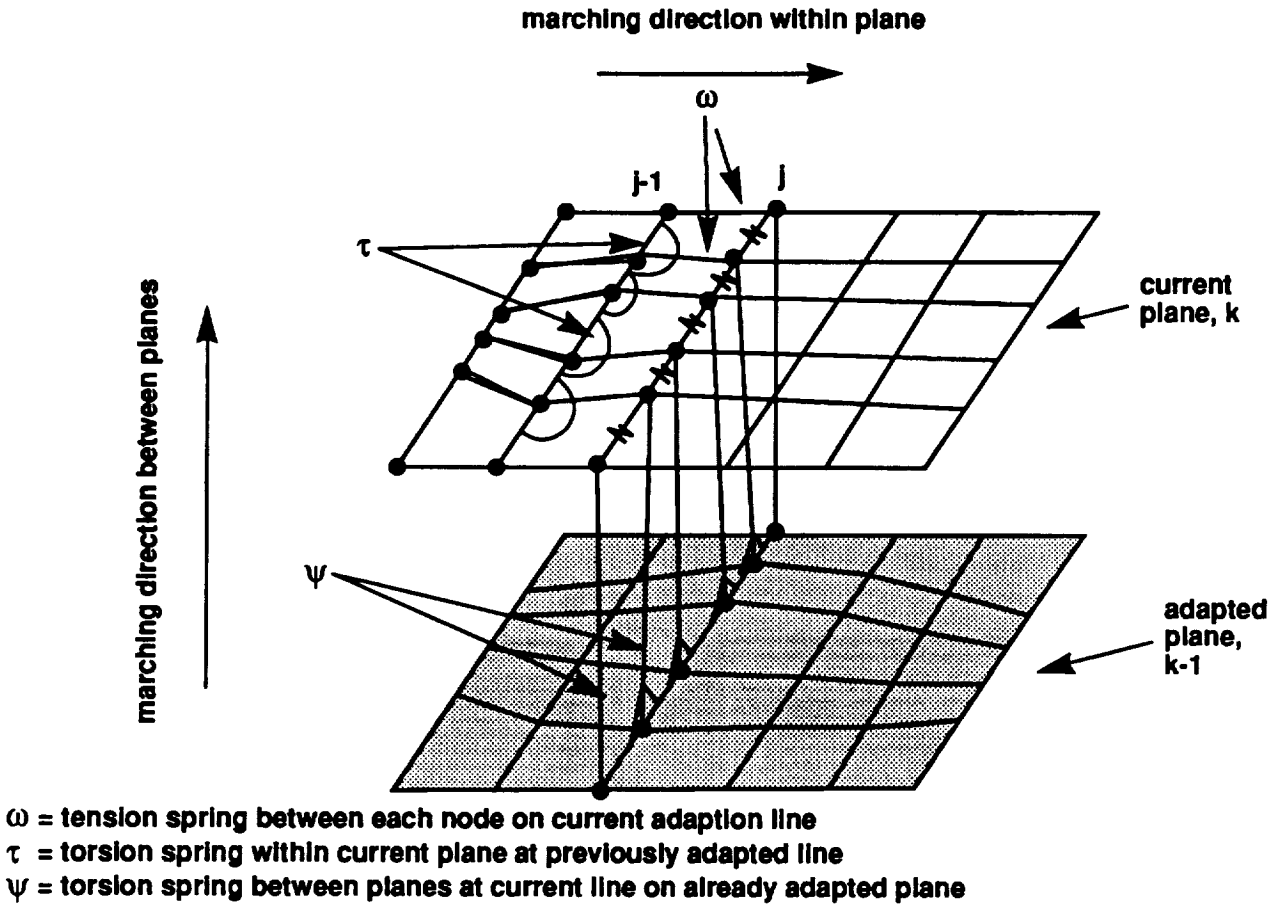


Figure 2. Line-by-line adaption, showing tension and torsion forces.

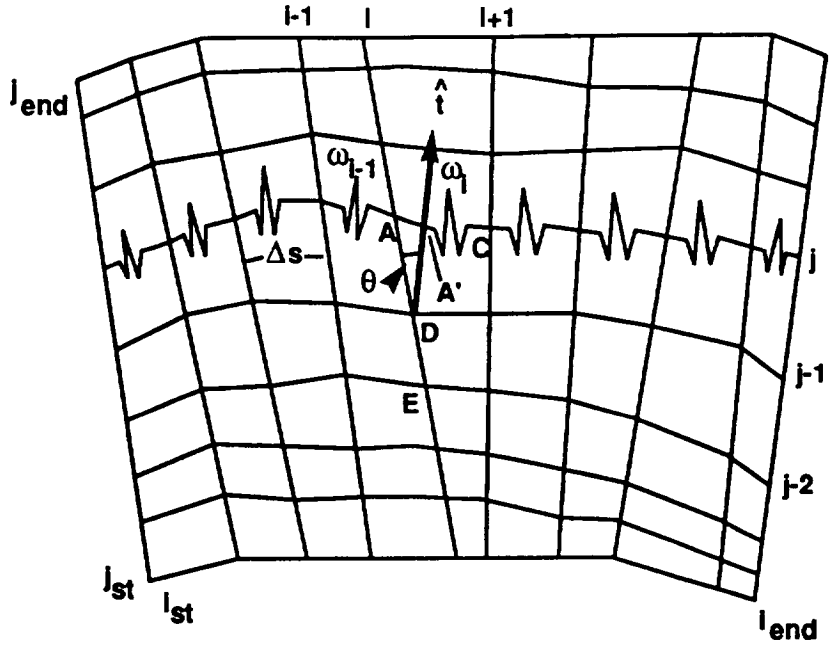


Figure 3. Current adaption line j, shown on a 2-D surface.

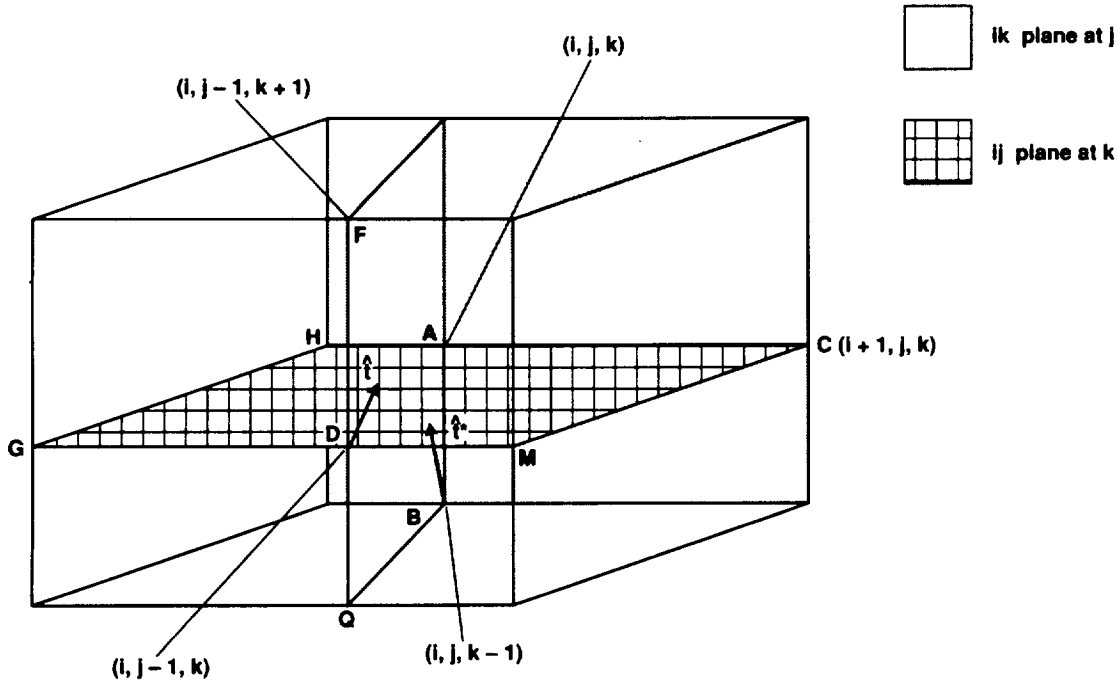


Figure 4. Torsion vectors, $\hat{t}$ and $\hat{t}^*$.

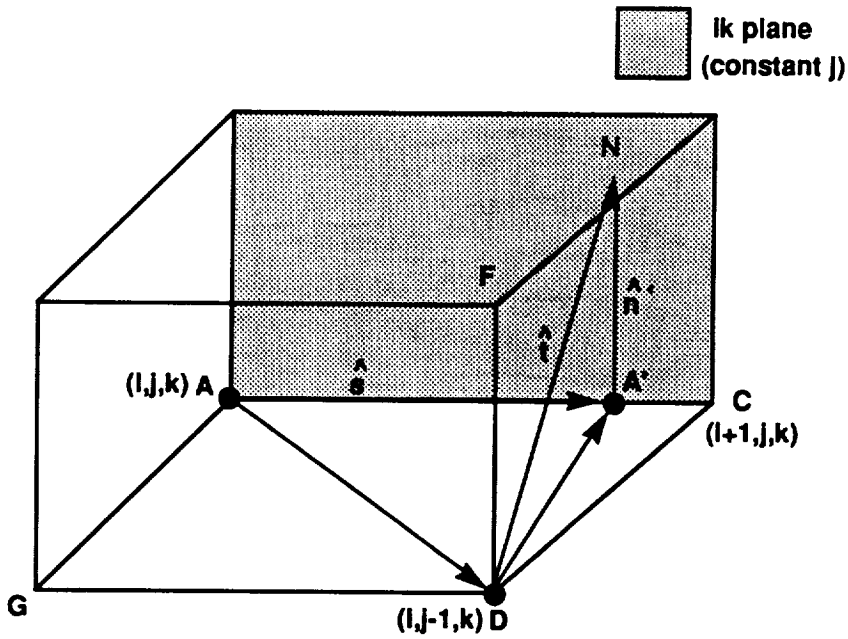


Figure 5. Calculation of A', intersection of torsion and streamwise vectors.

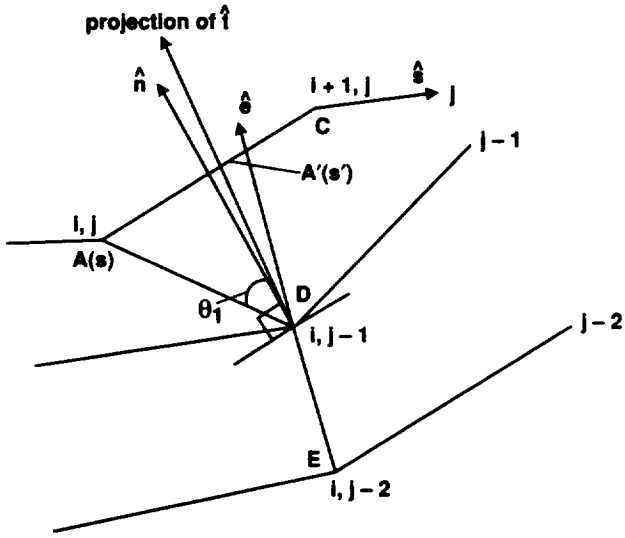


Figure 6. $\hat{t}$ as a combination of normal and straightness vectors, shown in 2-D.

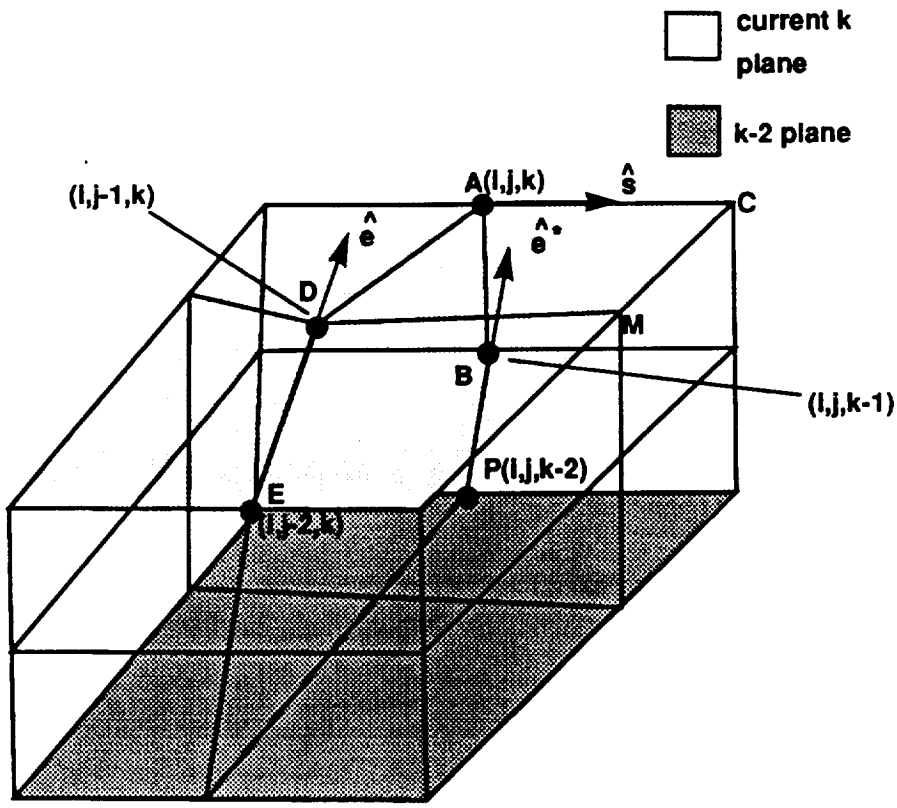


Figure 7. Straightness vectors, $\hat{e}$ and $\hat{e}^*$.

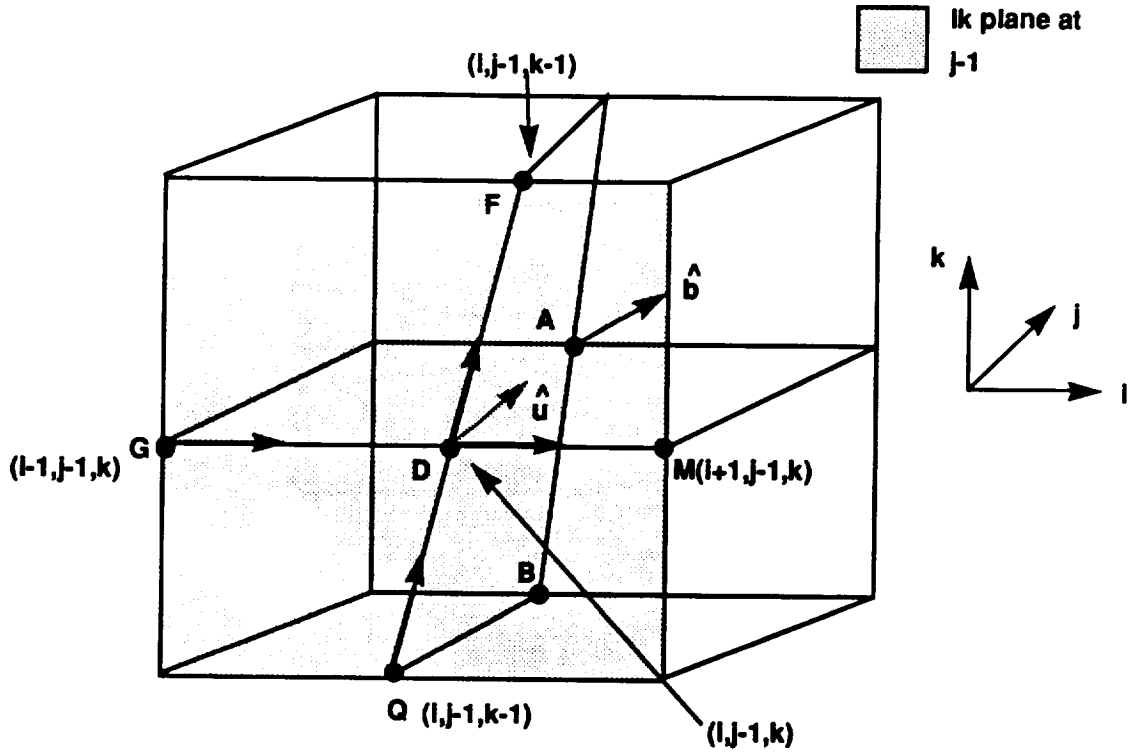


Figure 8. Normal vectors, $\hat{u}$ and $\hat{b}$.

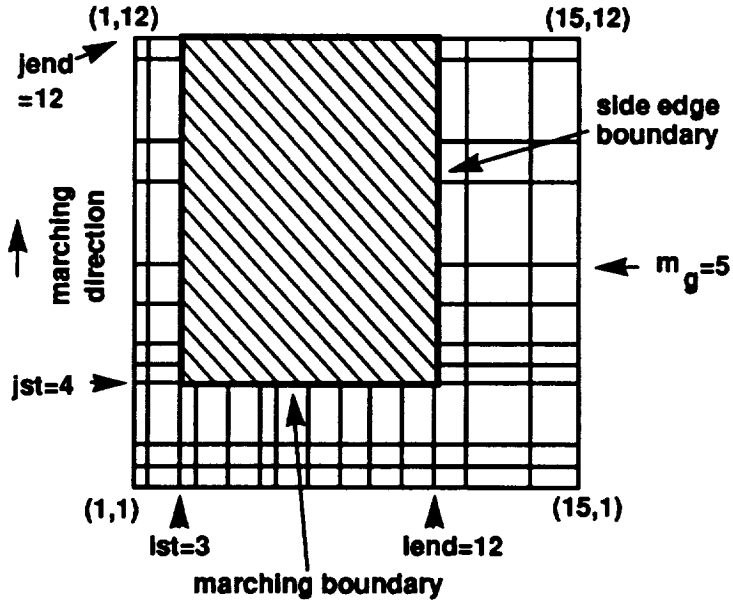


Figure 9. Adaption domain as a subset of the physical domain.

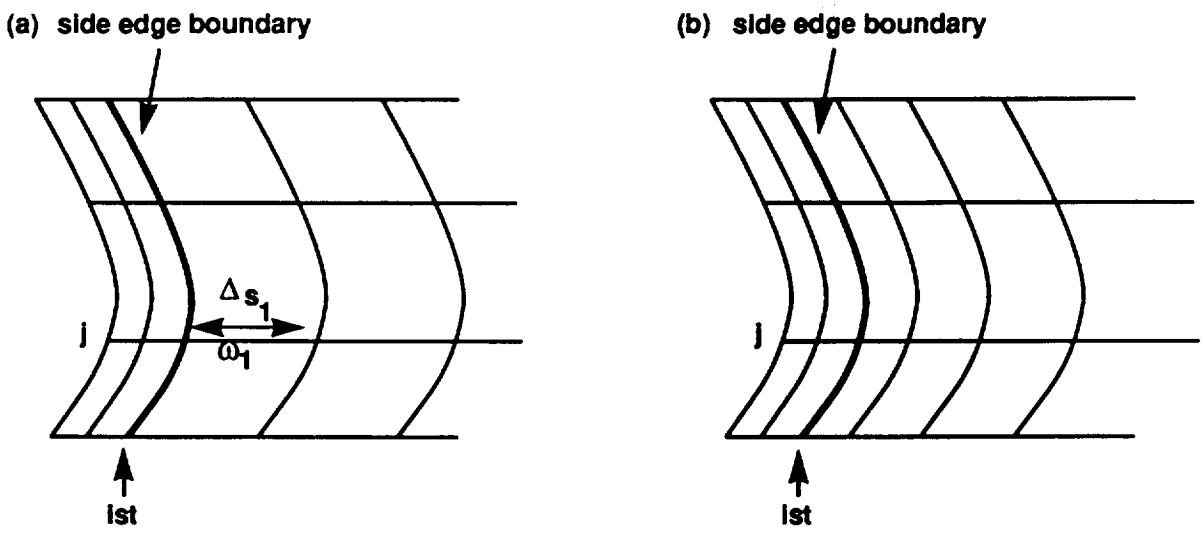


Figure 10. Control of side-edge boundaries. (a) With no edge control (nedge=0); (b) with edge control.

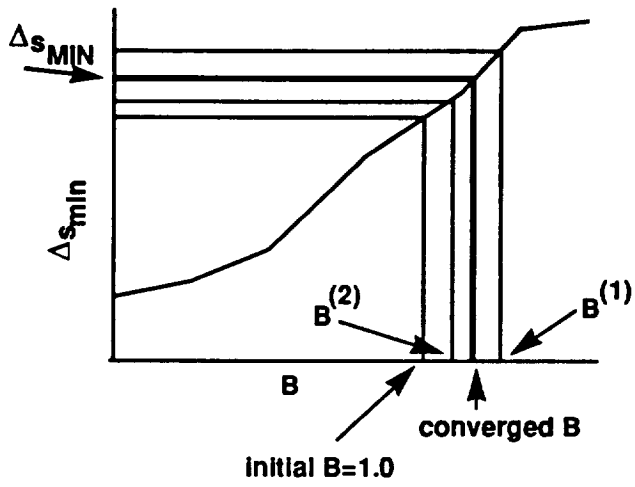
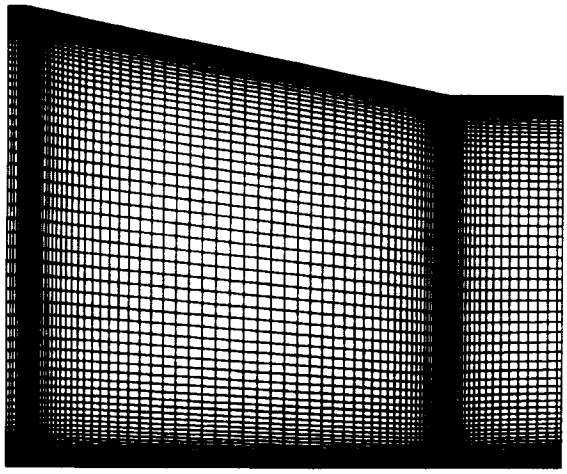
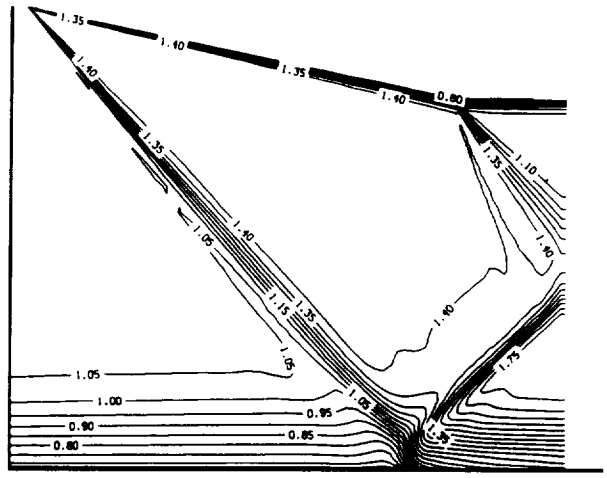


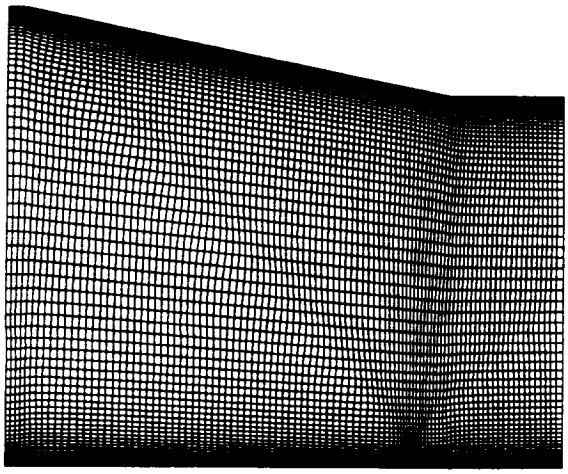
Figure 11. Calculation of B.



(a)



(b)



(c)

Figure 12. Flow in a supersonic inlet. (a) Initial grid; (b) density contours; (c) adapted grid using all default parameters.

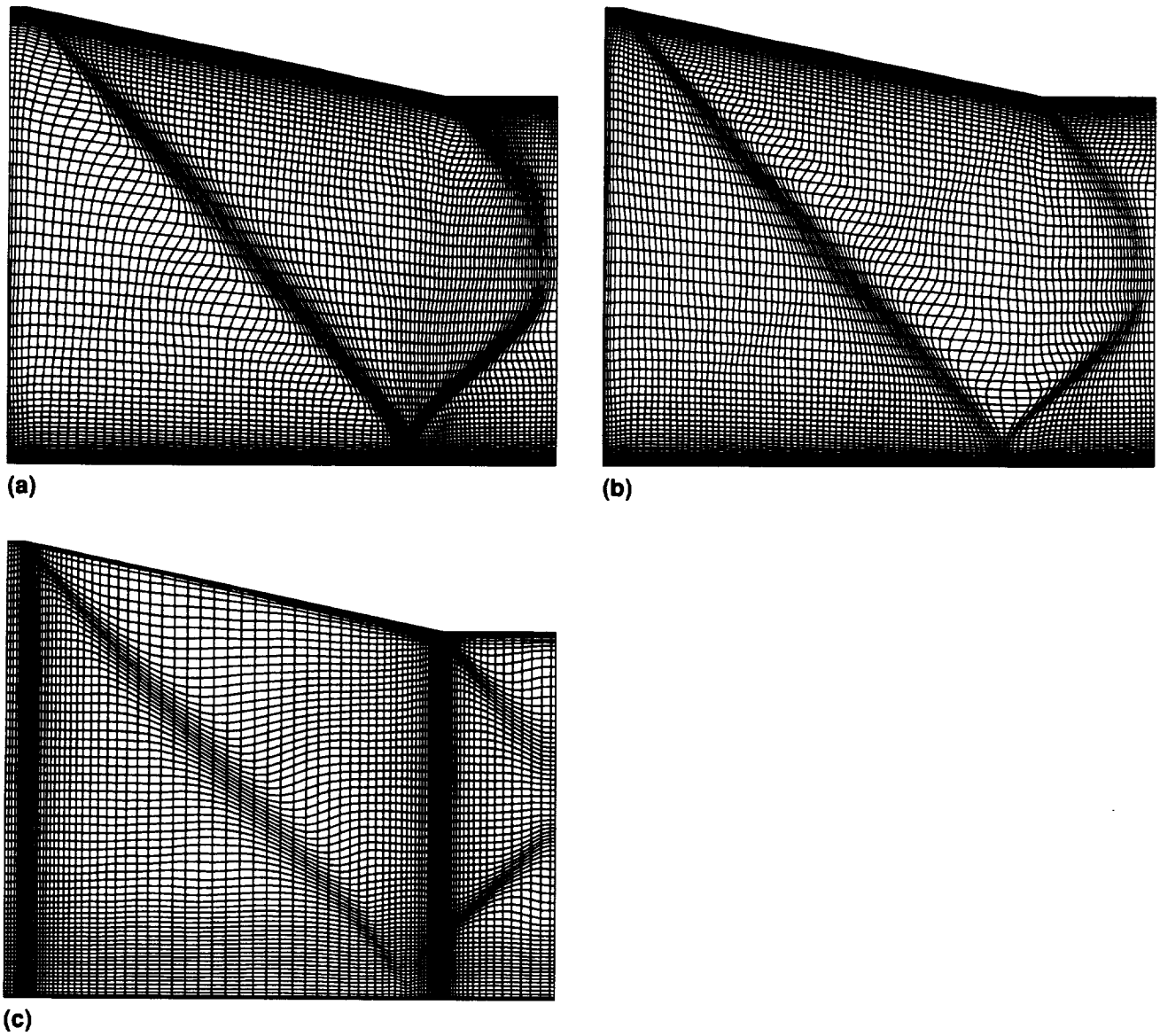


Figure 13. Effect of adaption direction. (a) Adaption from bottom to top; (b) adaption from top to bottom, $jst = 79, jend = 1$; (c) marching in i, from right to left, $jstep = false, ist = 101, iend = 1$.

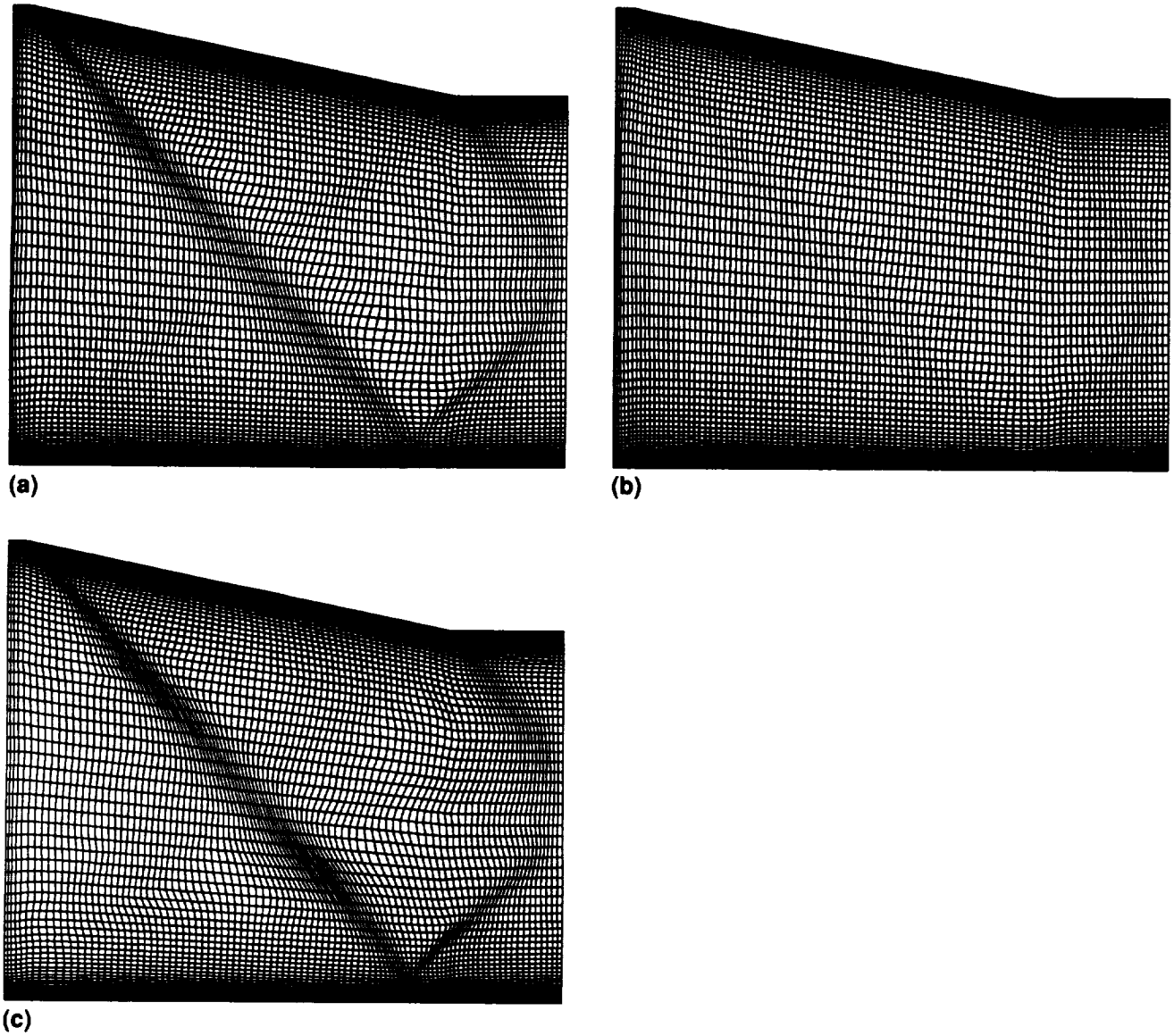
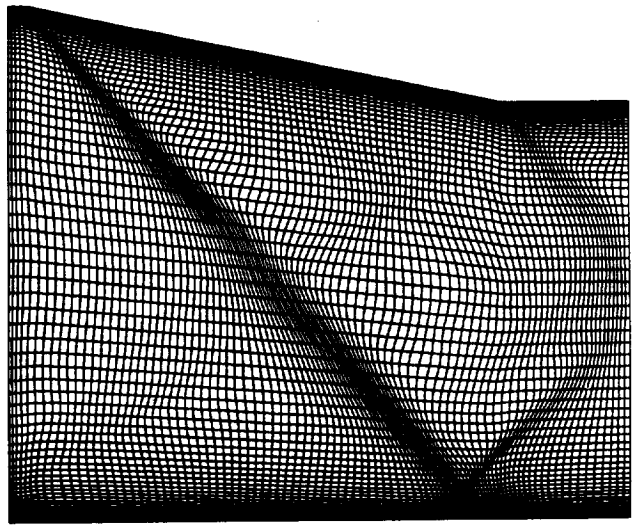
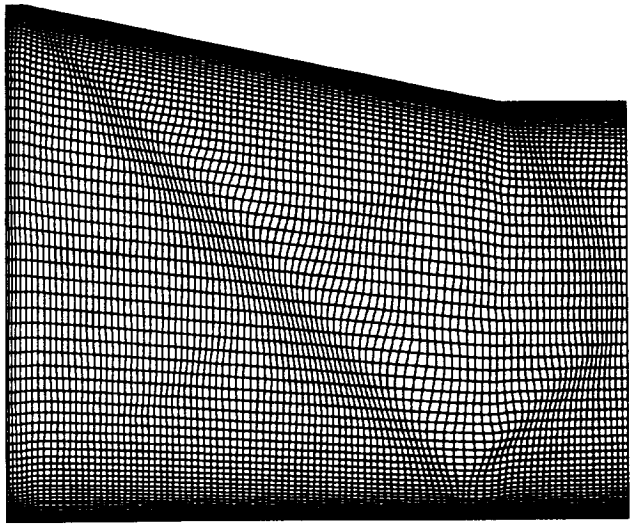


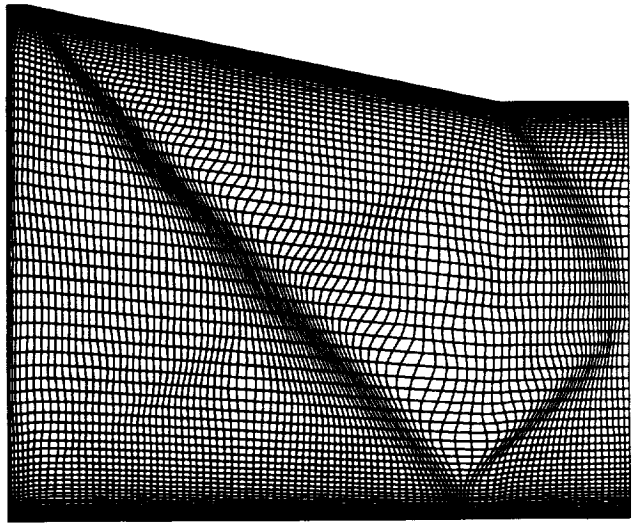
Figure 14. Effect of control parameters. (a) "Baseline" adapted grid, $\lambda = .001, jst = 79, jend = 1$; (b) $\lambda = .01$; (c) $\lambda = .0001$.



(d)



(e)



(f)

Figure 14. Concluded. (d) $C_t = 1.0$; (e) $C_t = .0$; (f) $\Delta s_{min} = .25, \Delta s_{max} = 4.0$.

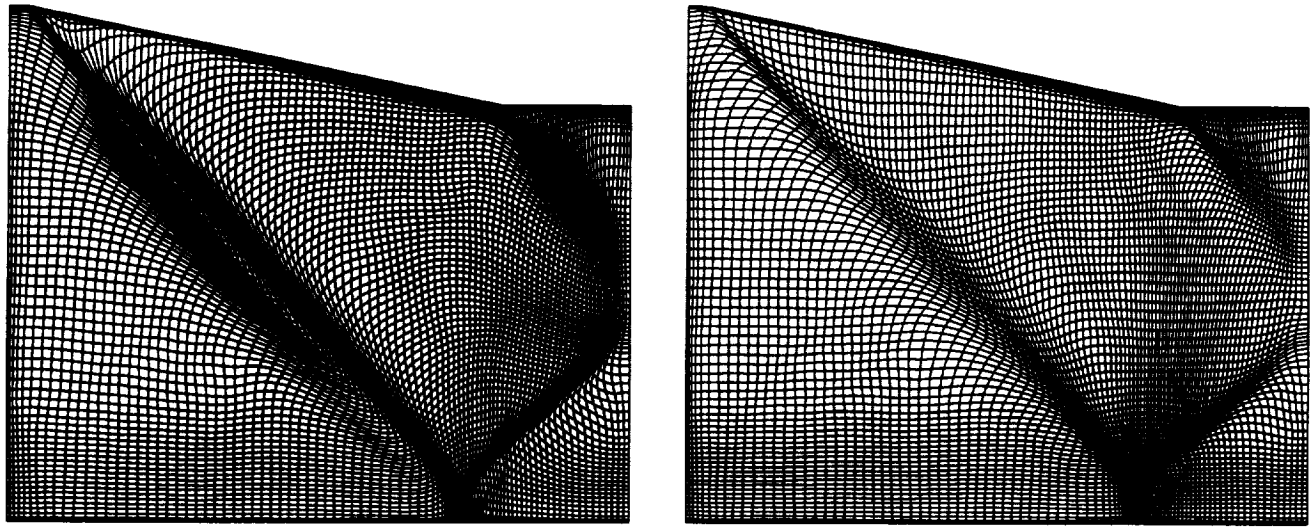


Figure 15. Two-directional adaption. (a) Marching in j followed by marching in i; (b) marching in i followed by j.

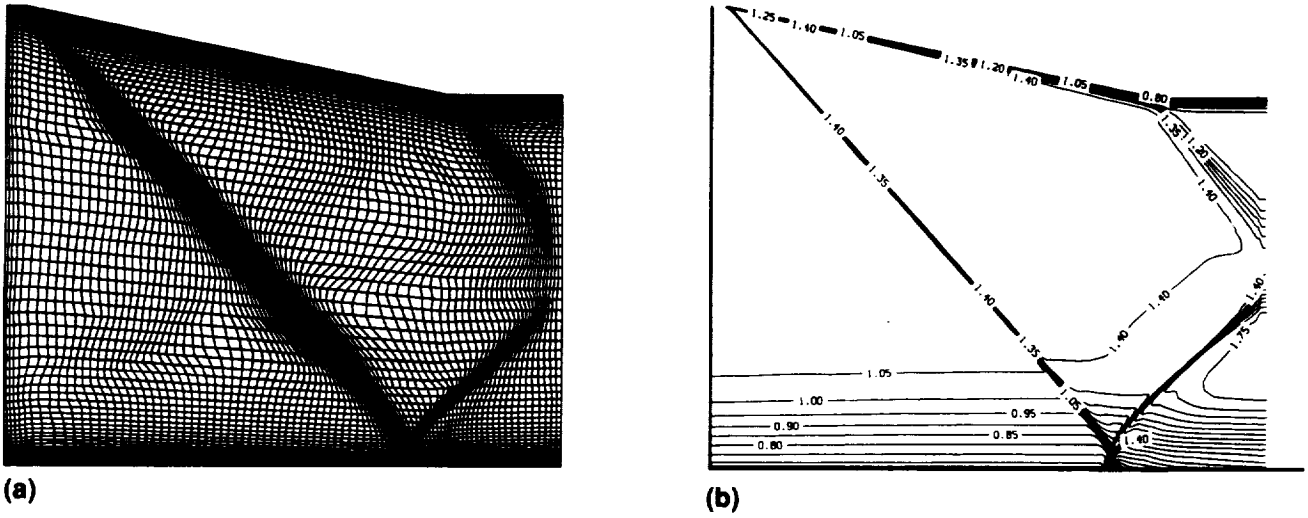


Figure 16. Final solution using adapted grid as input. (a) "Best" adaption, $\Delta s_{min} = .25, \Delta s_{max} = 4.0, \lambda = .0001$; (b) solution density contours, using "best" adapted grid.

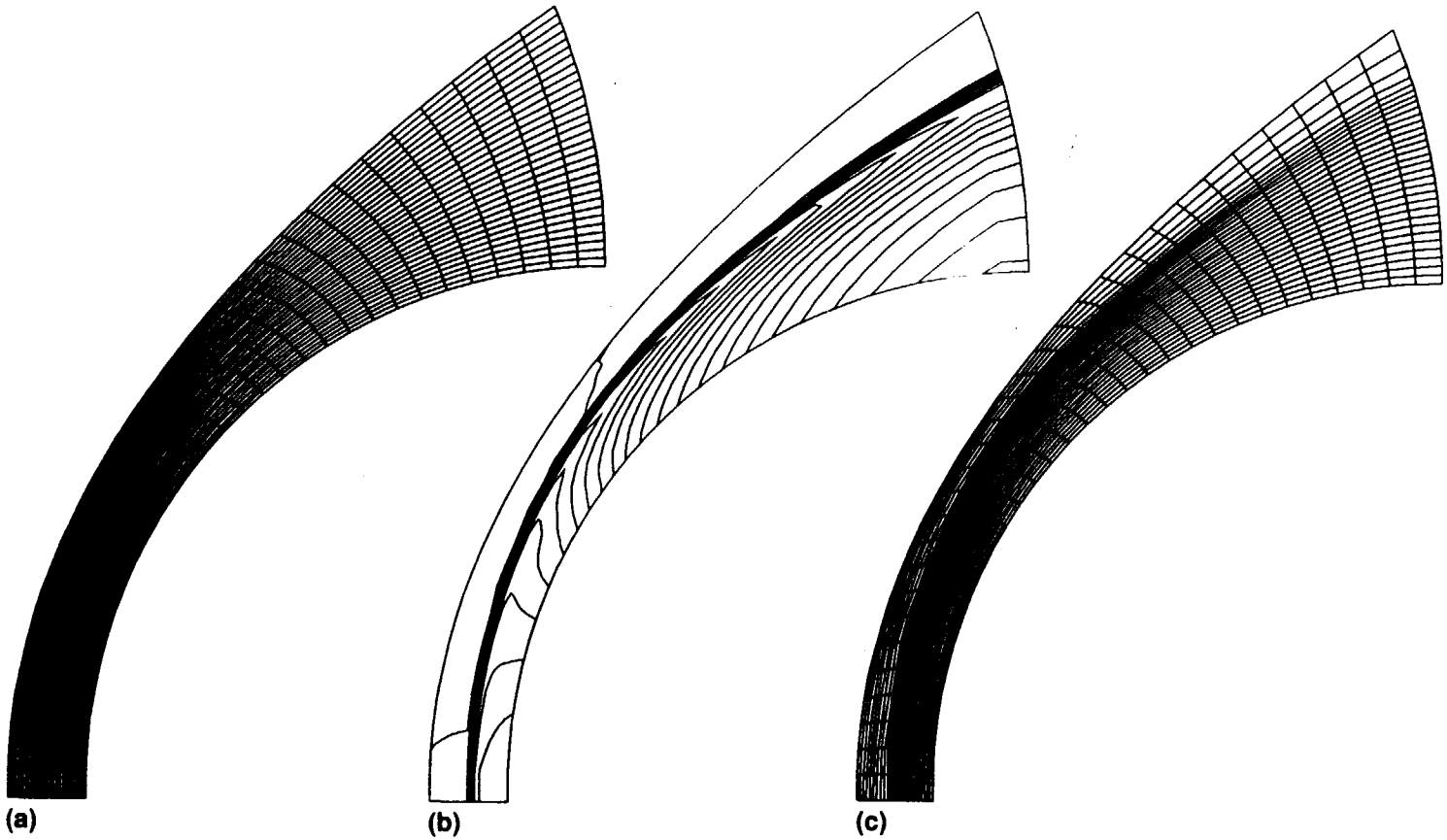


Figure 17. Hypersonic blunt body. (a) Initial grid; (b) initial flow solution, shown as density contours; (c) adapted grid, $\lambda = .001$.

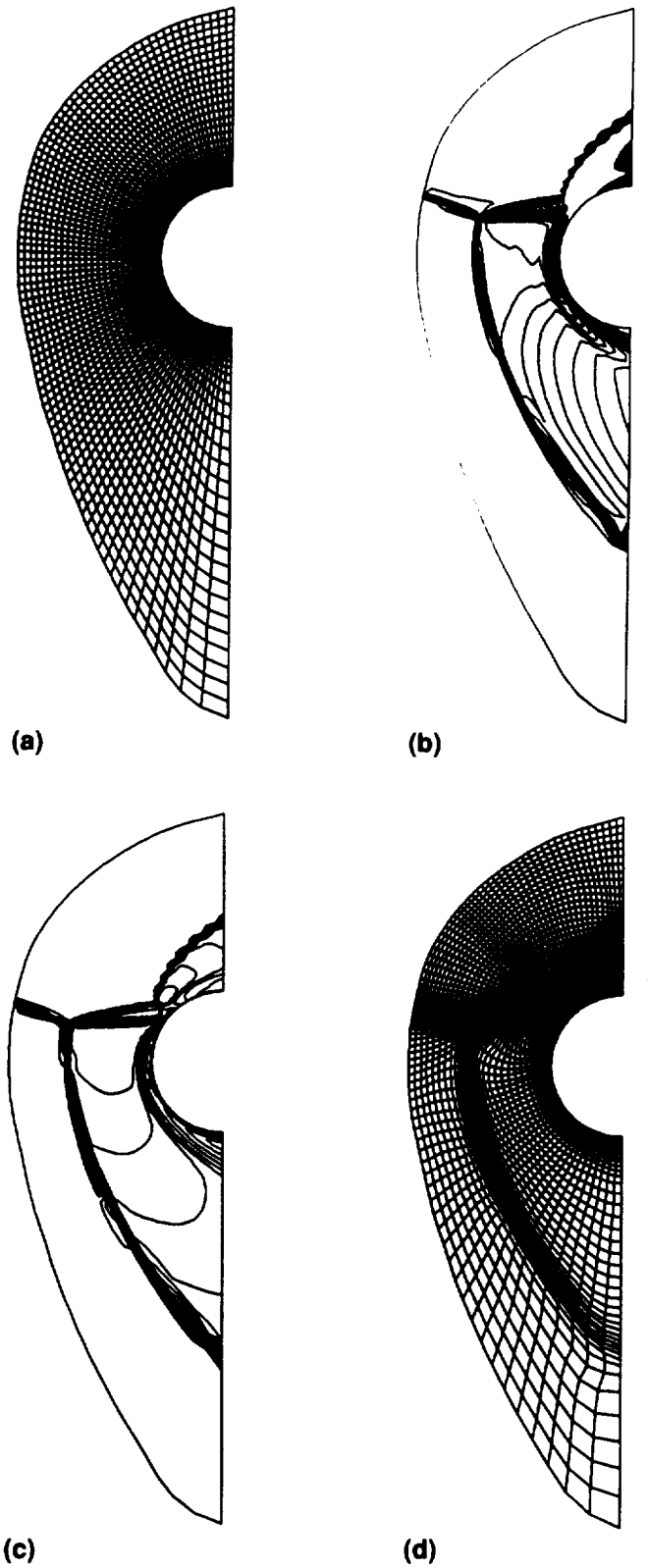


Figure 18. Blunt-body shock impingement problem. (a) Initial grid; (b) initial density contours; (c) initial Mach contours; (d) adapted grid.

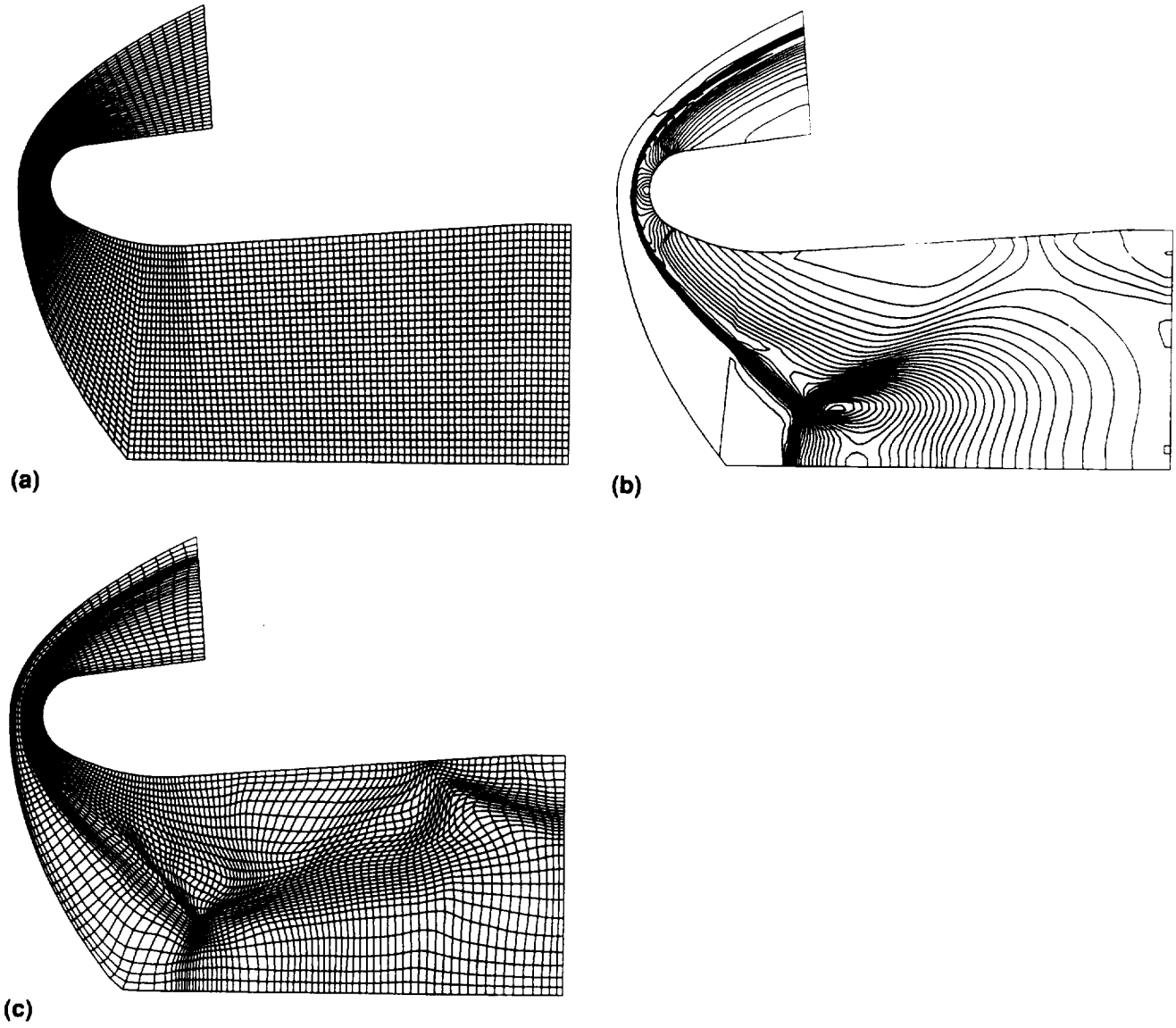


Figure 19. Hypersonic inlet, zonal adaption. (a) Initial grid; (b) initial density contours; (c) adapted grid, using five adaption passes.

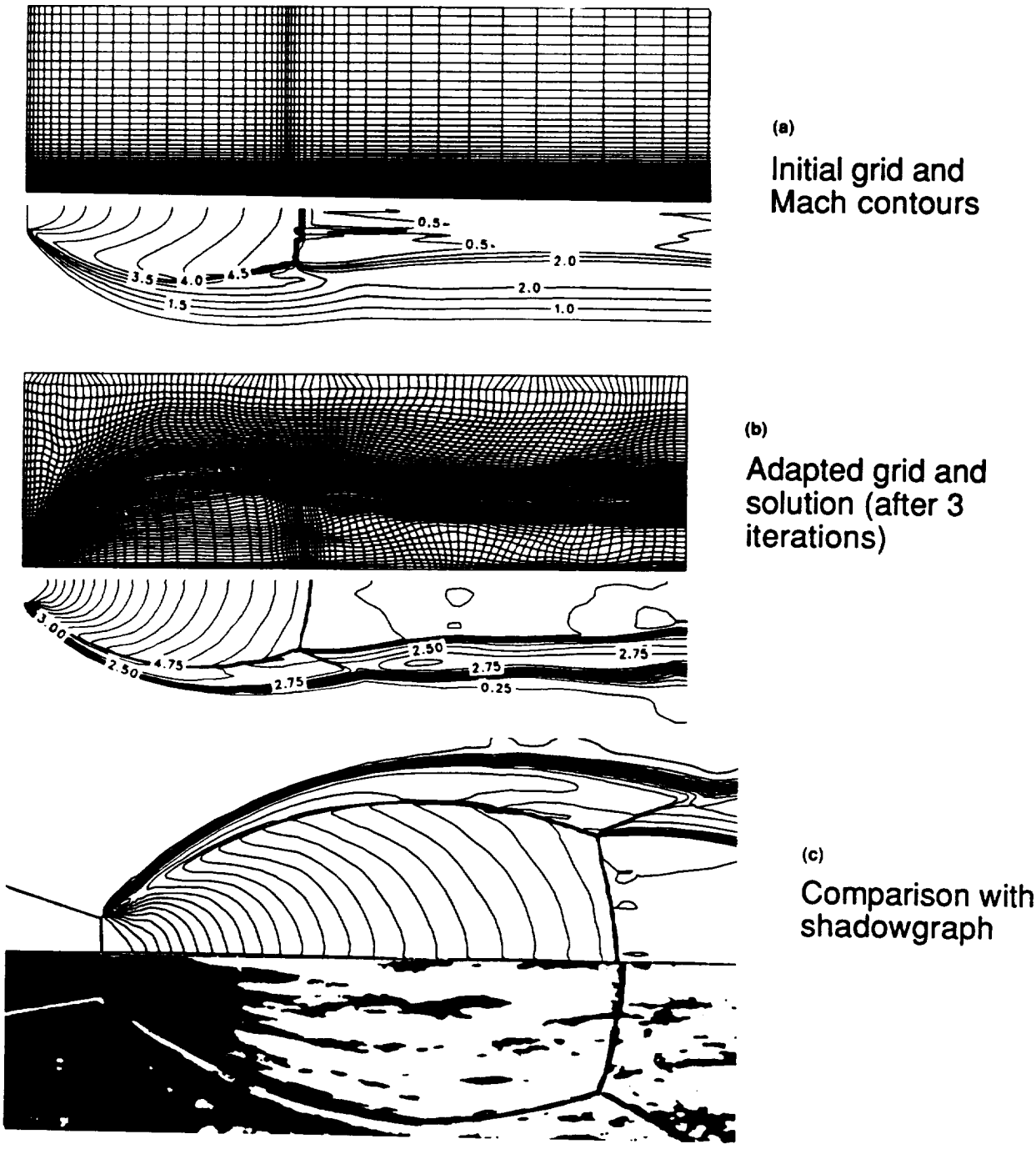
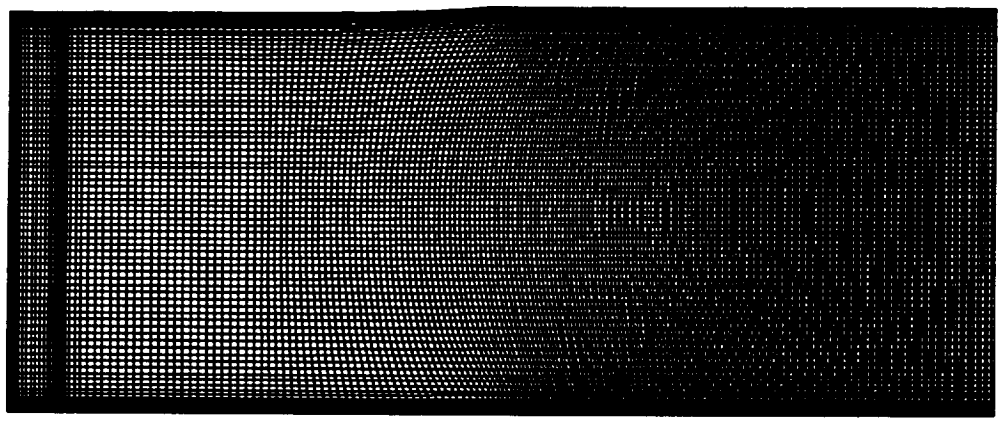
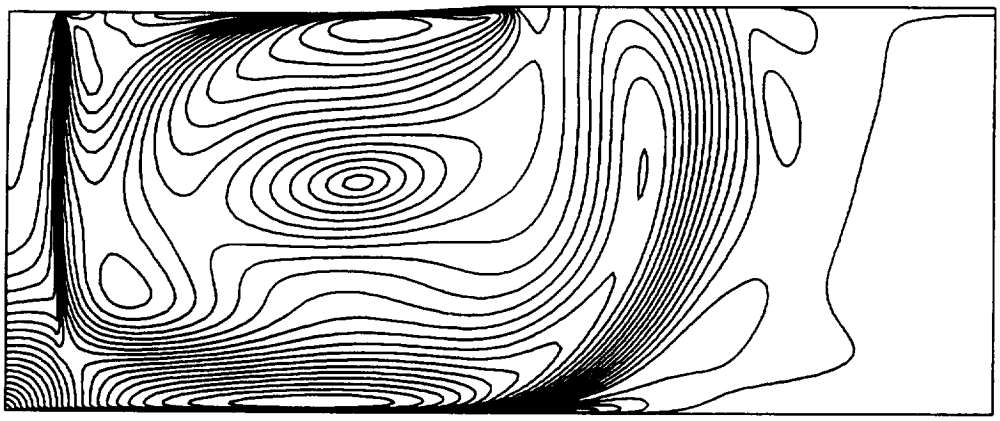


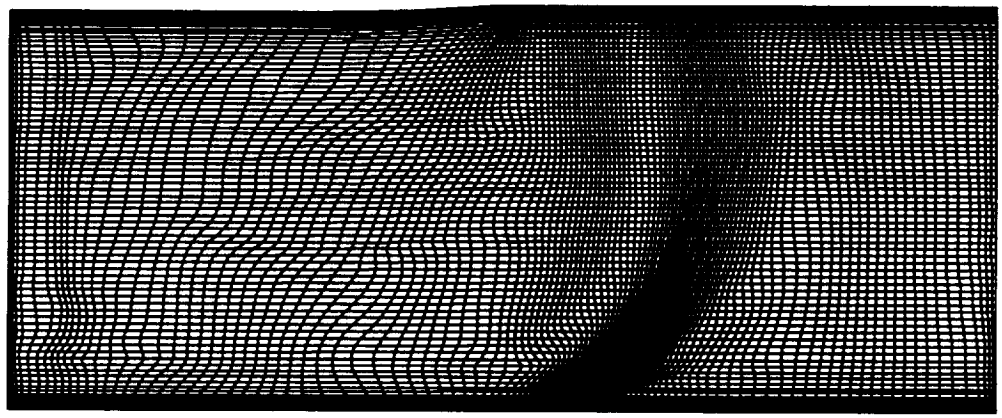
Figure 20. Axisymmetric plume flow. (a) Initial grid and solution showing undeveloped flow features; (b) final adapted grid and flow solution after several iterations; (c) comparison of computed solution with experimental shadowgraph.



(a)



(b)



(c)

Figure 21. Subsonic impinging jet. (a) Initial grid; (b) initial Mach contours; (c) adapted grid, with reduced grid points.

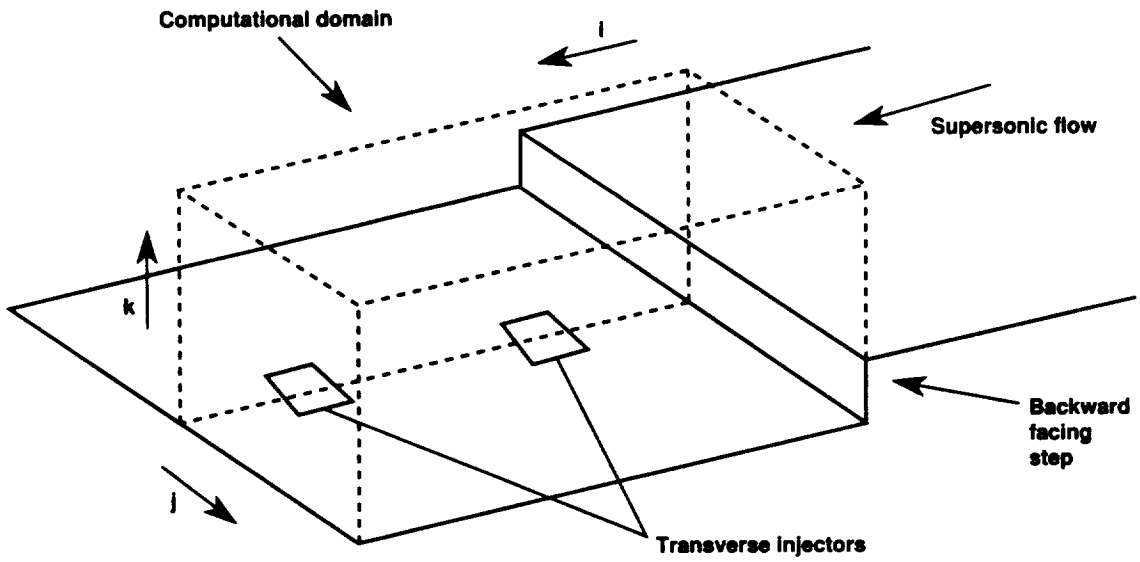


Figure 22. Staged transverse injectors in a supersonic combustor, showing computational grid.

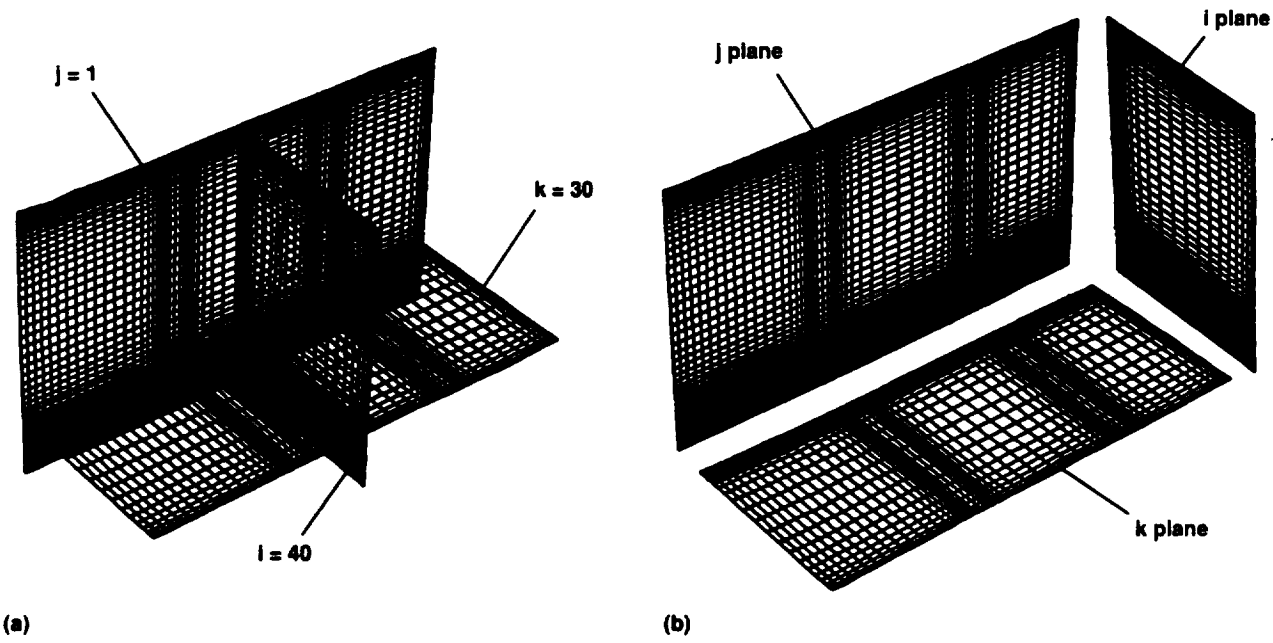


Figure 23. Three planes from initial grid. (a) Actual location, $i = 40, j = 1, k = 30$; (b) same planes, separated for clarity.

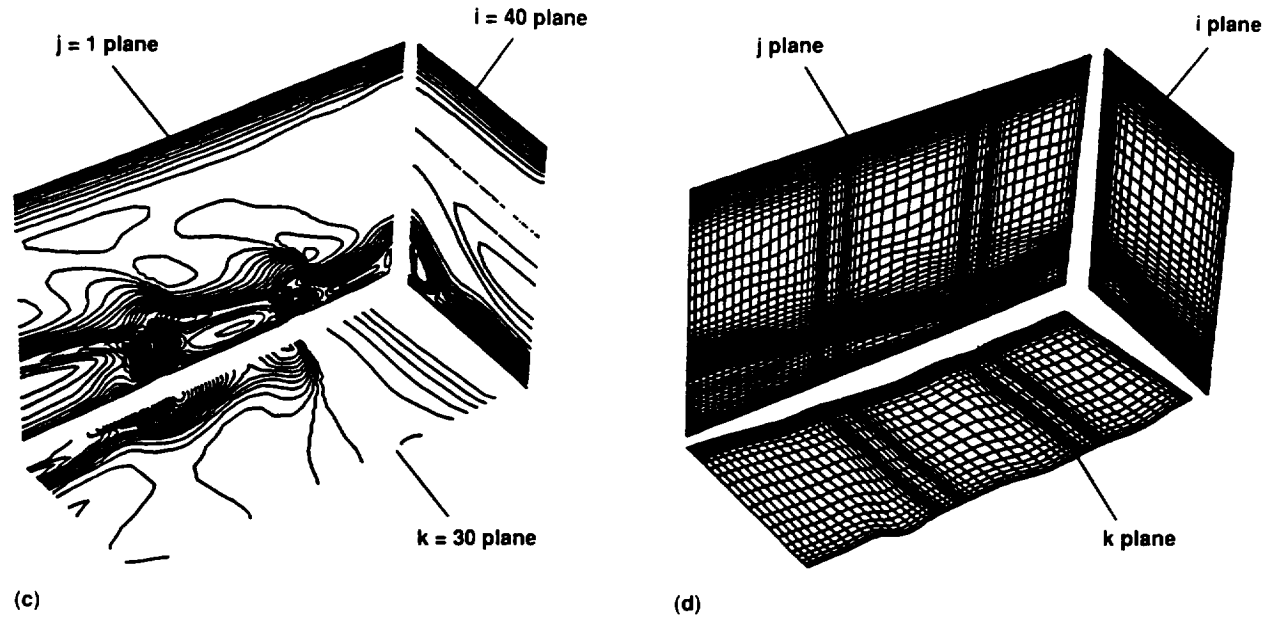


Figure 23. Concluded. (c) Initial Mach contours; (d) single direction adaption, adapting ik planes, stepping in i direction within the plane.

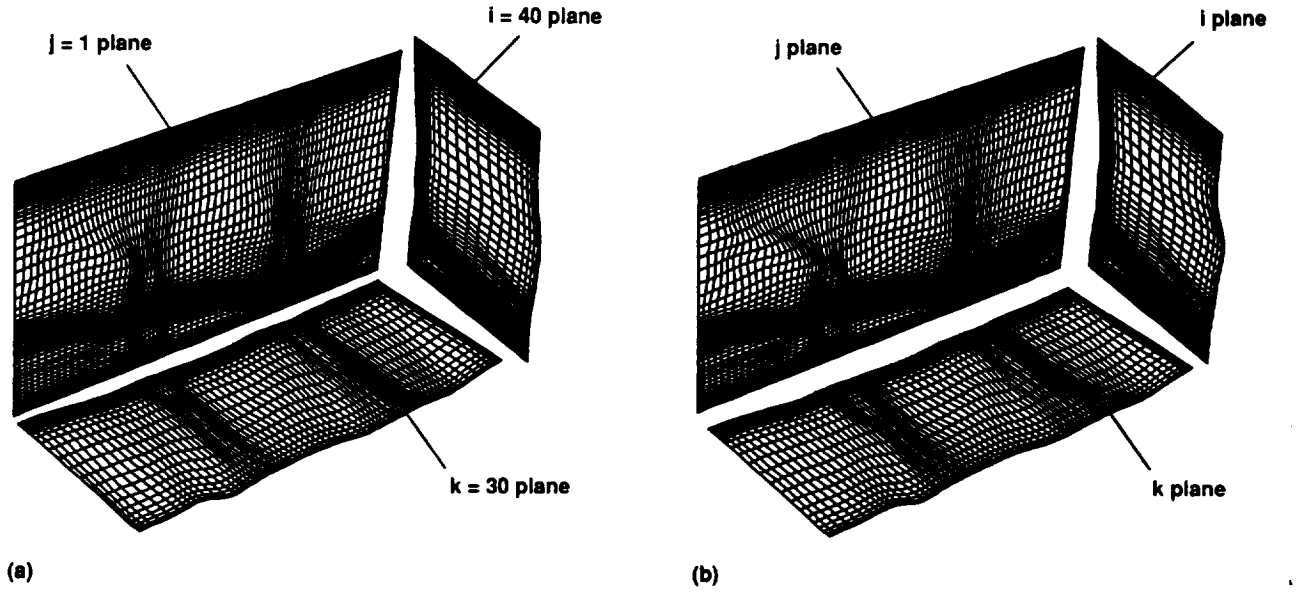


Figure 24. Two-directional adaption, using grid in Fig. 23(d) as input. (a) adapting ik planes, stepping in k lines; (b) adapting ij planes, stepping in j.

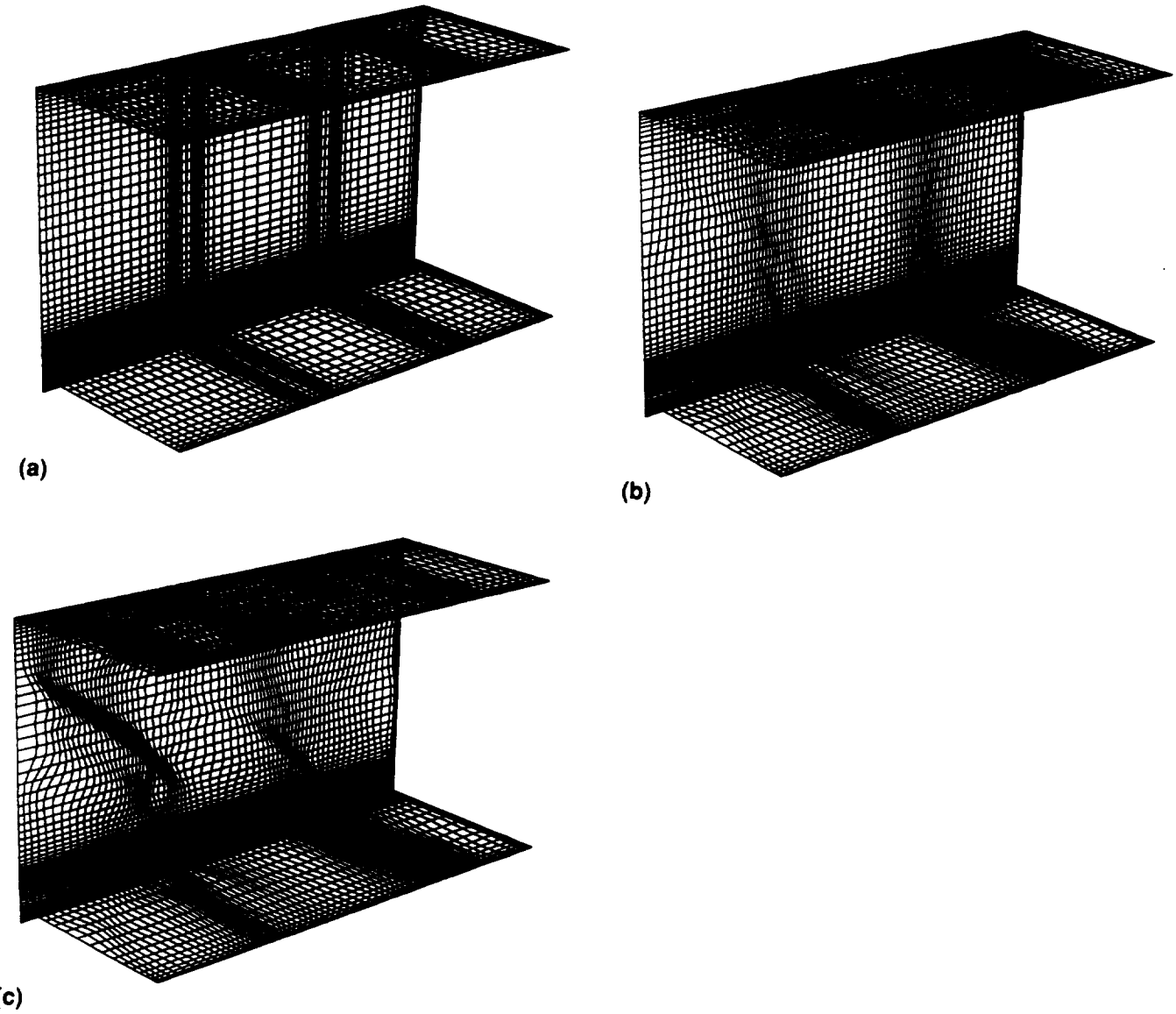


Figure 25. Effect of varying λ^*. (a) Initial grid, showing $j = 1$ plane; (b) adaption with $\lambda^* = .001$; (c) adaption with $\lambda^* = .0001$.

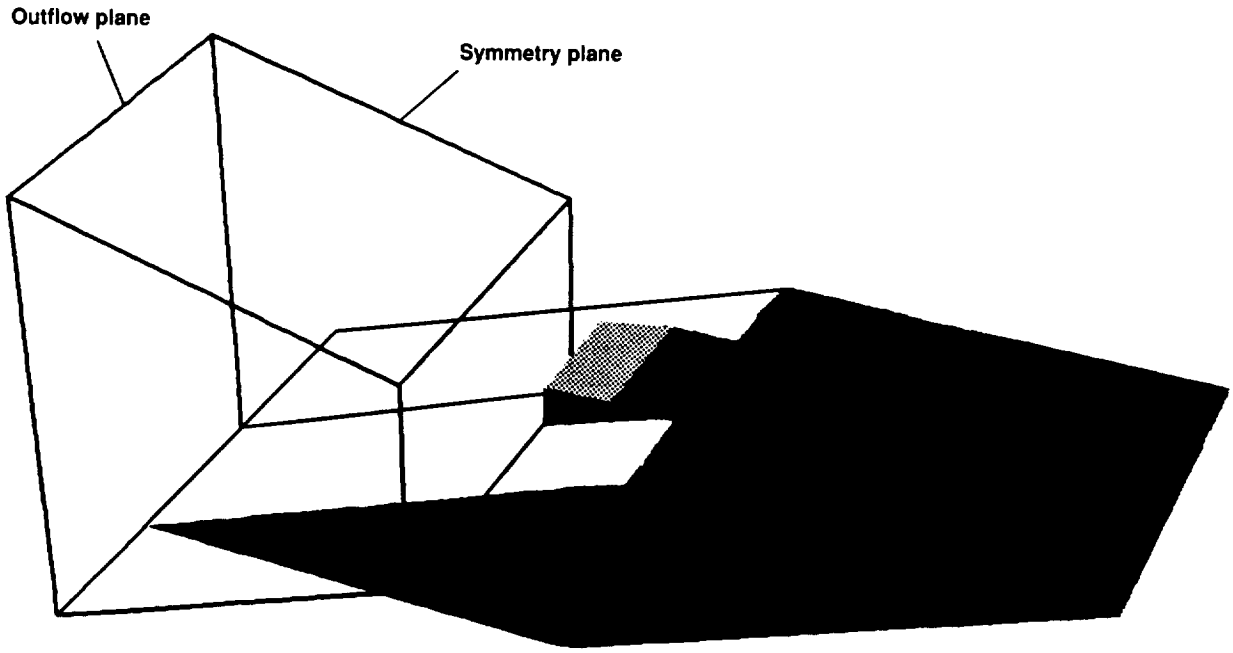


Figure 26. SERN experimental model and the computational space in the plume region.

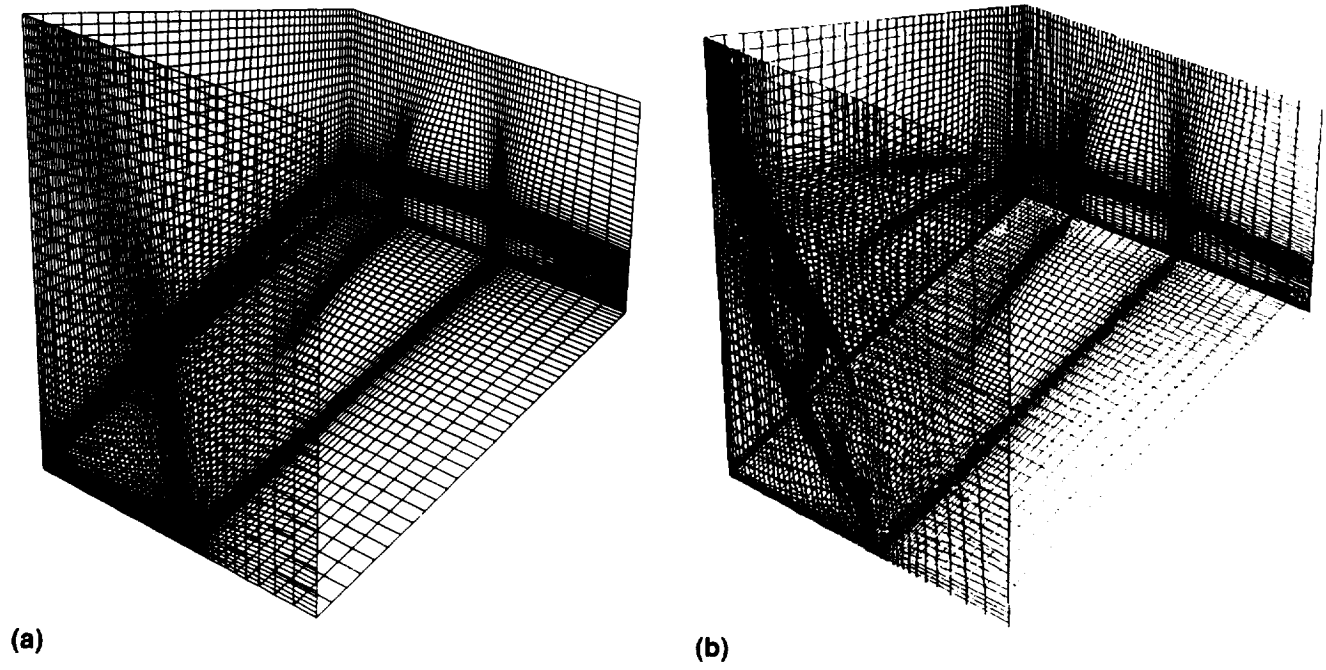


Figure 27. 3-D grid for SERN nozzle/afterbody. (a) Initial planes; (b) adapted planes.

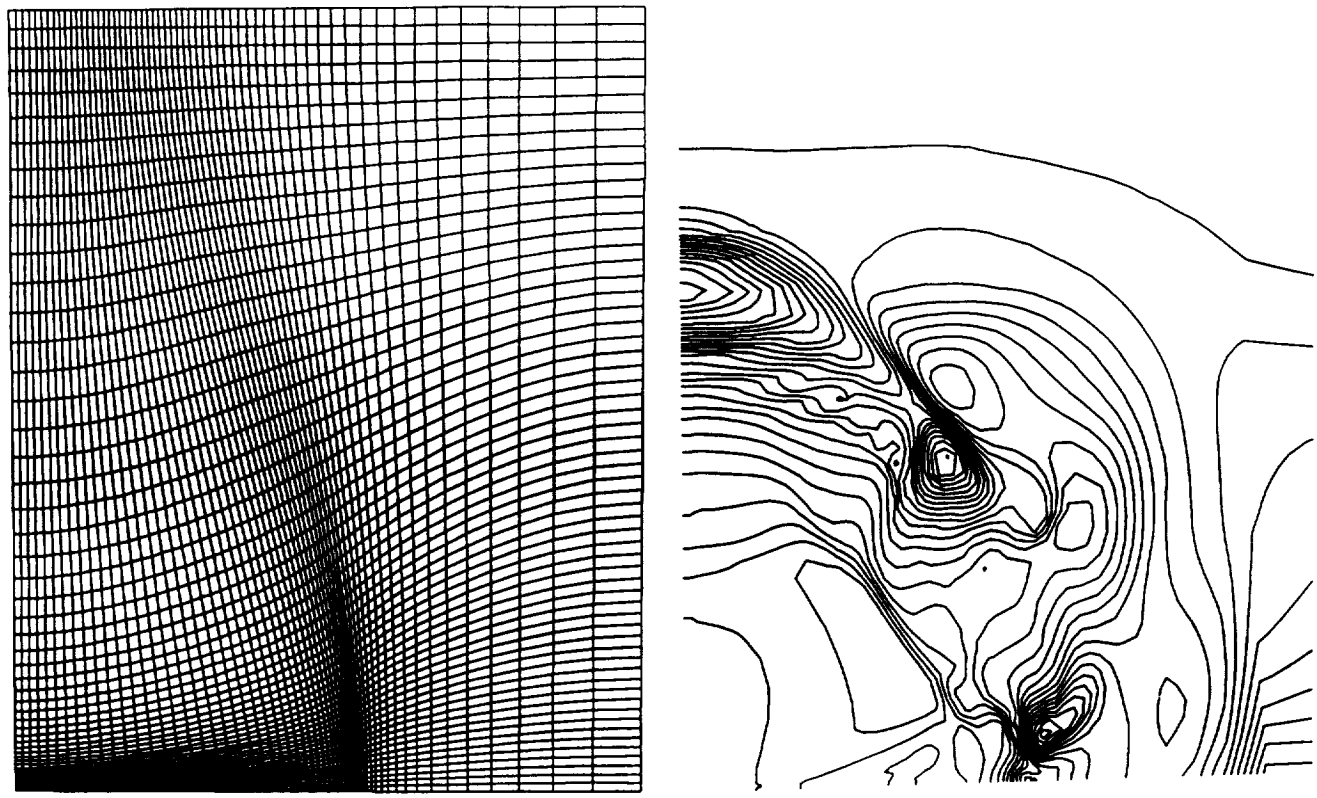
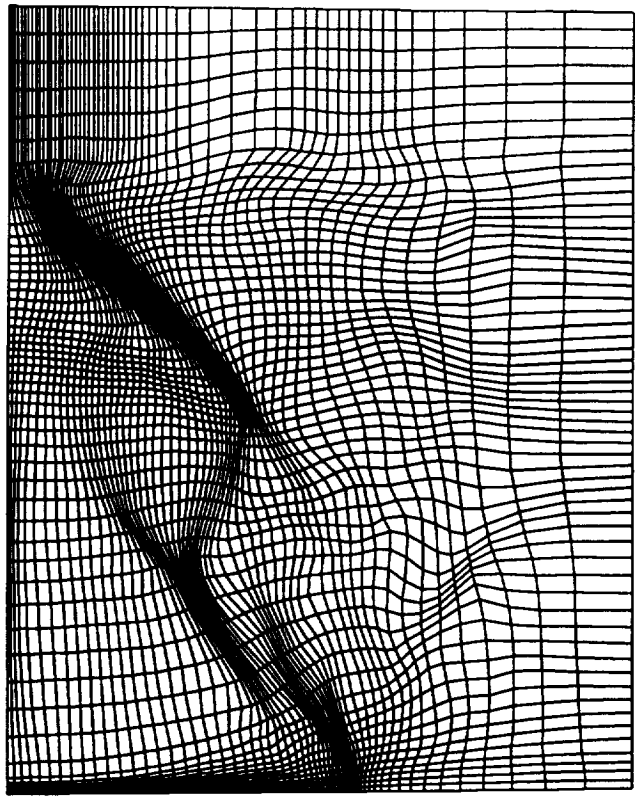


Figure 28. Downstream outflow plane ($i = 41$ in Fig. 27(a)); (a) Initial grid; (b) initial pressure gradients.



(c)

Figure 28. Concluded. (c) Adapted grid on outflow plane.

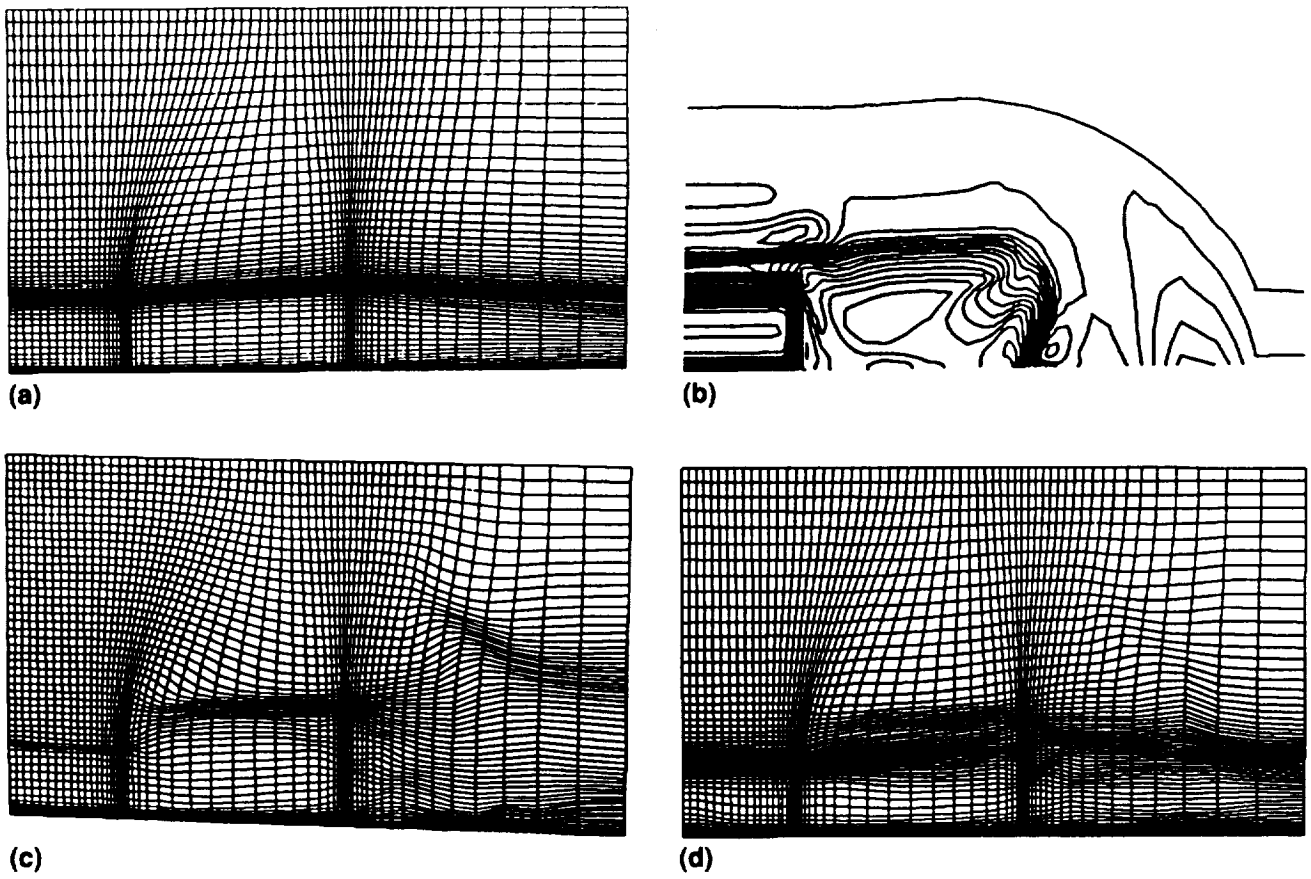


Figure 29. Example of merge planes parameter, mgpls. (a) Initial grid at $i = 1$ (and duplicated at $i = 2$); (b) initial pressure contours; (c) adapted grid at $i = 2$ with $ist = 2$; (d) adapted grid at $i = 2$ with $ist = 1$ and $mgpls = 4$.

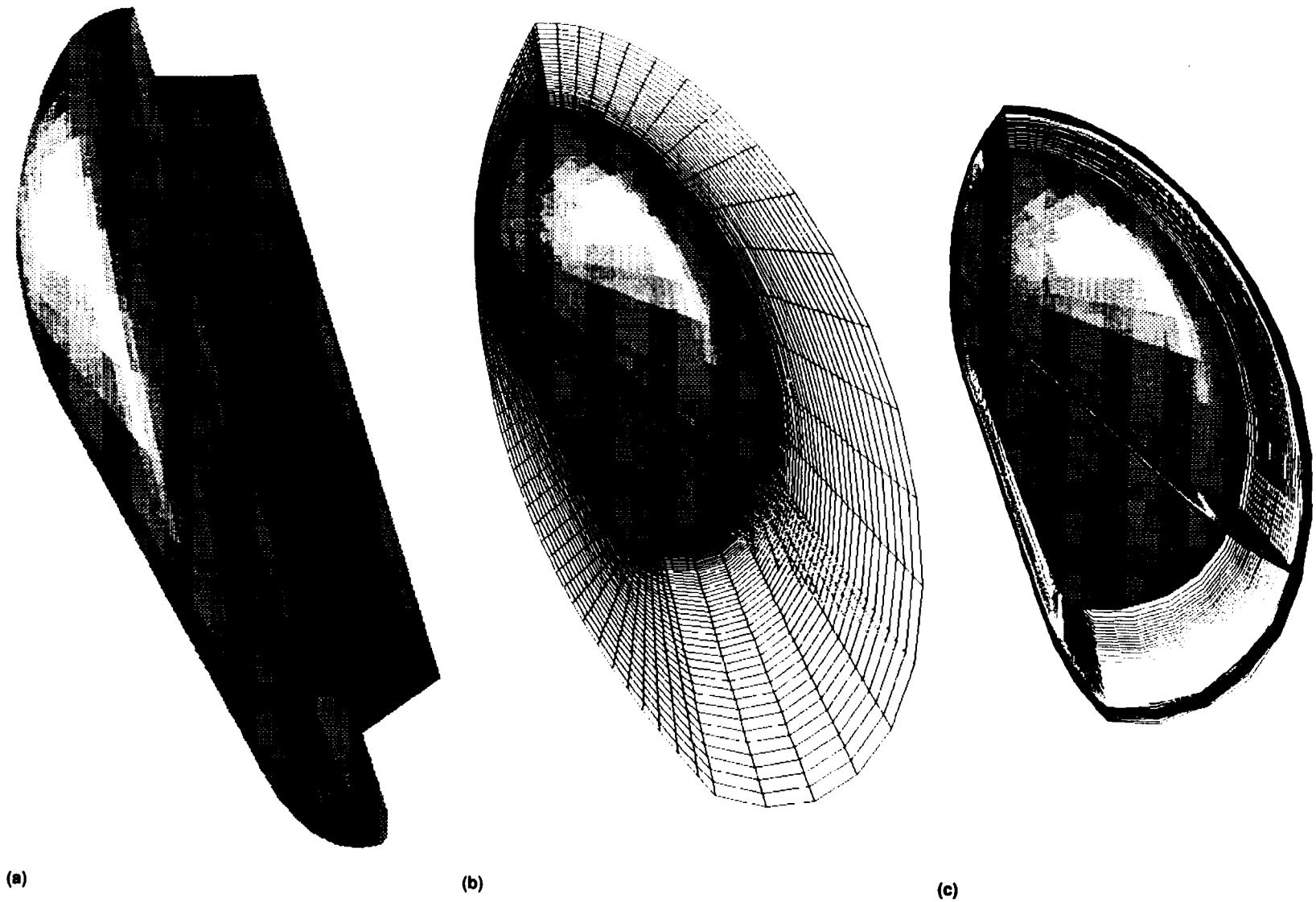


Figure 30. Hypersonic forebody flow. (a) Side view of vehicle geometry; (b) nonadapted grid around the forebody; (c) initial pressure contours.

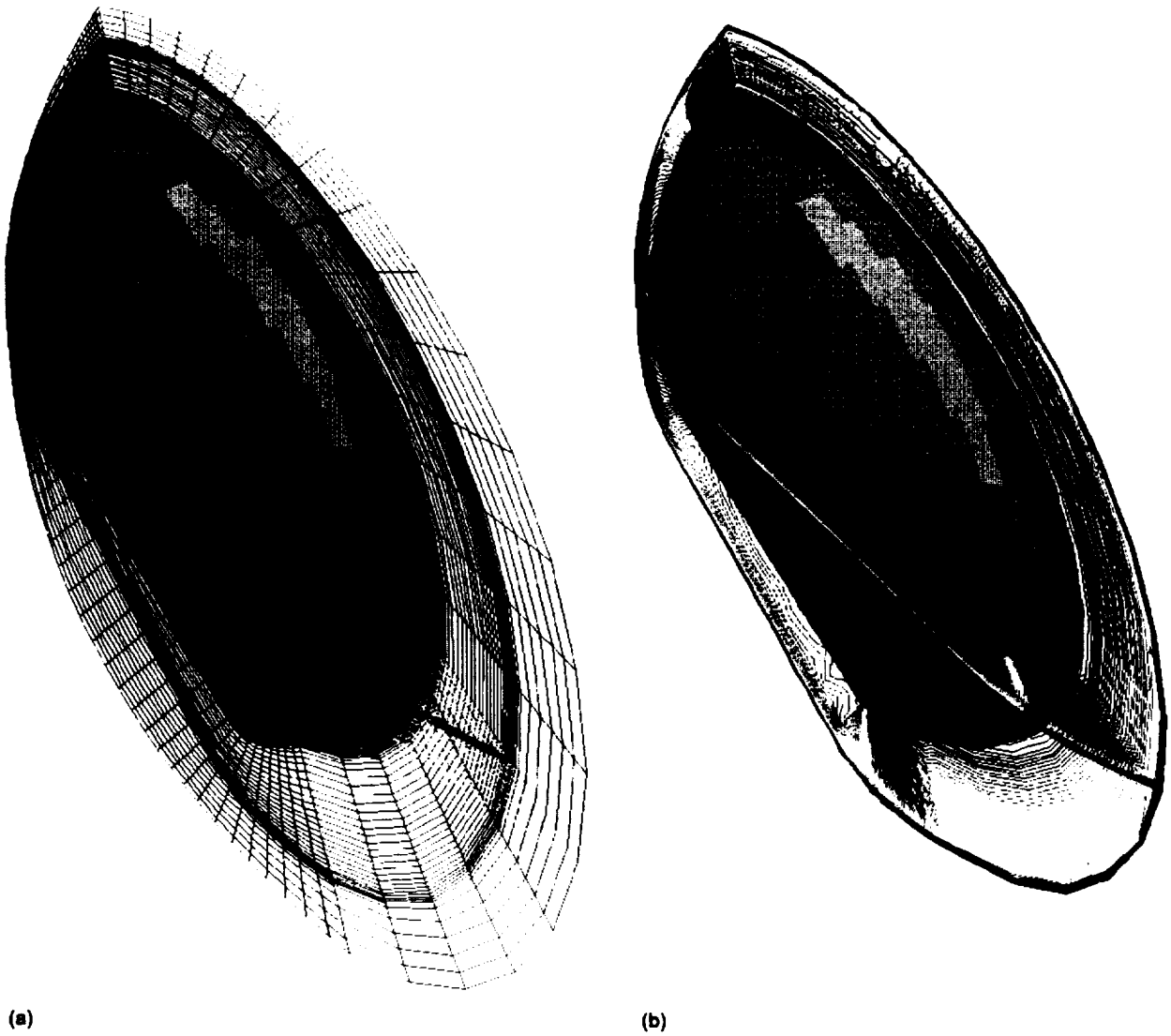


Figure 31. Adapted forebody flow. (a) Adapted grid; (b) pressure contours computed on adapted grid.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1992	Technical Memorandum

4. TITLE AND SUBTITLE
The Multidimensional Self-Adaptive Grid Code, SAGE

5. FUNDING NUMBERS
506-40

6. AUTHOR(S)
Carol B. Davies (Sterling Software, Palo Alto, CA) and
Ethiraj Venkatapathy (Eloret Institute, Sunnyvale, CA)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER
A-92019

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TM-103905

11. SUPPLEMENTARY NOTES
Point of Contact: Carol Davies, Ames Research Center, MS 230-2, Moffett Field, CA 94035-1000
(415) 604-6204

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified — Unlimited
Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

This report describes the multidimensional self-adaptive grid code SAGE. A two-dimensional version of this code was described in an earlier report by the authors. The formulation of the multidimensional version is described in the first section of this document. The second section is presented in the form of a user guide that explains the input and execution of the code and provides many examples. Successful application of the SAGE code in both two and three dimensions for the solution of various flow problems has proven the code to be robust, portable, and simple to use. Although the basic formulation follows the method of Nakahashi and Deiwert, many modifications have been made to facilitate the use of the self-adaptive grid method for complex grid structures. Modifications to the method and the simplified input options make this a flexible and user-friendly code. The new SAGE code can accommodate both two-dimensional and three-dimensional flow problems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Grids, Adaption, Refinement	73
	16. PRICE CODE: A04

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		