AIAA JOURNAL
Vol. 32, No. 6, June 1994

# Time Integration Algorithms for the Two-Dimensional Euler Equations on Unstructured Meshes

David C. Slack*
*AeroSoft, Inc., Blacksburg, Virginia 24060*
D. L. Whitaker†
*Analytical Services and Materials, Inc., Hampton, Virginia 23666*
and
Robert W. Walters‡
*Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060*

**Explicit and implicit time integration algorithms for the two-dimensional Euler equations on unstructured grids are presented. Both cell-centered and cell-vertex finite volume upwind schemes utilizing Roe's approximate Riemann solver are developed. For the cell-vertex scheme, a four-stage Runge-Kutta time integration, a four-stage Runge-Kutta time integration with implicit residual averaging, a point Jacobi method, a symmetric point Gauss-Seidel method, and two methods utilizing preconditioned sparse matrix solvers are presented. For the cell-centered scheme, a Runge-Kutta scheme, an implicit tridiagonal relaxation scheme modeled after line Gauss-Seidel, a fully implicit lower-upper (LU) decomposition, and a hybrid scheme utilizing both Runge-Kutta and LU methods are presented. A reverse Cuthill-McKee renumbering scheme is employed for the direct solver to decrease CPU time by reducing the fill of the Jacobian matrix. A comparison of the various time integration schemes is made for both first-order and higher order accurate solutions using several mesh sizes. higher order accuracy is achieved by using multidimensional monotone linear reconstruction procedures. The results obtained for a transonic flow over a circular arc suggest that the preconditioned sparse matrix solvers perform better than the other methods as the number of elements in the mesh increases.**

## Introduction

ALGORITHMS for solving the Euler equations using a perfect gas model on structured grids in two and three dimensions have become widespread in recent years.[1,2] However, these algorithms have shown difficulties in predicting satisfactory results around complex geometries due to mesh irregularities. As a result, attention has turned to the development of solution algorithms on arbitrary unstructured grids. Impressive results have been obtained for a wide range of problems.[3,4] The superior flexibility of unstructured over structured grids is demonstrated in Fig. 1 for a relatively simple body. A typical cross section of the SR-71 reconnaissance aircraft is shown in which the left half has been discretized by a structured mesh technique, and the right half has been discretized by an unstructured grid technique. The cells in the structured mesh are highly skewed, whereas the cell distribution in the unstructured mesh is quite smooth. The highly skewed mesh will lead to solution inaccuracies.

One problem associated with unstructured meshes is the increased difficulty in obtaining smooth higher order spatial approximations to state data at cell interfaces. Two methods have been used to obtain higher order accuracy on unstructured meshes. A method used by several researchers for cell-vertex schemes (Stoufflet et al.[5] and Whitaker[6]) was applied to obtain higher order accuracy in a method analogous to MUSCL differencing on a structured mesh. A conventional structured mesh limiter can be employed in this scheme to obtain near monotone results near flow discontinuities. The second method, which was proposed by Barth and Jespersen,[3] linearly reconstructs the cell-averaged data and imposes a monotone preserving limiter to achieve smooth results near flow discontinuities.

A major hurdle remaining for unstructured algorithms is the increased computational cost of these methods over those incurred by existing structured mesh solvers. Many current unstructured algorithms utilize explicit time integration to integrate to the steady state, which can be computationally expensive. Mavriplis and Jameson[7] have implemented multigrid techniques in an attempt to accelerate convergence on unstructured meshes. Other authors, including Wigton et al.[8] and Bender and Khosla,[9] have investigated sparse matrix solution algorithms on structured

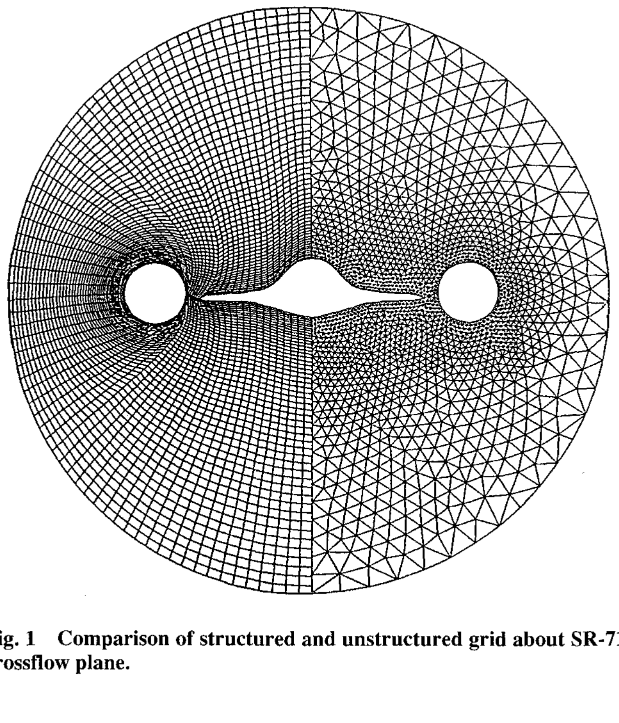
**Fig. 1 Comparison of structured and unstructured grid about SR-71 crossflow plane.**

Presented as Paper 90-0697 at the AIAA 28th Aerospace Sciences Meeting, Reno, NV, Jan. 8–11, 1990; received Aug. 8, 1991; revision received Nov. 18, 1992; accepted for publication Nov. 22, 1992. Copyright © 1990 by D. Slack, D. Whitaker, and R. Walters. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

*Research Scientist. Member AIAA.
†Research Engineer. Member AIAA.
‡Associate Professor, Department of Aerospace and Ocean Engineering. Senior Member AIAA.

meshes. The purpose of this paper is to extend the implicit techniques historically employed by structured grid algorithms to unstructured solvers to accelerate convergence rates. In the following sections a comparison is made among nine time integration algorithms: a four-stage Runge-Kutta explicit time integration, a four-stage Runge-Kutta explicit time integration scheme with implicit averaging of the residuals, a point Jacobi method, a point Gauss-Seidel method, two preconditioned sparse matrix solvers, a tridiagonal implicit relaxation (similar to line Gauss-Seidel for structured meshes), a fully implicit lower-upper (LU) decomposition, and a hybrid scheme that utilizes both Runge-Kutta and LU methods. All of these methods are implemented using a finite volume upwind scheme containing Roe's approximate Riemann solver. Results are given for both cell-centered and cell-vertex discretizations. Only Runge-Kutta explicit time integration is common to both codes. The cell-centered code uses tridiagonal implicit relaxation, a fully implicit LU decomposition, and a hybrid scheme. The cell-vertex code uses the residual averaging with Runge-Kutta, the point Jacobi method, the point Gauss-Seidel method, and the two preconditioned sparse matrix solvers.

## Mesh Generation

The procedure used for generating triangular elements about an arbitrary configuration is an advancing front method discussed by Löhner.[10] This technique requires input of certain stretching parameters given in the context of a coarse background grid of triangular elements that covers the solution domain. These parameters are first used to place nodes along the boundaries of the computational region. The sides connecting these nodes form the initial generation front. Elements are added by interpolating the stretching parameters from the background grid, the entire front is updated, and the process is repeated. Once the entire domain has been triangulated, a smoothing routine is used to improve the quality of the generated mesh.

## Spatial Discretization of the Governing Equations

The two-dimensional Euler equations in integral form can be written as

$$\frac{\partial}{\partial t}\iint_{\Omega} Q \, d\Omega + \oint_{S} \boldsymbol{F} \cdot \hat{n} \, ds = 0 \tag{1}$$

with

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_0 \end{bmatrix} \tag{2}$$

$$\boldsymbol{F} = f\hat{i} + g\hat{j} \tag{3}$$

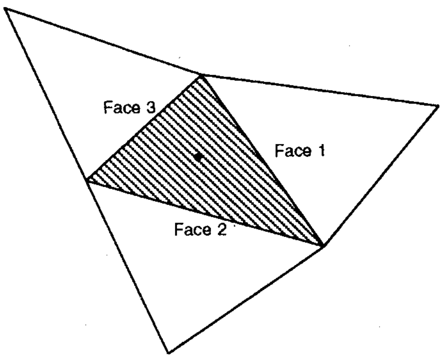

**Fig. 2 Typical control volume for cell-centered case.**

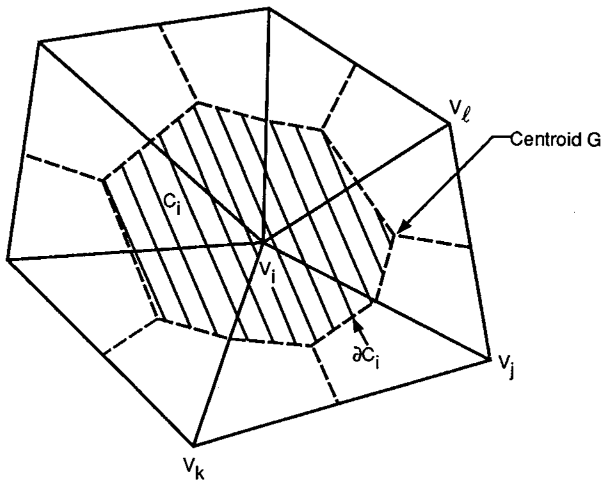

**Fig. 3 Typical control volume for cell-vertex case.**

and

$$f = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho u h_0 \end{bmatrix}, \qquad g = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho v h_0 \end{bmatrix} \tag{4}$$

where $\Omega$ is the area of the domain, and $S$ surrounds $\Omega$. The density is $\rho$, the velocity vector is $\boldsymbol{U} = (u\hat{i} + v\hat{j})$, the total energy per unit mass is $e_0$, the total enthalpy per unit mass is $h_0$, the pressure is $p$, and the outward facing unit normal is $\hat{n}$. The system of equations is closed by the equation of state for a perfect gas:

$$p = \rho(\gamma - 1)\left[e_0 - \frac{(u^2 + v^2)}{2}\right]$$

where $\gamma$ is the ratio of specific heats (typically taken as 1.4 for air). The density is nondimensionalized by $\rho_\infty$, velocity by $|\boldsymbol{U}_\infty| = \sqrt{u_\infty^2 + v_\infty^2}$, and the pressure and energy per unit volume by $\rho_\infty (\boldsymbol{U}_\infty \cdot \boldsymbol{U}_\infty)$.

Both cell-centered and cell-vertex finite volume forms of discretization are explored. The cell-centered method (as shown in Fig. 2) stores the conserved variables at the centroid of each triangle, and the edges of a triangle define the faces of the control volume. The cell-vertex scheme stores the conserved variables at the vertices of the triangles, and the faces of the control volume are formed by connecting the centroids of the triangles surrounding each vertex to the midpoints of the edges emanating from each vertex, as shown in Fig. 3. Each vertex then becomes the approximate cell center of the control volume created around it. Equation (1) can be rewritten in the form

$$V(C_i)\frac{\partial Q}{\partial t} = -\oint_{\partial C_i} \boldsymbol{F} \cdot \hat{n} \, ds \tag{5}$$

where $C_i$ denotes a cell or control volume, and $V(C_i)$ is the area of $C_i$.

To evaluate the right-hand side of Eq. (5), we sum the flux vectors at each face of $C_i$. In the cell-centered case, the number of cell faces is always three, but for the cell-vertex case the number of cell faces varies. For a particular vertex it is equal to the number of its neighboring vertices. The outward facing normal of the face in the cell vertex scheme is found as an area-weighted average of the sides defining the control volume face. For both methods, the resulting flux integral can be written in discrete form as

$$\oint_{\partial C_i} \boldsymbol{F} \cdot \hat{n} \, ds = \sum_{j = \kappa(i)} F_{i,j} \Delta s \tag{6}$$

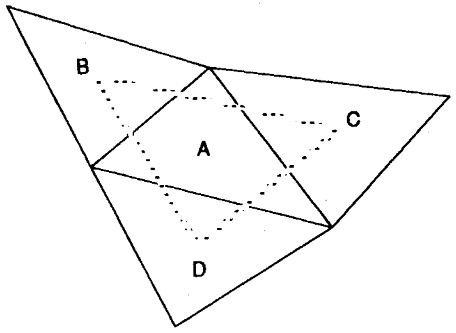

**Fig. 4 Integration path for gradient calculation of cell-centered case.**

where $F_{i,j}$ is the numerical approximation for the flux at each cell face, $\Delta s$ is the length of each cell face, and $\kappa(i)$ is a list of neighboring vertices (cells).

To apply Roe's characteristic-based flux difference splitting scheme,[11] it is first necessary to obtain two discrete fluid dynamic states separated by a cell interface. In the approach taken here, the state variables are interpolated from the cell centers to the cell faces, creating a left and right state for each interior face. For first-order accuracy, a Taylor's series approximation to the left and right fluid states is simply

$$Q_L = Q(C_i)$$

$$Q_R = Q(C_j)$$

where the subscripts $R$ and $L$ indicate right and left fluid states. Roe's upwind flux differencing is then applied in the form

$$F_{i,j} = 1/2[F(Q_L) + F(Q_R) - |\tilde{A}| (Q_R - Q_L)]$$

where $|\tilde{A}|$ is the Roe-averaged matrix. The Roe-averaged matrix is constructed so that a flux difference between two fluid states is satisfied identically by

$$F(Q_R) - F(Q_L) = \tilde{A} (Q_R - Q_L)$$

The absolute value symbols placed around the matrix simply indicate that the absolute values of the eigenvalues of $\tilde{A}$ were used to evaluate $\tilde{A}$.

## Cell-Centered Higher Order Correction

A piecewise linear redistribution of the cell-averaged flow variables is given by Barth and Jespersen[3] as

$$Q(x, y) = Q(x_0, y_0) + \nabla Q \cdot \boldsymbol{r} \tag{7}$$

where $\boldsymbol{r}$ is the vector from the cell center $(x_0, y_0)$ to any point $(x, y)$ in the cell, and $\nabla Q$ represents the solution gradient in the cell. Note that this equation is simply the first-order accurate Taylor approximation plus a higher order correction. With this approximation, the solution gradient $\nabla Q$ is constant in each cell and can be computed from

$$\nabla Q_A = \frac{1}{S_\Omega} \oint_\Omega Q \hat{n} \, d\Omega \tag{8}$$

where $S_\Omega$ is the area contained in the path of integration. For the cell-centered case, the path chosen passes through the centroids of the three surrounding cells $B$, $C$, and $D$ of the given cell $A$, as shown in Fig. 4.

Consider a limited version of the linear function about the centroid of cell $A$

$$Q(x, y)_A = Q(x_0, y_0)_A + \Phi_A \nabla Q_A \cdot \boldsymbol{r}_A, \qquad \Phi_A \, \varepsilon \, [0, 1] \tag{9}$$

To find the value of $\Phi_A$, a monotonicity principle is enforced on the unlimited quantities $Q_{A_i} = Q(x_i, y_i)_A$ calculated in Eq. (7) at the faces of cell $A$. It requires that the values computed at the faces must not exceed the maximum and minimum of neighboring cell values, including the value in cell $A$, i.e., that

$$Q_A^{\min} \le Q_{A_i} \le Q_A^{\max} \tag{10}$$

where $Q_A^{\min} = \min(Q_A, Q_{\text{neighbors}})$ and $Q_A^{\max} = \max(Q_A, Q_{\text{neighbors}})$. The value $\Phi_A$ can now be calculated for each face $i$ of cell $A$ as

$$\Phi_{A_i} = \begin{cases} \min\left(1, \dfrac{Q_A^{\max} - Q_A}{Q_i - Q_A}\right), & \text{if} \quad Q_i - Q_A > 0 \\ \min\left(1, \dfrac{Q_A^{\min} - Q_A}{Q_i - Q_A}\right), & \text{if} \quad Q_i - Q_A < 0 \\ 1 & \text{if} \quad Q_i - Q_A = 0 \end{cases} \tag{11}$$

with $\Phi_A = \min(\Phi_{A_i})$, where $i$ is the index of the set of faces surrounding cell $A$. New limited values for $Q_{A_i}$ at each of the faces of cell $A$ are then calculated from Eq. (9) using the value of $\Phi_A$ calculated for the cell.

## Cell-Vertex Higher Order Correction

Alternatively, an approach shown in Fig. 5 can be used to evaluate $\nabla Q$ for the cell-vertex scheme.[6] For each edge surrounding vertex $V_i$ in cell $C_i$, a search can be made of the triangular elements surrounding nodes $V_i$ and $V_j$ that define each edge. Triangular elements that contain the line defined by the vectors $\boldsymbol{V}_{i'}\boldsymbol{V}_i$, and $V_jV_{j'}$ and surrounding nodes $V_i$ and $V_j$, respectively, are used to evaluate the gradient for higher order MUSCL differencing. Phantom nodes $V_{i'}$ and $V_{j'}$ define the points where data are necessary to evaluate a MUSCL differencing formula in a structured grid manner. higher order data are reconstructed at the midpoint of the edge defined by vertices $V_i$ and $V_j$. The MUSCL differencing formulas for Fig. 10, can be written as

$$\begin{aligned} (Q_L)_{i,j} &= Q_i + \frac{s}{4} [(1 - \kappa s)\Delta_- + (1 + \kappa s)\, \Delta_+ ]Q_i \\ (Q_R)_{i,j} &= Q_j - \frac{s}{4} [(1 - \kappa s)\Delta_+ + (1 + \kappa s)\, \Delta_- ]Q_j \end{aligned} \tag{12}$$

where

$$\begin{aligned} \Delta_-(Q_i) &= Q_i - Q_{i'} \qquad & \Delta_+(Q_i) &= Q_j - Q_i \\ \Delta_-(Q_j) &= Q_j - Q_i \qquad & \Delta_+(Q_j) &= Q_{j'} - Q_j \end{aligned} \tag{13}$$

where $s$ is Van Albada's limiter[12]

$$s = \frac{2\Delta_+\Delta_- + \delta}{(\Delta_+)^2 + (\Delta_-)^2 + \delta} \tag{14}$$

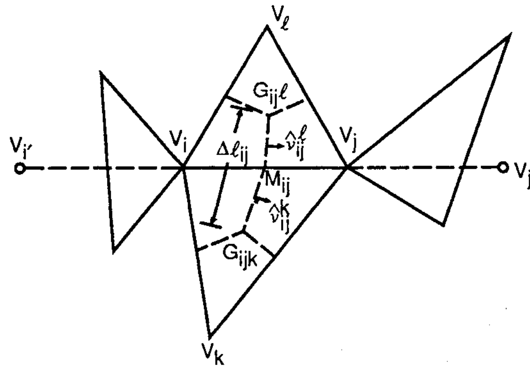

**Fig. 5 Integration path for gradient calculation of cell-vertex case.**

and $\delta$ is a small number used to prevent division by zero in regions of small gradient.

The limiter $s$ serves to limit the accuracy of the MUSCL-differencing formulas in regions of large gradient. This limiting causes a loss of accuracy in the solution but does insure that oscillations in the numerical solution near discontinuities are minimized or prevented. Van Albada's limiter was used because it acted in a continuously differentiable manner that tended to prevent limit cycles in the convergence history.

The parameter $\kappa$ controls the accuracy of the MUSCL differencing. With $\kappa = -1$, the solution is second-order accurate and fully upwinded. Using $\kappa = 1/3$, the solution is upwind biased. The backward and forward differences to imaginary points can be approximated by

$$\begin{aligned} \Delta_-(Q_i) &= \vec{\nabla} Q_i \cdot \boldsymbol{V}_{i'}\boldsymbol{V}_i = \vec{\nabla} Q_i \cdot \boldsymbol{V}_i\boldsymbol{V}_j \\ \Delta_+(Q_j) &= \vec{\nabla} Q_j \cdot \boldsymbol{V}_j\boldsymbol{V}_{j'} = \vec{\nabla} Q_j \cdot \boldsymbol{V}_i\boldsymbol{V}_j \end{aligned} \tag{15}$$

The notations $\vec{\nabla}Q_i$ and $\vec{\nabla}Q_j$ represent the gradient in the triangular element surrounding $V_i$ and containing vector $\boldsymbol{V}_{i'}\boldsymbol{V}_i$ and surrounding $V_j$ and containing vector $\boldsymbol{V}_j\boldsymbol{V}_{j'}$, respectively.

## Time Integration to Steady State

To obtain a steady-state solution, the spatially discretized Euler equations must be integrated in time. Nine time integration methods are compared: a four-stage explicit Runge-Kutta,[4] a four-stage Runge-Kutta with implicit residual smoothing, point Jacobi, point Gauss-Seidel, a line Gauss-Seidel type relaxation, two sparse matrix solvers, a fully implicit LU decomposition, and a hybrid scheme that combines Runge-Kutta and an LU decomposition.

The Runge-Kutta scheme can be written as

$$\begin{aligned} Q^{(0)} &= Q^{(n)} \\ Q^{(1)} &= Q^{(0)} + \alpha_1 \frac{\Delta t}{V} R\,[Q^{(0)}] \\ Q^{(2)} &= Q^{(0)} + \alpha_2 \frac{\Delta t}{V} R\,[Q^{(1)}] \\ Q^{(3)} &= Q^{(0)} + \alpha_3 \frac{\Delta t}{V} R\,[Q^{(2)}] \\ Q^{(4)} &= Q^{(0)} + \alpha_4 \frac{\Delta t}{V} R\,[Q^{(3)}] \\ Q^{(n+1)} &= Q^{(4)} \end{aligned} \tag{16}$$

where $R(Q)$ is the right-hand side of Eq. (5), $V$ is the cell area, and the coefficients used are

$$\alpha_1 = 0.15 \qquad \alpha_2 = 0.3275 \qquad \alpha_3 = 0.57 \qquad \alpha_4 = 1 \tag{17}$$

These weighting coefficients[2] were determined by numerical experiments to best accelerate convergence to the steady state in an upwind, structured code. Convergence to the steady state is also accelerated by using a local time-stepping technique in which the maximum permissible time step for each individual cell in the flowfield is used. A local stability analysis was used to determine the maximum permissible time step.

The Runge-Kutta scheme can be accelerated by applying implicit residual smoothing at every stage of the time integration. The residuals were smoothed for every stage of the Runge-Kutta solver. The following equation was solved for the new value of the residual:

$$\bar{R}_i = R_i + \varepsilon \sum_{j=\kappa(i)} (\bar{R}_j - \bar{R}_i) \tag{18}$$

The variable $\kappa(i)$ contains a list of vertices that surround vertex $V_i$ and are adjacent to $V_i$. The resulting implicit equation for $\bar{R}_i$ is solved by Jacobi iteration. Typically, two Jacobi iterations were performed with $\varepsilon = 0.5$. See Ref. 4 for further information.

The point Jacobi, point Gauss-Seidel, line Gauss-Seidel, and LU decomposition utilize the Euler implicit time integration of Eq. (5), which can be represented in delta form as

$$V\frac{\Delta Q}{\Delta t} = R^{n+1} \tag{19}$$

where $\Delta\{\} = \{\}^{n+1} - \{\}^n$ and $V$ is the cell area. The equation can be linearized as

$$V\frac{\Delta Q}{\Delta t} = R^n + \frac{\partial R^n}{\partial Q}\Delta Q$$

After linearization, this equation can be written as the linear system

$$A\Delta Q = R^n \tag{20}$$

where

$$A = \left(\frac{V}{\Delta t}I - \frac{\partial R^n}{\partial Q}\right) \tag{21}$$

For two-dimensional problems, $A$ is generally a block $4 \times 4$ matrix of dimension $n$ (the number of cells) with a variable bandwidth. The linear system can be an approximate or exact linearization of the residual $R$. Generally, the linearization of the residual is only approximate if Roe's approximate Riemann solver is used to evaluate the residual, due to the difficulty in linearizing Roe's scheme. Because of the approximate linearization and nonlinearity of the residual, it may be well justified to obtain only an approximate solution to the linear system.

For solving linear problems using classic relaxation schemes, it is convenient to express

$$A = M + D + N \tag{22}$$

where $M$ is a lower triangular matrix, $D$ is a diagonal matrix, and $N$ is an upper diagonal matrix. In this case all elements of $M$, $D$, and $N$ are block $4 \times 4$ matrices.

The point Jacobi method was used to compute an approximate solution to the linear system. The block $4 \times 4$ matrices on the diagonal were inverted and multiplied by the right-hand side:

$$\Delta Q = D^{-1}R^n \tag{23}$$

As can easily be seen, the method is actually a block Jacobi iteration with the blocks being $4 \times 4$ matrices formed from the linearization of the four coupled equations at each grid point. The vector $Q$ is updated after $\Delta Q$ is found, and a new residual vector and Jacobian are computed. In the point Jacobi method, it is not necessary to compute the off-diagonal terms in the $A$ matrix.

To implement the point Gauss-Seidel method, the off-diagonal terms of the Jacobian matrix $A$ must be found. The iteration sequence was identical to point Jacobi, except that the off-diagonal terms of matrix $A$ multiplied the current approximation to $\Delta Q$ and were subtracted from the residual. The point Gauss-Seidel method can be written for $i = 1, \ldots, m$ as

$$\Delta Q_i^{(1)} = D_{i,i}^{-1}\left[R_i^n - \sum_{j=1}^{i-1} M_{i,j} * \Delta Q_j^{(1)} - \sum_{j=i}^{m} N_{i,j} * \Delta Q_j^{(0)}\right] \tag{24}$$

The superscripts on $\Delta Q$ refer to the inner iteration number of the Gauss-Seidel method on the linear system. Typically, $\Delta Q^{(0)}$ is an initial guess for the Gauss-Seidel solver, and $\Delta Q^{(1)}$ is used to update $Q^n$ to $Q^{n+1}$. Because of the recursive nature of the point Gauss-Seidel algorithm, complete vectorization of this method on a modern vector supercomputer is not possible. Point Gauss-Seidel can be made symmetrical by sweeping through the list of vertices in both directions before updating $Q$. Utilizing Eq. (24) as the first direction, symmetric point Gauss-Seidel can be written for $i = n, \ldots, 1$ as

$$\Delta Q_i^{(2)} = D_{i,i}^{-1}\left[R_i^n - \sum_{j=1}^{i} M_{i,j} * \Delta Q_j^{(1)} - \sum_{j=i+1}^{m} N_{i,j} * \Delta Q_j^{(2)}\right] \quad (25)$$

with $\Delta Q^{(2)}$ being used to update $Q$.

In the last several years, extensive research has been done in the use of sparse matrix solvers for the solution of nonsymmetric and nonpositive definite matrices.[13–15] The matrices involved in the present work are nonsymmetric and sparse. For a two-dimensional Euler problem with 1000 vertices in a cell-vertex code, there are 4000 unknowns and $16 \times 10^6$ matrix coefficients. Only about 28,000 to 32,000 coefficients are typically nonzero. Many of the sparse matrix solvers are iterative and only require that the user supply a subroutine to compute a matrix multiply of $A$ and a vector. Generally, the matrix multiply is the most expensive operation required to perform a single iteration of the sparse matrix solver. In the present work, the linear system is not an exact Jacobian of the right-hand side, and the diagonal dominance of the system can be controlled by adjusting the time step $\Delta t$. Hence, an exact solution of the linear system is probably not necessary or useful in view of the nonlinearity of the overall problem. What is desired in the overall problem is a method that will give a good approximation to $\Delta Q$ with a minimum of work.

The generalized minimum residual (GMRES) algorithm, developed by Saad and Schultz,[16] is a generalization of the preconditioned conjugate gradient algorithm for nonsymmetric matrices. Both GMRES and the conjugate gradient method can be classified as Krylov subspace methods. The conjugate gradient method and GMRES converge the linear problem when they have eliminated the error associated with all nonunity eigenvalues present in matrix $A$. In the worst case, none of the eigenvalues of the linear system has a value of 1, and the error associated with all eigenvalues must be eliminated. The classical conjugate gradient methods eliminate the error associated with one eigenvalue every iteration. The eigenvalues of the linear system can be clustered about one by premultiplying the linear system by a preconditioner matrix. Eigenvalues that have a value of 1 do not have any error associated with the solution of the problem and do not have to be eliminated by the sparse matrix solver. Preconditioning was used with GMRES and will be discussed later in this work. GMRES has parameters to control the size of the Krylov subspace, the number of restart attempts, and the overall convergence tolerance of the linear system. One iteration of GMRES represents one dimension of the Krylov subspace. Once the iteration number reaches the subspace dimension, the GMRES solver must be restarted. Each dimension of the Krylov subspace requires the storage of an estimate for $\Delta Q$, and the work increases as the square of the subspace dimension. Still, a larger Krylov subspace dimension will tend to give faster convergence when it is necessary to converge the linear system several orders of magnitude. In this work, a value of 5 was used as

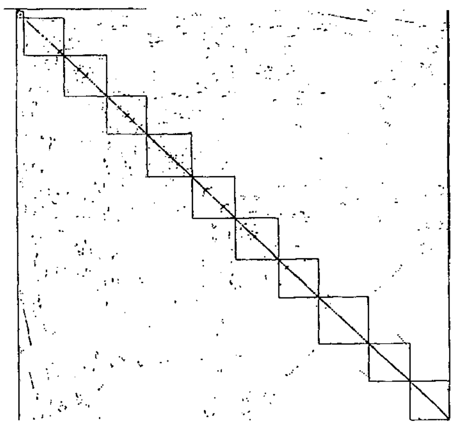
**Fig. 6 Tridiagonal form of Jacobian matrix for line Gauss-Siedel type relaxation.**

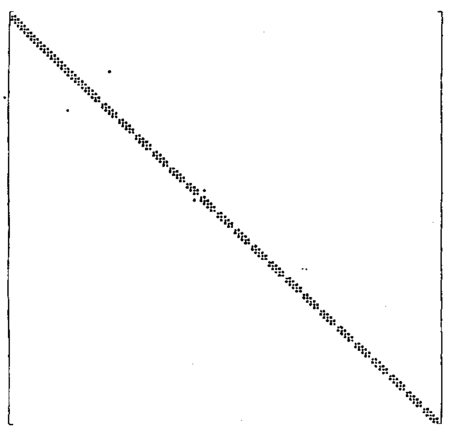
**Fig. 7 Closeup of third submatrix for line Gauss-Siedel type relaxation.**

the Krylov subspace dimension. Twelve restart cycles were used for the GMRES solver before an update of vector $Q$. Twelve restart cycles and a Krylov subspace dimension of 5 represent 60 inner iterations on the linear system equation, Eq. (20). The linear system was considered converged when GMRES reduced the residual of the linear system by a factor of 0.05. The parameters used in the present work may not be optimal for all types of problems.

CHEBYCODE[17,18] was the second sparse matrix solver that was used. CHEBYCODE is an implementation of classical Chebyshev acceleration for a system of equations. Classical Chebyshev acceleration methods require that the maximum and minimum absolute values of the eigenvalues be known. CHEBYCODE will try to approximate the maximum and minimum absolute values of the eigenvalues during the iteration process. CHEBYCODE is limited to solving linear systems whose eigenvalues have only positive real parts. Since the linear system is the Jacobian of an upwind scheme, the matrix is positive definite. Hence, the matrix has only eigenvalues with positive real parts. Preconditioning will improve the convergence rate of CHEBYCODE and was used in the present work. In the present work, the maximum number of iterations used by CHEBYCODE to solve the linear problem was set to 60. The linear problem was considered converged when the residual of the linear problem was reduced by a factor of 0.05. At intervals of 20 CHEBYCODE iterations, the matrix solver would attempt to find new approximations to the maximum and minimum absolute values of the eigenvalues. The user manual[18] recommended an average of the eigenvalues as the initial location of the center of the convex hull (which encloses the eigenvalue spectrum) and a convex hull diameter of zero. In the present work, the initial convex hull was set to a value of 1 (real value) with a diameter of zero. These parameters agreed with the preconditioning used.

The purpose of preconditioning is to cluster the eigenvalues of the resulting system into a group near the value of 1, so as to decrease the number of nonunity eigenvalues that must be eliminated. When all eigenvalues of the linear system are clustered at a value of 1, the system of equations is effectively solved. The linear system $A\Delta Q = R^n$ can be preconditioned by multiplying the system by matrix $B$, as in

$$BA\Delta Q = BR^n \quad (26)$$

and then passing the resulting system to the sparse matrix solver. The ideal preconditioning matrix is $B = A^{-1}$. Obviously, the complete inversion of matrix $A$ will give the exact solution of the linear system without using a sparse matrix solver. The objective then should be to find a preconditioner that is inexpensive to compute yet strongly clusters the eigenvalues of the linear system so that an excellent approximation can be found for $\Delta Q$ in just a few iterations of the sparse matrix solver.

One simple and easy-to-implement method of preconditioning is to use an approximate inverse of matrix $A$ as a preconditioner.

The simplest approximate inverse of $A$ is the inversion of a matrix composed of only the diagonal elements of $A$. Using an inverse of the diagonal elements of $A$ as a preconditioner will result in a simple scaling of the equations in the resulting linear system. A slightly more sophisticated form of this diagonal preconditioning would use an inverse of the diagonal block $4 \times 4$ matrices as a preconditioner, such that $B = D^{-1}$. The matrix $D^{-1}$ is easily computed, takes little storage space, and can be reused if the Jacobian matrix $A$ is reused in the nonlinear iteration. Block-diagonal matrix preconditioning was used in the present work.

The next iterative method tried was the line Gauss-Seidel method. The line Gauss-Seidel method was used in the cell-centered computer code. To perform the line Gauss-Seidel type relaxation, it is first necessary to renumber the cells to get as many elements of $A$ as possible into tridiagonal form. The structure of the resulting matrix $A$ is given in Fig. 6 for a 1005 element mesh. The matrix can be subdivided into several subsections (denoted by the blocks) of variable length. A closeup of the third section, Fig. 7, clearly shows the almost block tridiagonal form. Each subsection is then solved by an equation of the form

$$T\Delta Q = R(Q^n, Q^{n+1}) \qquad (27)$$

where $T$ denotes a tridiagonal submatrix, and the residual becomes a function of $Q^n$ and $Q^{n+1}$. Since the subsections are independent of each other, the inversion of the submatrices can be vectorized over the number of subsections. Vectorization can only be done for subsections with an equal number of elements. When matrix $A$ is subdivided into sections, a minimum number of elements per subsection is imposed. The inversion procedure is then vectorized for the elements in every subsection up to the minimum number of elements allowed in a subsection, and the rest of the elements for each subsection that is larger than the minimum length are computed in scalar mode.

Once the block tridiagonal submatrices have been inverted, the values of $\Delta Q$ are solved for sequentially over each subsection of the matrix $A$. The elements to the left of the diagonal on a forward sweep (or the elements to the right of the diagonal on a reverse sweep) through the matrix are included in the solution procedure through relaxation. After the values for $\Delta Q$ of a subsection are calculated, the residual for the next subsection is computed using these new updated values, which results in a nonlinear update of the residual.

The LU decomposition method attempts to solve the system $A$ exactly. A reverse Cuthill-McKee[19] renumbering scheme was used as a preprocessor for the LU decomposition to reduce the fill of the Jacobian matrix (Fig. 8), thus reducing the storage requirements and CPU time required. The solution procedure involves factoring the linearized coefficient matrix $A$ into the product of a lower triangular and upper triangular matrix (L and U) such that

$$[LU]\Delta Q = R^n \qquad (28)$$

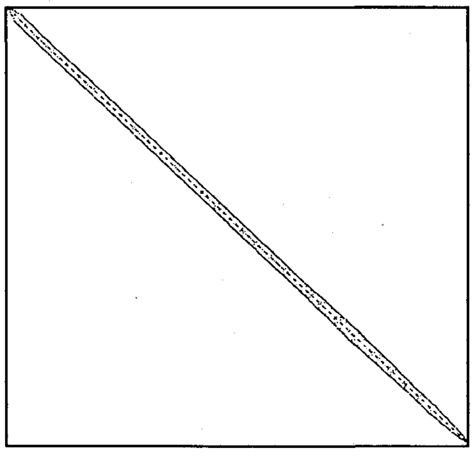

**Fig. 8 Jacobian matrix after reverse Cuthill-McKee renumbering scheme, used for LU decomposition.**

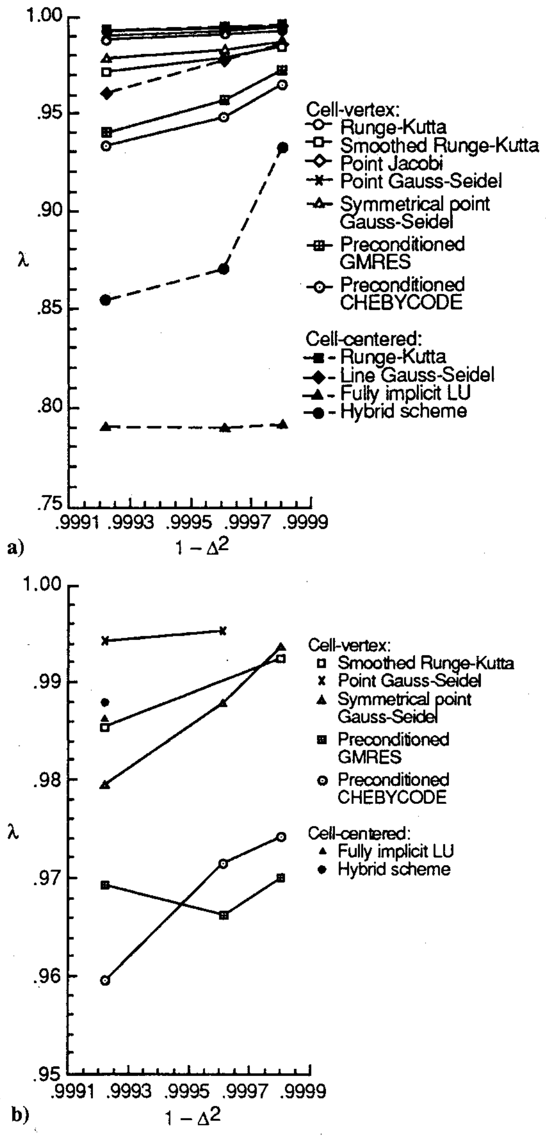

**Fig. 9 Transonic circular arc in a channel, $M_\infty = 0.85$: a) first-order spectral radius comparison and b) higher order spectral radius comparison.**

and then solving the system $L\Delta Q^* = R^n$ by forward substitution and $U\Delta Q = \Delta Q^*$ by backward substitution. This procedure reduces to Newton's method as $\Delta t \to \infty$, and as a result it exhibits quadratic convergence to the solution of the nonlinear system of equations $R(Q^{n+1})$ when the left-hand side of the linear system is an exact linearization of the right-hand side.

The hybrid scheme is simply a combination of the Runge-Kutta and LU decomposition with a reuse of the Jacobian. The purpose of using Runge-Kutta in conjunction with LU decomposition is to avoid most of the factorization CPU time by using Runge-Kutta to converge the residual one order of magnitude, and to perform only one LU factorization. The rest of the iterations are performed using a frozen Jacobian and forward-backward substitution.

## Results and Conclusions

Convergence rates for transonic ($M_\infty = 0.85$) flow over a circular arc in a channel are presented for each of the time integration methods discussed earlier. The rates are compared on 1005, 1999, and 4008 element meshes for both first-order and higher order accurate cell-centered and cell-vertex schemes. The higher order solutions were generated using the MUSCL limiter for the cell-vertex scheme and Barth's limiter for the cell-centered scheme.

Convergence rates are given in Figs. 9 and 10 for each method in terms of spectral radius and time convergence rate plots with

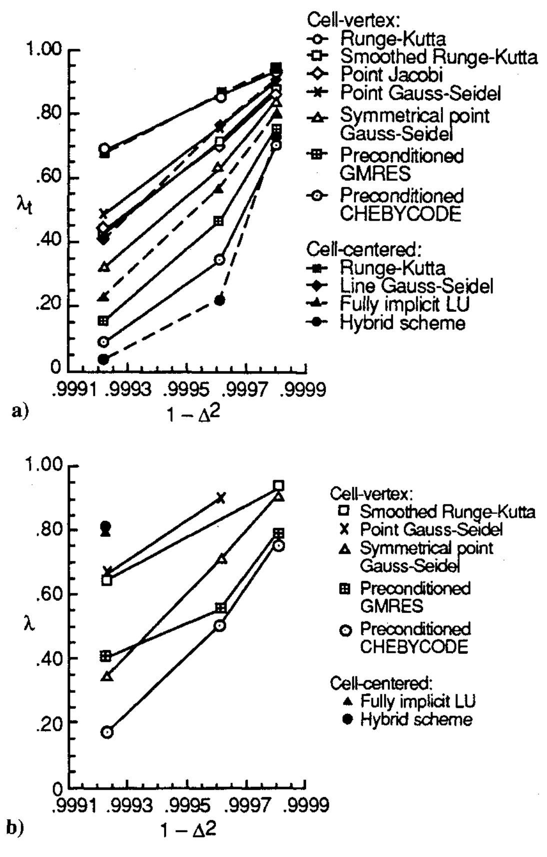

Fig. 10 Transonic circular arc in a channel, $M_\infty = 0.85$: a) first-order time convergence rate comparison and b) higher order convergence rate comparison.

respect to grid spacing $\Delta$. The spectral radius $\lambda$ was computed as

$$\lambda = \left[\frac{\|L_2(R^n)\|}{\|L_2(R^1)\|}\right]^{1/n}$$

where $L_2(R)$ is the Euclidean norm of the residual, and $n$ is the number of iterations. The time convergence rate $\lambda_t$ was defined as

$$\lambda_t = \left[\frac{\|L_2(R^n)\|}{\|L_2(R^1)\|}\right]^{1/t}$$

where $t$ is the CPU time when iteration $n$ occurred. The time convergence rate could be viewed as the factor by which the $L_2$ norm of the residual is reduced per 1 s of CPU time. These plots are extremely important when considering the application of these time integration schemes to three-dimensional problems. The spectral radius and time convergence rate plots give an indication of the possible behavior of these algorithms for larger problems.

Looking at the first-order results, the performance of the hybrid scheme degrades rapidly as the mesh increases from 1999 to 4008 elements. However, the individual fully implicit LU and explicit Runge-Kutta follow nearly linear paths. Therefore, the poor performance of the hybrid method for larger grid sizes can be attributed to either a decrease in the compatibility of the two schemes or to an increase in the percentage of time spent performing Runge-Kutta iterations as the number of elements in the mesh increases. The remaining algorithms also display linear behavior as mesh size increases with GMRES and CHEBYCODE presenting the most attractive results.

The higher order results contain mainly those algorithms from the cell-vertex code because the limiter in the cell-centered code prevented convergence. The performance of the symmetric Gauss-Seidel method degrades quickly with steadily increasing mesh fineness for a higher order solution. The sparse matrix solvers also degrade with increasing mesh sizes, but the asymptotic limit appears to be better than the explicit schemes.

In general, for first-order problems, the fully implicit LU method was fastest in terms of iteration count, whereas the hybrid scheme was best in terms of CPU usage. However, as the mesh size increased, the sparse matrix solvers GMRES and CHEBYCODE appeared more attractive than the hybrid scheme. Also for larger problems such as three-dimensional applications, GMRES and CHEBYCODE require far less storage than the LU and hybrid methods. For higher order problems, GMRES used the fewest iterations, whereas CHEBYCODE was the fastest. The significant difference between the first- and higher order results is believed to be mainly due to the different methods used for the higher order calculations. The line Gauss-Seidel type relaxation, LU, and hybrid schemes were computed using Barth's method, whereas Gauss-Seidel, GMRES, and CHEBYCODE were converged utilizing MUSCL differencing. The difficulty in converging the higher order problems was thought to be the highly nonlinear behavior of Barth's limiter and the inconsistency of a first-order left-hand side and a higher order right-hand side.

Another attractive scheme for three-dimensional application is the Runge-Kutta with residual smoothing scheme. This scheme requires very little storage, is simple to code, and has reasonable convergence behavior.

A concern arose when comparing the line Gauss-Seidel type scheme to the Gauss-Seidel scheme. One would expect the line

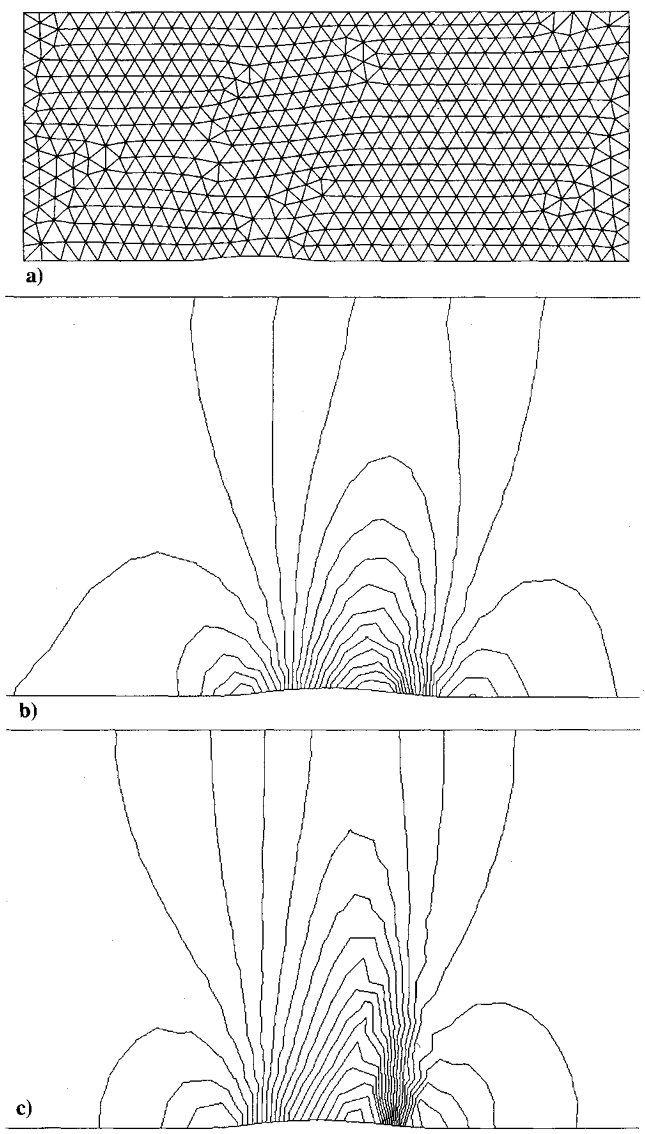

Fig. 11 Transonic circular arc in a channel, $M_\infty = 0.85$: a) mesh; 1005 elements, 551 vertices, b) higher order cell-centered pressure contours, and c) higher order cell-vertex pressure contours.

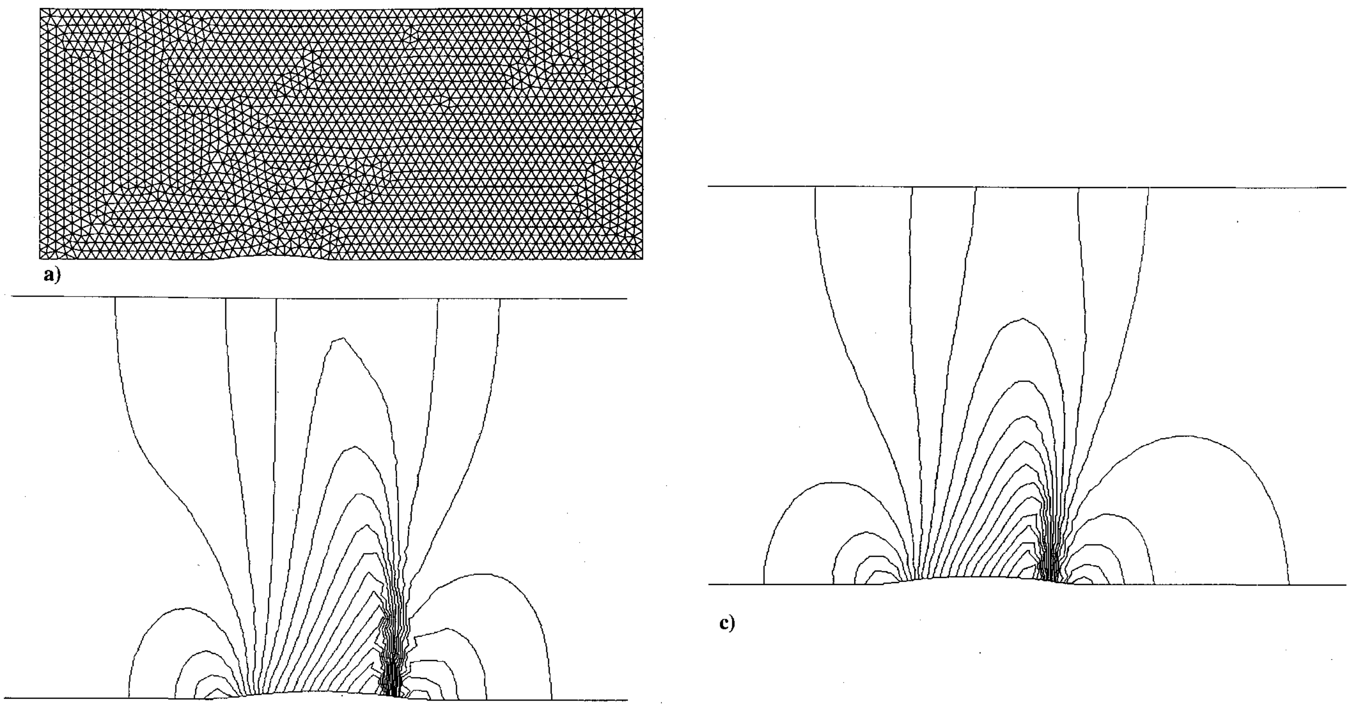

Fig. 12 Transonic circular arc in a channel, $M_\infty = 0.85$: a) mesh; 4008 elements, 2098 vertices, b) higher order cell-centered pressure contours, and c) higher order cell-vertex pressure contours.

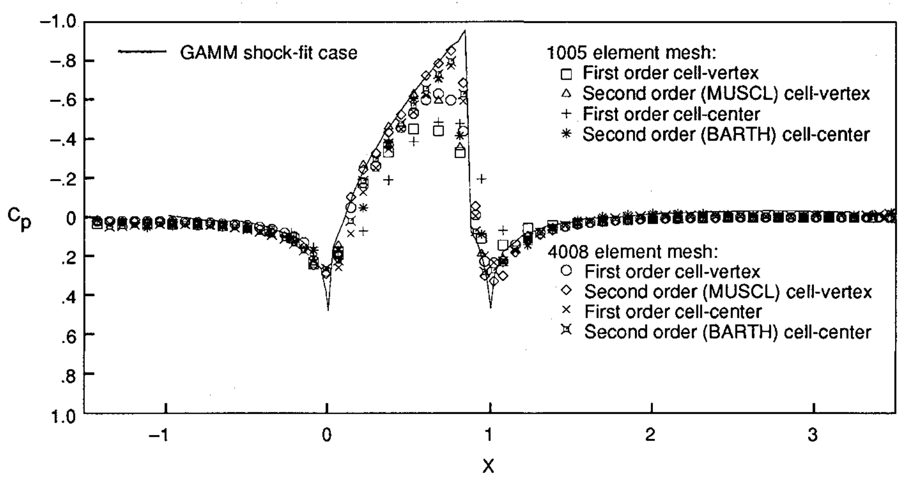

Fig. 13 Comparison of unstructured results with GAMM workshop[20] results on structured mesh, 72 × 21 grid for coefficient of pressure on lower surface of transonic circular arc in a channel, $M_\infty = 0.85$.

Gauss-Seidel type method to converge faster than the Gauss-Seidel method as it does for structured mesh calculations because it computes more terms implicitly and has less terms to relax. However, several problems exist with the currently developed line Gauss-Seidel type scheme that could be fixed to make it better. First, the terms inside each submatrix that are not part of the block tridiagonal form are completely neglected. Even if only a few terms per submatrix are neglected, the effect on convergence could be very significant because the solution to each submatrix affects the solution of the remaining submatrices. Second, because the line Gauss-Seidel type scheme utilizes a nonlinear update of the residual to relax the terms outside the submatrices, it was not possible to make it symmetrical as was done for the Gauss-Seidel scheme. A possible solution to both of these problems would be to treat the terms that are not in block tridiagonal form exactly as the off-diagonal terms of the Gauss-Seidel method.

Figure 11 presents the mesh and higher order pressure contour solutions for the 1005 element test case. Figure 12 contains the corresponding plots for the 4008 element test case. higher order pressure contours are shown for both the cell-centered and cell-vertex discretizations. In this problem, the flow first compresses near the leading edge of the circular arc. It then expands over top until it becomes supersonic near the leading edge, causing a normal shock to form. Finally, the flow expands over the trailing edge of the circular arc. The contours in the solutions appear to be smooth except near the normal shock. In Fig. 13, a plot of the pressure coefficient $C_p$ on the lower surface is compared between all of the first and higher order solutions for the 1005 and 4008 element test cases and a shock-fitted solution from a GAMM workshop.[20] The present results compare quite favorably with the shock-fitted solution from the GAMM workshop. It is easy to see that the finest mesh (4008 elements) higher order solution has the best comparison.

## Acknowledgments

The first and third authors were supported by NASA Grants NAG-1-776 and NAG-1-992. The second author was supported by

NASA Contract NAS-1-18599. The authors would also like to extend their sincere thanks to Steven Ashby for supplying the CHEBYCODE Fortran subroutines.

## References

[1]Riggins, D. W., Walters, R. W., and Pelletier, D., "The Use of Direct Solvers for Compressible Flow Computations," AIAA Paper 88-0229, Jan. 1988.

[2]Turkel, E., and Van Leer, B., "Flux-Vector Splitting and Runge-Kutta Methods for the Euler Equations," Inst. for Computer Applications in Science and Engineering, ICASE Rept. 84-27, June 1984; also NASA CR 172415, 1984.

[3]Barth, T. J., and Jespersen D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper 89-0336, Jan. 1989.

[4]Mavriplis, D., and Jameson, A., "Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes," Inst. for Computer Applications in Science and Engineering, ICASE Rept. 87-53, July 1987; also NASA CR 178346, Jan. 1987.

[5]Stoufflet, B., Périaux, J., Fezoui, F., and Dervieux, A., "Numerical Simulation of 3-D Hypersonic Euler Flows Around Space Vehicles Using Adapted Finite-Elements," AIAA Paper 87-0560, Jan. 1987.

[6]Whitaker, D. L., "Two-Dimensional Euler Computations on a Triangular Mesh Using an Upwind, Finite-Volume Scheme," Ph.D. Thesis, Dept. of Aerospace and Ocean Engineering, Virginia Polytechnic Inst. and State Univ., Blacksburg, VA, 1988.

[7]Mavriplis, D., and Jameson, A., "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes," AIAA Paper 87-0353, Jan. 1987.

[8]Wigton, L. B., Yu, N. J., and Young, D. P., "GMRES Acceleration of Computational Fluid Dynamics Codes," AIAA Paper 85-1494, July 1985.

[9]Bender, E. E., and Khosla, P. K., "Solution of the Two-Dimensional Navier-Stokes Equations Using Sparse Matrix Solvers," AIAA Paper 87-0603, Jan. 1987.

[10]Löhner, R., "Improved Adaptive Refinement Strategies for Finite Element Aerodynamic Computations," AIAA Paper 86-0499, Jan. 1986.

[11]Roe, P. L., "Characteristic Based Schemes for the Euler Equations," *Annual Review of Fluid Mechanics,* Vol. 18, 1986, pp. 337–365.

[12]Van Albada, G. D., Van Leer, B., and Roberts, W. W., Jr., "A Comparative Study of Computational Methods in Cosmic Gas Dynamics," *Astronomy and Astrophysics,* Vol. 108, 1982, pp. 76–84.

[13]Concus, P., and Golub, G., "A Generalized Conjugate Gradient Method for Nonsymmetric Systems of Linear Equations," Computer Science Dept., Stanford Univ., Rept. Stan-CS-76-535, Stanford, CA, 1976.

[14]Elman, H. C., "Preconditioned Conjugate Gradient Methods for Nonsymmetric Systems of Linear Equations," *Advances in Computer Methods for Partial Differential Equations-IV,* edited by R. Vichnevetsky and R. S. Stepleman, IMACS, New Brunswick, NJ, 1981, pp. 409–417.

[15]Axelsson, O., "Conjugate Gradient Type Methods for Partial Differential Equations," *Linear Algebra and Applications,* Vol. 29, 1980, pp. 1–16.

[16]Saad, Y., and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," Research Rept. YALEU/DCS/RR-254, Aug. 1983.

[17]Manteuffel, T. A., "An Iterative Method for Solving Nonsymmetric Linear Systems with Dynamic Estimation of Parameters," Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, IL, 1975.

[18]Ashby, S. A., "CHEBYCODE: A FORTRAN Implementation of Manteuffel's Adaptive Chebyshev Algorithm," Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Rept. UIUCDCS-R-85-1203, Urbana, IL, May 1985.

[19]Duff, I. S., Erisman, A. M., and Reid, J. K., *Direct Methods for Sparse Matrices,* Oxford Univ. Press, New York, 1986, pp. 153–157.

[20]"Numerical Methods for the Computation of Inviscid Transonic Flows with Shock Waves—A GAMM Workshop," *Notes on Numerical Fluid Mechanics,* Vol. 3, Vieweg, Braunschweig, Germany, 1981.