AIAA JOURNAL
Vol. 33, No. 5, May 1995

Direct and Iterative Algorithms for the Three-Dimensional Euler Equations

K. J. Vanden*
Wright Laboratory, Eglin Air Force Base, Florida 32542
and
D. L. Whitfield†
Mississippi State University, Starkville, Mississippi 39762

Direct and iterative algorithms have been developed for solving a finite volume discretization of the three-dimensional Euler equations in curvilinear coordinates. The Euler equations are discretized using numerical derivatives of the numerical flux vector for the Jacobians. Two direct solvers are formulated, one of which has a diagonal plane matrix structure with significantly lower memory requirements. The direct solvers are used as a benchmark in measuring the convergence rate and robustness of more computationally efficient solvers which include two factored approaches, a Newton-relaxation algorithm and a discretized Newton-relaxation algorithm, which uses numerical Jacobians. A diagonal plane formulation for the Newton-relaxation algorithms has also been developed that may have the potential for massive parallelization. It is demonstrated that the Newton-relaxation approach can give convergence rates and robustness equal to that of a direct solver for three-dimensional problems. As a demonstration of the robustness of both the Newton-relaxation algorithm and numerical Jacobians, quadratic convergence to machine zero is demonstrated.

Introduction

UPWIND computational fluid dynamic (CFD) methods are popular because unlike centrally differenced schemes they do not require the addition of artificial viscosity terms to achieve stability. In addition, implicit upwind formulations are more advantageous than implicit central difference formulations because they generally result in linear systems which are more diagonally dominant. In recent years a number of highly efficient relaxation methods have emerged which take advantage of this fact.

Many of these formulations can be shown to be identical to a Newton's method for finding the solution to a set of nonlinear equations except for approximations made in the Jacobian matrix. A number of researchers including Venkatakrishnan,[1] Bailey and Beam,[2] and Orkwis[3] have studied Newton's method solvers. Although all of these studies reported quadratic convergence, the direct solution type of approach that was used is impractical for large three-dimensional problems due to storage requirements. However, these studies have proved useful in determining the limiting performance of algorithms in use today.

Another approach taken to applying Newton's method to CFD problems has been through the use of relaxation methods to obtain approximate solutions to the linear systems at each Newton iteration. These Newton-relaxation methods offer the advantage of having memory requirements comparable to current factored methods and, thus, may be practical for large three-dimensional problems. Whitfield[4] has studied a Newton-relaxation method that bears a close relationship to the current factorization approaches. More recently, Taylor et al.[5] have demonstrated this approach for the two-dimensional, thin layer, laminar Navier–Stokes equations. They used a Gauss-Seidel algorithm for the relaxation step of their two-dimensional Newton's method solver which was shown to be capable of quadratic convergence.

Presented as Paper 93-3378 at the AIAA 11th Computational Fluid Dynamics Conference, Orlando, FL, July 6–9, 1993; received Sept. 25, 1993; revision received June 2, 1994; accepted for publication June 15, 1994.

*Research Scientist, Armament Directorate, Flight Vehicles Branch. Senior Member AIAA.

†Professor of Aerospace Engineering, NSF Engineering Research Center. Member AIAA.

The specific goal of this study is to evaluate the Newton-relaxation approach as a means of accelerating convergence and increasing robustness. Unlike previous efforts, which were for two-dimensional or axisymmetric problems, this study is for three-dimensional problems. An additional goal of this study is to demonstrate three-dimensional quadratic convergence with numerical Jacobians and to determine its practicality.

Numerical Formulation

The nondimensionalized Euler equations in curvilinear coordinates, written in strong conservation form without body forces, are

$$\frac{\partial \boldsymbol{Q}}{\partial \tau} + \frac{\partial \boldsymbol{F}}{\partial \xi} + \frac{\partial \boldsymbol{G}}{\partial \eta} + \frac{\partial \boldsymbol{H}}{\partial \zeta} = 0 \tag{1}$$

where

$$\boldsymbol{Q} = J \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{bmatrix} \qquad \mathcal{K} = J \begin{bmatrix} \rho \mathcal{Q} \\ \rho u \mathcal{Q} + k_x p \\ \rho v \mathcal{Q} + k_y p \\ \rho w \mathcal{Q} + k_z p \\ \mathcal{Q}(e+p) \end{bmatrix}.$$

Here $\boldsymbol{F} = \mathcal{K}$ for $\mathcal{Q} = U, k = \xi$; $\boldsymbol{G} = \mathcal{K}$ for $\mathcal{Q} = V, k = \eta$; and $\boldsymbol{H} = \mathcal{K}$ for $\mathcal{Q} = W, k = \zeta$. The density of the fluid is ρ; the velocity components in the curvilinear coordinate directions ξ, η, and ζ are U, V, and W, respectively; $e = [p/(\gamma - 1)] + \frac{1}{2}\rho(u^2 + v^2 + w^2)$ is the energy per unit volume; and p is the pressure. The relationship between e and p is the result of the perfect gas assumption where γ is the ratio of specific heats.

Discretizations of the Governing Equations

An implicit, finite volume, first-order time-accurate discretization of Eq. (1) may be written as

$$\Delta \boldsymbol{Q}^n = -\Delta\tau\left(\delta_\xi \boldsymbol{F}^{n+1} + \delta_\eta \boldsymbol{G}^{n+1} + \delta_\zeta \boldsymbol{H}^{n+1}\right) \tag{2}$$

where

$$\Delta \boldsymbol{Q}^n = \boldsymbol{Q}^{n+1} - \boldsymbol{Q}^n \qquad \delta_\xi \boldsymbol{F}^{n+1} = \left(\boldsymbol{F}^{n+1}_{i+\frac{1}{2},j,k} - \boldsymbol{F}^{n+1}_{i-\frac{1}{2},j,k}\right)$$

The $i+\frac{1}{2}$ denotes a cell interface, and the dependent variables are assumed to be constant over each computational cell. For a first-order numerical flux (denoted by a caret) of the form

$$\hat{F}^{n+1}_{i+\frac{1}{2}} = F\left(Q^{n+1}_{i,j,k}, Q^{n+1}_{i+1,j,k}\right) \tag{3}$$

Taylor's theorem can be used to linearize $\delta_\xi F^{n+1}$ in time. Thus,

$$\delta_\xi F^{n+1} = \hat{F}^n_{i+\frac{1}{2},j,k} - \hat{F}^n_{i-\frac{1}{2},j,k} - \frac{\partial \hat{F}^n_{i-\frac{1}{2},j,k}}{\partial Q_{i-1}} \Delta Q^n_{i-1,j,k} + \left(\frac{\partial \hat{F}^n_{i+\frac{1}{2},j,k}}{\partial Q_{i,j,k}} - \frac{\partial \hat{F}^n_{i-\frac{1}{2},j,k}}{\partial Q_{i,j,k}} \right) \Delta Q^n_{i,j,k} + \frac{\partial \hat{F}^n_{i+\frac{1}{2},j,k}}{\partial Q_{i+1,j,k}} \Delta Q^n_{i+1,j,k} + \cdots \tag{4}$$

where the $[p, r]$ component of a given flux Jacobian is calculated numerically by using one-sided derivatives

$$\frac{\partial f_p}{\partial q_r} \doteq \frac{f_p(Q^n + \epsilon e^r)_{i+\frac{1}{2},j,k} - f_p(Q^n)_{i+\frac{1}{2},j,k}}{\epsilon} \tag{5}$$

where f_p is the pth component of the numerical flux vector and q_r is the rth component of the dependent variable vector associated with the flux Jacobian. Also, e^r is the rth canonical vector and ϵ is a small number. The choice of epsilon will be discussed later. Note that although four 5×5 flux Jacobian matrices are present in this equation for each point in the grid, only two are actually calculated and stored. By defining

$$\overline{A^n}_{i,j,k} = \frac{\partial \hat{F}^n_{i+\frac{1}{2},j,k}}{\partial Q_{i,j,k}} \qquad \overline{\overline{A^n}}_{i,j,k} = \frac{\partial \hat{F}^n_{i+\frac{1}{2},j,k}}{\partial Q_{i+1,j,k}}$$

the other two flux Jacobians can be written as

$$\frac{\partial \hat{F}^n_{i-\frac{1}{2},j,k}}{\partial Q_{i,j,k}} = \overline{\overline{A^n}}_{i-1,j,k} \qquad \frac{\partial \hat{F}^n_{i-\frac{1}{2},j,k}}{\partial Q_{i-1,j,k}} = \overline{A^n}_{i-1,j,k}$$

Following a similar linearization procedure for the terms $\delta_\eta G^{n+1}$ and $\delta_\zeta H^{n+1}$, and introducing the difference operators

$$\nabla_\xi(\ )_{i,j,k} = (\ )_{i,j,k} - (\ )_{i-1,j,k}$$

$$\Delta_\xi(\ )_{i,j,k} = (\ )_{i+1,j,k} - (\ )_{i,j,k}$$

Equation (2) can be written as

$$\left[\frac{I}{\Delta\tau} + \nabla_\xi \overline{A^n}_{i,j,k} \cdot + \Delta_\xi \overline{\overline{A^n}}_{i-1,j,k} \cdot + \nabla_\eta \overline{B^n}_{i,j,k} \cdot + \Delta_\eta \overline{\overline{B^n}}_{i,j-1,k} \cdot + \nabla_\zeta \overline{C^n}_{i,j,k} \cdot + \Delta_\zeta \overline{\overline{C^n}}_{i,j,k-1} \cdot \right] \Delta Q^n_{i,j,k} = -\left(\delta_\xi F^n + \delta_\eta G^n + \delta_\zeta H^n\right) \tag{6}$$

where I is the identity matrix and the $(\cdot)$ indicates that the ∇_ξ operator acts on the entire term $(\overline{A^n}_{i,j,k} \Delta Q^n_{i,j,k})$, for example. To obtain steady-state solutions to the Euler equations the discretizations just developed will be used with time treated as an iteration parameter.

Numerical Flux

Equation (6) requires the evaluation of the flux vectors at the cell faces. Two popular methods for calculating these fluxes are the flux vector split scheme of Steger and Warming[6] and an approximate Riemann solver due to Roe.[7] Both of these schemes are examples of upwind schemes in which physical information propagation is introduced into the discretization of the governing equations. A second-order-accurate Steger and Warming numerical flux can be obtained by extrapolating the dependent variables in a monotonic upwind-centered scheme for conservation (MUSCL) fashion before constructing the flux. Details of this approach are found in Anderson et al.[8] Second-order accuracy for the Roe scheme can be obtained by using the higher order numerical flux due to Osher and Chakravarthy.[9]

Direct Method

Equation (6) represents a linear system of equations that resulted from the discretization and linearization in time of the unsteady Euler equations. This system can be expressed in the general form

$$\left[\frac{I}{\Delta\tau} + \left(\frac{\partial R}{\partial Q} \right)^n \right] [\Delta Q^n] = -[R^n] \tag{7}$$

where

$$R^n_{i,j,k} = \left(\delta_\xi F^n_{i,j,k} + \delta_\eta G^n_{i,j,k} + \delta_\zeta H^n_{i,j,k}\right)$$

The notation convention is that $Q^n_{i,j,k}$ and $R^n_{i,j,k}$ denote 5×1 subvectors of the column vectors $[Q^n]$ and $[R^n]$, respectively. Each equation in the linear system given by Eq. (7) is of the form

$$D^n_{i,j,k} \Delta Q^n_{i,j,k} - \overline{A^n}_{i-1,j,k} \Delta Q^n_{i-1,j,k} + \overline{\overline{A^n}}_{i,j,k} \Delta Q^n_{i+1,j,k} - \overline{B^n}_{i,j-1,k} \Delta Q^n_{i,j-1,k} + \overline{\overline{B^n}}_{i,j,k} \Delta Q^n_{i,j+1,k} - \overline{C^n}_{i,j,k-1} \Delta Q^n_{i,j,k-1} + \overline{\overline{C^n}}_{i,j,k} \Delta Q^n_{i,j,k+1} = R^n_{i,j,k} \tag{8}$$

where

$$D^n_{i,j,k} = \left[(I/\Delta\tau) + \overline{A^n}_{i,j,k} - \overline{\overline{A^n}}_{i-1,j,k} + \overline{B^n}_{i,j,k} - \overline{\overline{B^n}}_{i,j-1,k} + \overline{C^n}_{i,j,k} - \overline{\overline{C^n}}_{i,j,k-1} \right]$$

The direct solution of Eq. (7), which can be written as

$$[\Delta Q^n] = -\left[\frac{I}{\Delta\tau} + \left(\frac{\partial R}{\partial Q} \right)^n \right]^{-1} [R^n] \tag{9}$$

is computationally impractical for problems of current interest even though it is not necessary to explicitly compute the inverse of the Jacobian matrix. Approximations to Eq. (7) are normally solved which are well within the capabilities of currently available supercomputers, although, in general, this usually results in a lower convergence rate as compared to solving the direct problem at each iteration. Therefore it is the goal to identify which approximate methods, if any, give a convergence performance comparable to that obtained from a direct solution of Eq. (7). For this reason, direct solutions for small three-dimensional problems, however inefficient, are desired as a means of comparison.

In the limit as $\Delta\tau \to \infty$ Eq. (7) can be viewed as a Newton's method for solving the nonlinear system of equations

$$\delta_\xi F + \delta_\eta G + \delta_\zeta H = 0 \tag{10}$$

which results from the discretization of the steady Euler equations. In terms of Eq. (6) this is more properly called a discretized Newton method since the Jacobian matrix is a difference approximation to the true analytical Jacobian matrix. The direct solution of Eq. (7) with large time steps can be shown to give extremely fast convergence under certain conditions. Dennis and Schnabel[10] show that if $[R^n]$ is continuously differentiable in a neighborhood of a solution Q^*, and the Jacobian matrix is nonsingular at Q^*, then the nondiscretized Newton's method can be expected to have convergence of the form

$$\|Q^{n+1} - Q^*\| \le c\|Q^n - Q^*\|^2 \tag{11}$$

where c is a constant. This type of convergence is called quadratic since the $(n+1)$th error is proportional to the square of the nth error. Once the computed solution becomes sufficiently close to a solution Q^*, extremely rapid convergence is obtained. Ortega and Rheinboldt[11] show that for discretized Newton methods it is also possible to obtain quadratic convergence provided the difference

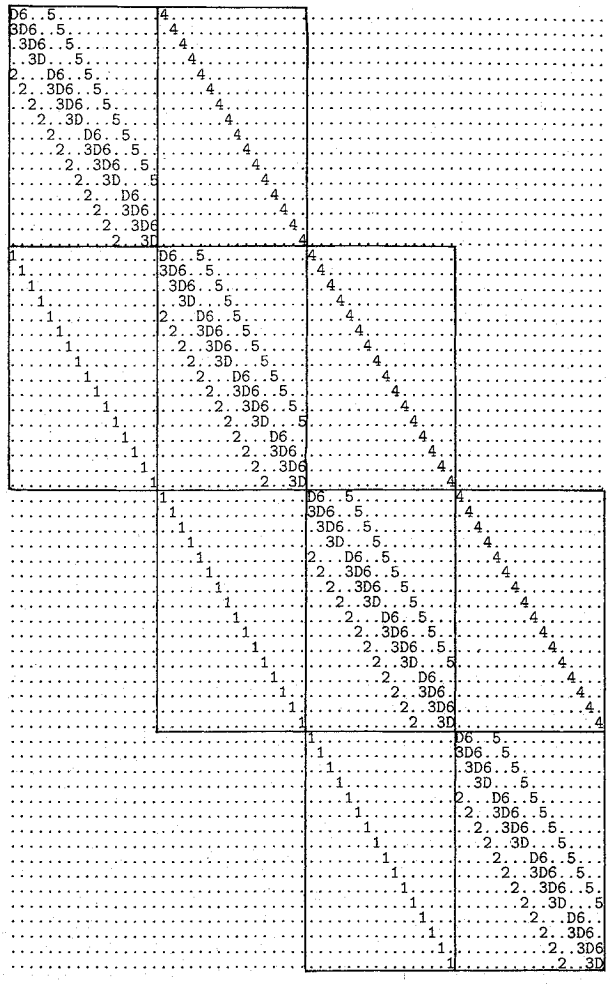

Fig. 1 System matrix structure for conventional direct solver.

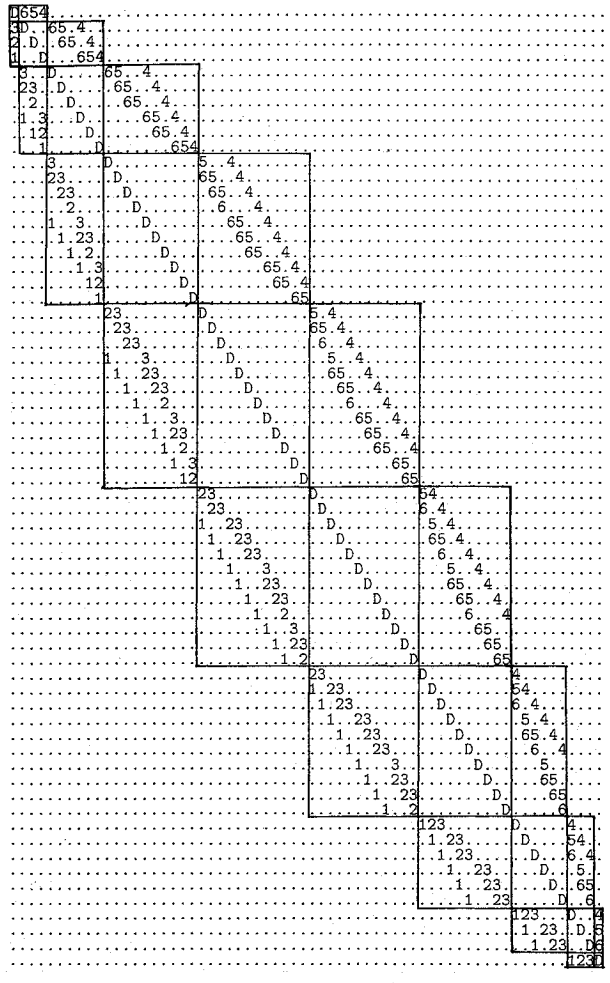

Fig. 2 System matrix structure for diagonal plane direct solver.

approximation to the Jacobian is computed in a certain way. Thus, when the analytical Jacobian is impossible or too difficult to obtain one can use a numerical approximation and still retain the quadratic convergence. Ortega and Rheinboldt point out that it is necessary that

$$\lim_{\epsilon \to 0} \frac{f_p(\boldsymbol{Q}^n + \epsilon \boldsymbol{e}^r)_{i+\frac{1}{2},j,k} - f_p(\boldsymbol{Q}^n)_{i+\frac{1}{2},j,k}}{\epsilon} = \frac{\partial f_p}{\partial q_r} \tag{12}$$

and

$$\lim_{n \to \infty} \epsilon^n = 0 \tag{13}$$

for quadratic convergence. If a fixed epsilon is maintained for all time steps then, in general, the convergence rate will be only linear.

Conventional Direct Solver

The direct solution of Eq. (7) involves storing and performing operations with the system matrix

$$\left[\frac{I}{\Delta\tau} + \left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}\right)^n\right] \tag{14}$$

which even for small grids can be extremely large. The structure of this matrix can have a big impact on the efficiency of the solution procedure. A relatively simple form of the system matrix results when the $[\boldsymbol{R}^n]$ vector is loaded up by incrementing the i, j, k indices in a cyclic fashion. The structure of the system matrix resulting from this approach is shown in Fig. 1 for a $5 \times 5 \times 5$ grid. Here the numerals 1, 2, 3, 4, 5, and 6 represent the $\overline{A^n}$, $\overline{B^n}$, $\overline{C^n}$, $\overline{\overline{A^n}}$, $\overline{\overline{B^n}}$, and $\overline{\overline{C^n}}$ terms in Eq. (8), respectively, and each $(\cdot)$ represents a 5×5 sub-block composed of zeros. The points on the planes $(1, j, k)$, $(ni + 1, j, k)$, $(i, 1, k)$, $(i, nj + 1, k)$, $(i, j, 1)$, and $(i, j, nk + 1)$ are phantom points at which the boundary conditions are calculated explicitly. They are not included in $[\boldsymbol{R}^n]$.

As shown in Fig. 1 this banded structure may be blocked off into a block tridiagonal system where the elements of these blocks are themselves 5×5 blocks. The resulting linear system can then be solved with a block lower-upper (LU) factorization along with block forward and back substitution. The resulting direct solver will be referred to as the conventional direct solver. Of course, the conventional direct solver has a significant number of floating point operations compared to more approximate methods. As noted before, the direct method solvers presented in this study are intended for comparison purposes only.

Diagonal Plane Direct Solver

Alternate structures for the system matrix can be found which lead to direct solvers that are more efficient than the conventional direct solver. Another possible ordering is by planes on which the sum of the three indices i, j, k is held constant. This ordering is also used in wave front or level scheduling methods which are described in Van Der Vorst,[12] and Saad and Shultz.[13] The planes of this ordering will be diagonal planes in the rectangular computational region and lead to the system matrix shown in Fig. 2. This matrix was also blocked so that the previously described block tridiagonal solution procedure could still be used. Although the blocks are no longer constant in size, they have dimensions that allow the necessary matrix-vector and matrix-matrix multiplications. Although the coding of this diagonal plane solver was extremely complex due to the dynamically changing block sizes as one progresses through the solution procedure it has the advantage of requiring significantly less memory than the conventional direct solver. Grid sizes of $49 \times 9 \times 9$, $10 \times 10 \times 10$, and $100 \times 100 \times 100$ would require 5.36, 44.43, and 44.97% less storage, respectively. It should be noted that the diagonal plane structure yields a memory savings because a block tridiagonal solution

procedure was used. Had the system matrix shown in Fig. 2 been treated as a banded system, no memory savings would have been obtained. However, even with the lower memory requirements the diagonal plane direct solver is only usable for small three-dimensional problems. A $49 \times 9 \times 9$ grid only required less than 7×10^6 words of memory on a the Eglin AFB Cray Y-MP 8/2128, but a $97 \times 17 \times 17$ grid would require more memory than the 128×10^6 words of available in core memory.

Factored and Iterative Methods

In normal practice Eq. (7) is approximated by factoring the system matrix into two or more parts. Instead of then solving one large linear system as with a direct solver, a number of smaller linear systems are solved in sequence for each time step. These schemes are much more efficient in terms of memory requirements and execution time per time step, although the factorization error can substantially degrade the convergence rate and reduce stability as compared to a direct solver. A factored or iterative scheme that offers direct solver performance, although requiring substantially lower computational requirements, would be highly desirable.

A number of factored and iterative solvers will now be presented including the standard two-pass solver, a modified two-pass solver, and a Newton-relaxation solver. The Newton-relaxation solver will be shown to be capable of quadratic convergence on three-dimensional grids.

Two-Pass Factorization

Equation (8) can be approximately factored into the following two steps:

$$\left[I + \Delta\tau\nabla_\xi \overline{A^n}\cdot + \Delta\tau\nabla_\eta \overline{B^n}\cdot + \Delta\tau\nabla_\zeta \overline{C^n}\cdot\right][\boldsymbol{Q}^*] = -\Delta\tau[\boldsymbol{R}^n] \quad (15)$$

$$\left[I + \Delta\tau\Delta_\xi \overline{\overline{A^n}}\cdot + \Delta\tau\Delta_\eta \overline{\overline{B^n}}\cdot + \Delta\tau\Delta_\zeta \overline{\overline{C^n}}\cdot\right][\Delta\boldsymbol{Q}^n] = [\boldsymbol{Q}^*] \quad (16)$$

where the first step solves a sparse block lower triangular system, and the second step solves a sparse block upper triangular system. Each of these systems can be solved by passing through the grid and solving a 5×5 linear system at each grid point. These 5×5 systems are solved by LU factorization followed by back and forward substitution.

Modified Two-Pass

A number of techniques have been advocated to replace the standard two-pass factorization, including those by Whitfield,[4] and Yoon and Kwak.[14] As compared to the original two pass factorization, the modified two pass due to Whitfield only requires one LU factorization at each grid point resulting in a significant savings in CPU time. An additional advantage of the modified two pass over the standard two pass is a strengthened block diagonal.

Assuming $(D^n_{i,j,k})^{-1}$ to be nonsingular, Eq. (8) can be written

$$\Delta\boldsymbol{Q}^n_{i,j,k} + \left(D^n_{i,j,k}\right)^{-1}\left[-\overline{A^n}_{i-1,j,k}\Delta\boldsymbol{Q}^n_{i-1,j,k} + \overline{\overline{A^n}}_{i,j,k}\Delta\boldsymbol{Q}^n_{i+1,j,k} - \overline{B^n}_{i,j-1,k}\Delta\boldsymbol{Q}^n_{i,j-1,k} + \overline{\overline{B^n}}_{i,j,k}\Delta\boldsymbol{Q}^n_{i,j+1,k} - \overline{C^n}_{i,j,k-1}\Delta\boldsymbol{Q}^n_{i,j,k-1} + \overline{\overline{C^n}}_{i,j,k}\Delta\boldsymbol{Q}^n_{i,j,k+1}\right] = -\left(D^n_{i,j,k}\right)^{-1}\boldsymbol{R}^n_{i,j,k} \quad (17)$$

where again

$$D^n_{i,j,k} = \left[(I/\Delta\tau) + \overline{A^n}_{i,j,k} - \overline{\overline{A^n}}_{i-1,j,k} + \overline{B^n}_{i,j,k} - \overline{\overline{B^n}}_{i,j-1,k} + \overline{C^n}_{i,j,k} - \overline{\overline{C^n}}_{i,j,k-1}\right]$$

This equation can then be factored in the same manner as the original two-pass scheme to obtain

$$\left[I - \left(D^n_{i,j,k}\right)^{-1}L^n_{i,j,k}\cdot\right]\left[I + \left(D^n_{i,j,k}\right)^{-1}U^n_{i,j,k}\cdot\right]\left[\Delta\boldsymbol{Q}^n_{i,j,k}\right] = -\left(D^n_{i,j,k}\right)^{-1}\boldsymbol{R}^n_{i,j,k} \quad (18)$$

where the $L\cdot$ and $U\cdot$ operators

$$\left[L^n_{i,j,k}\cdot\right]\left[\Delta\boldsymbol{Q}^n_{i,j,k}\right] = \overline{A^n}_{i-1,j,k}\Delta\boldsymbol{Q}^n_{i-1,j,k} + \overline{B^n}_{i,j-1,k}\Delta\boldsymbol{Q}^n_{i,j-1,k} + \overline{C^n}_{i,j,k-1}\Delta\boldsymbol{Q}^n_{i,j,k-1} \quad (19)$$

$$\left[U^n_{i,j,k}\cdot\right]\left[\Delta\boldsymbol{Q}^n_{i,j,k}\right] = \overline{\overline{A^n}}_{i,j,k}\Delta\boldsymbol{Q}^n_{i+1,j,k} + \overline{\overline{B^n}}_{i,j,k}\Delta\boldsymbol{Q}^n_{i,j+1,k} + \overline{\overline{C^n}}_{i,j,k}\Delta\boldsymbol{Q}^n_{i,j,k+1} \quad (20)$$

have been introduced to simplify the notation. This approximately factored equation can then be solved in two steps given by

$$[D^n - L^n\cdot][\boldsymbol{Q}^*] = -[\boldsymbol{R}^n] \quad (21)$$

$$[D^n + U^n\cdot][\Delta\boldsymbol{Q}^n] = [D^n\boldsymbol{Q}^*] \quad (22)$$

or

$$[D^n - L^n\cdot][\boldsymbol{Q}^*] = -[\boldsymbol{R}^n] \quad (23)$$

$$[U^n\cdot][\Delta\boldsymbol{Q}^n] = -[D^n\boldsymbol{Q}^{**}] \quad (24)$$

where

$$[\boldsymbol{Q}^{**}] = [\Delta\boldsymbol{Q}^n] - [\boldsymbol{Q}^*] \quad (25)$$

Newton-Relaxation

If Eq. (23) is solved by using the symmetric Gauss-Seidel method, the first two sweeps would be

$$[D - L][\Delta\boldsymbol{Q}^n]^1 = -[\boldsymbol{R}^n] \quad (26)$$

$$[D + U][\Delta\boldsymbol{Q}^n]^2 = -[\boldsymbol{R}^n] + [L][\Delta\boldsymbol{Q}^n]^1 \quad (27)$$

where $[\Delta\boldsymbol{Q}^n]^0$ has been taken to be zero. Here D, L, and U are the diagonal, lower, and upper parts of the system matrix in Eq. (8), respectively. Eliminating $[L]$ from the backward sweep, the first two sweeps of the symmetric Gauss-Seidel method becomes

$$[D - L][\Delta\boldsymbol{Q}^n]^1 = -[\boldsymbol{R}^n] \quad (28)$$

$$[D + U][\Delta\boldsymbol{Q}^n]^2 = [D][\Delta\boldsymbol{Q}^n]^1 \quad (29)$$

which are the same as the two passes of the modified two-pass scheme given by Eqs. (21) and (22). Thus, the modified two pass can be thought of as either a factored scheme or a relaxation scheme.

As noted before when $\Delta\tau \to \infty$ Eq. (7) becomes

$$\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}\right]^n[\Delta\boldsymbol{Q}^n] = -[\boldsymbol{R}^n]$$

which can be viewed as Newton's method for the discretized steady Euler equations where n now represents a Newton iterate instead of the time step. Also, Whitfield[4] has shown that Newton's method can be used to arrive at the discretization given by Eq. (7). The solution of the nonlinear system of equations given by Eq. (7) requires the solutions to a sequence of linear systems for which the symmetric Gauss-Seidel method can be used. The resulting composite method will be called the Newton-relaxation (NR) algorithm in which Newton's method is the primary iteration, and the symmetric Gauss-Seidel is the secondary iteration. When numerical Jacobians are used the algorithm will be referred to as a discretized Newton-relaxation (DNR) algorithm.

The system matrix structure of the NR and DNR algorithms may be based on either the conventional loading (Fig. 1) or the diagonal plane loading (Fig. 2). With a direct solver algorithm the diagonal plane loading was shown to give a significant memory savings as compared to the conventional loading. The advantage of the diagonal plane loading for the NR and DNR algorithms is in terms of parallel processing. Using a Gauss-Seidel method to solve the linear system resulting from the conventional loading would involve sweeping through the grid and computing a solution at each grid point. This process is in serial because solving for the dependent

variables at a given grid point makes use of the solution obtained for the grid point immediately preceding it in the matrix ordering. Now consider the same procedure for the diagonal plane loading. Referring to the diagonal blocks in Fig. 2 it can be seen that the solutions corresponding to the steps in a given block are not dependent on the solution of the other steps in the same block. They are only dependent on the solution obtained from the last step of the preceding block. The steps in a given block may then be sent to different CPUs and processed in parallel.

Boundary Conditions

The explicit boundary conditions used in this study are characteristic variable boundary conditions. An implicit implementation of the characteristic variable boundary conditions was developed for iterative solvers. In addition to implicit characteristic variable boundary conditions, the combination of an implicit zero pressure gradient boundary condition on the solid surfaces along with implicit far-field characteristic variable boundary conditions was used. Details of these implicit characteristic variable boundary conditions may be found in Vanden.[15]

Results

A comparison of convergence rates for the algorithms discussed in the previous sections will now be presented. First, the performance of a direct solver will be presented as a baseline. Next, the performance of the factored and iterative solvers will be demonstrated and then compared to the direct solver results. Finally, the NR and DNR algorithms with implicit boundary conditions will be shown to be capable of quadratic convergence. As a measure of robustness, all calculations are converged to machine zero.

These comparisons were performed on a wing geometry[16] at transonic and supersonic speeds. A coarse $49 \times 9 \times 9$ C grid and a finer $97 \times 17 \times 17$ C grid were used. The flight conditions for the results presented here are for an angle of attack of 3.06 deg and freestream Mach numbers of 0.84 and 1.5. Unless otherwise stated, the results will be for the Mach 0.84 freestream condition and with the Roe numerical fluxes. A comparison between second-order Roe and experimental data is given in Figs. 3 and 4 for the coarse and fine grids, respectively. The Van Leer limiter was used for all second-order Roe calculations.[17]

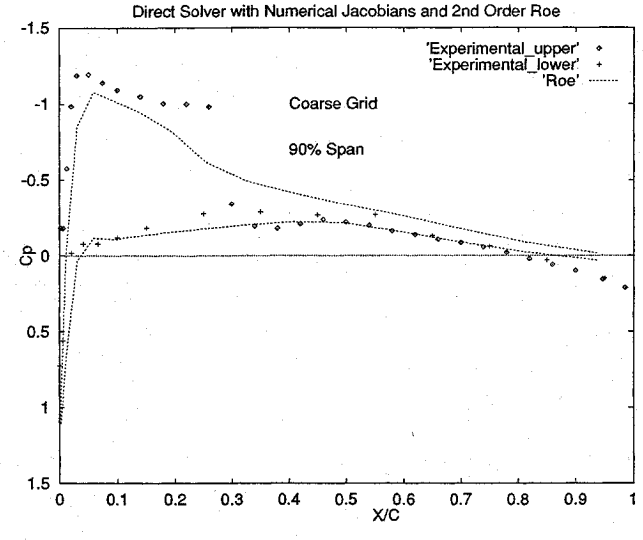

Fig. 3 Comparison of second-order Roe solution and experimental data for coarse grid.

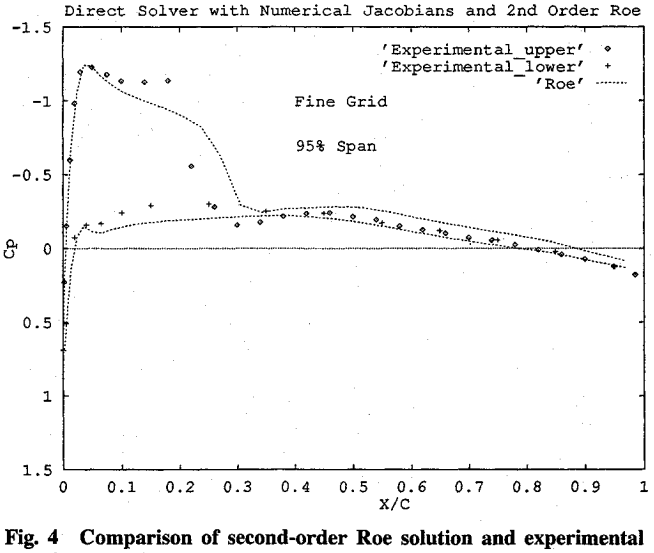

Fig. 4 Comparison of second-order Roe solution and experimental data for fine grid.

Direct Solver Performance

The conventional direct solver was run on the coarse grid for a variety of Jacobian and numerical flux combinations. Not enough memory was available to run it on the fine grid. In addition, explicit characteristic variable boundary conditions were used for all of the direct solver results presented here because implicit conditions would have increased the memory requirements. A popular numerical discretization in use today is the Roe numerical flux but with the Steger-Warming analytical Jacobians. Although better performance is expected with Roe Jacobians this combination is popular because the Steger-Warming Jacobians are easier to obtain analytically. Determination of the analytical Roe Jacobians in a reasonable amount of time would require the use of symbolic mathematics software, as has been done by Barth,[18] Orkwis,[3] and Knight,[19] for example. For relatively simple algorithms, including the one in this study, obtaining the analytical Jacobians may be the best approach. However, there are many instances where numerical Jacobians may be useful, perhaps when nondifferential turbulence models and chemistry terms are present, as one reviewer suggested. It is, therefore, important to verify the robustness of numerical Jacobian evaluations before their use is extended to more complicated algorithms. In this study, the obtaining of quadratic convergence with numerical Jacobians will be offered as evidence of their robustness. Although this study looks at only Euler flows on relatively coarse grids, a related study by Orkwis and Vanden[20] showed that numerical Jacobians retain their accuracy on much finer grids with large aspect ratio cells. This study compared the accuracy of numerical and analytical Roe Jacobians for some Mach 2.0–14.1 viscous flows using a conjugate gradient squared (CGS) algorithm to solve the Navier–Stokes equations.

Figure 5 shows the convergence rates of the the direct solver using both Steger-Warming and Roe Jacobians with second-order Roe fluxes. Referring to Eq. (5) an ϵ of 1×10^{-7} was used in the calcula-

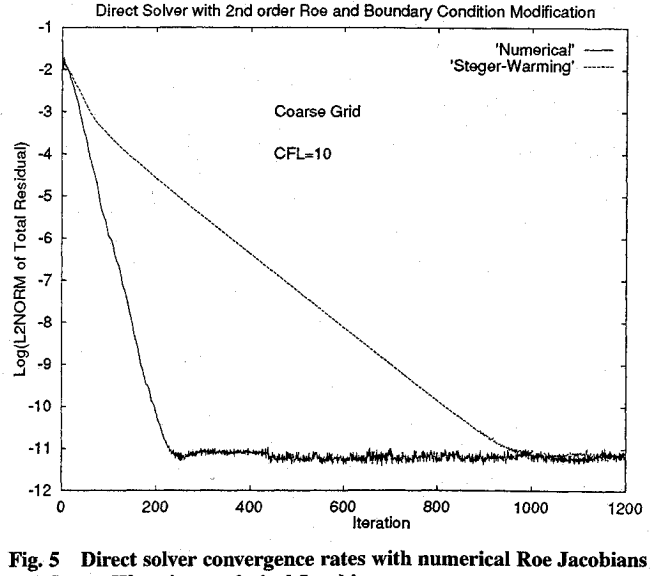

Fig. 5 Direct solver convergence rates with numerical Roe Jacobians and Steger-Warming analytical Jacobians.

tion of the numerical Jacobians. This value was chosen based on the machine accuracy of the Cray Y-MP. See Dennis and Schnabel[10] for a complete discussion. The increase in the CPU time due to the calculation of numerical Jacobians compared to the Steger-Warming analytical Jacobians was small compared to the overall CPU time per iteration for the direct solver. However, the increase in convergence performance when using the numerical Roe Jacobians is significant.

Iterative Solver Performance

The two-pass, modified two-pass, and the DNR iterative solvers were also run on the coarse grid with explicit boundary conditions, numerical Roe Jacobians, and second-order Roe fluxes, the results of which are shown in Figs. 6 and 7. The DNR solver was run both two and five cycles per iteration where a cycle is defined as a solution of Eqs. (24) and (25). At a CFL number of 5 the modified two pass was slightly slower in converging than the two-pass and DNR algorithms (Fig. 6). As the CFL number was increased to 20, Fig. 7, the two pass was unstable and would not run, and the modified two pass was again out performed by the DNR algorithm. In terms of CPU time the two-pass and modified two-pass solvers with explicit boundary conditions took 0.433 and 0.426 s per iteration, respectively. The modified two-pass will be more of a savings over the standard two-pass on much bigger problems because, as mentioned before, it only needs one LU decomposition at each grid point per iteration. The currently used $49 \times 9 \times 9$ grid is too small to show a significant savings between the two pass and modified two pass in terms of CPU time per iteration. In comparison, the DNR algorithm with two and five cycles took 0.572 and 0.833 s per iteration, respectively.

Comparison of Iterative and Direct Solvers

The question remains as to whether any of these iterative solvers approach or equal the convergence performance of a direct solver.

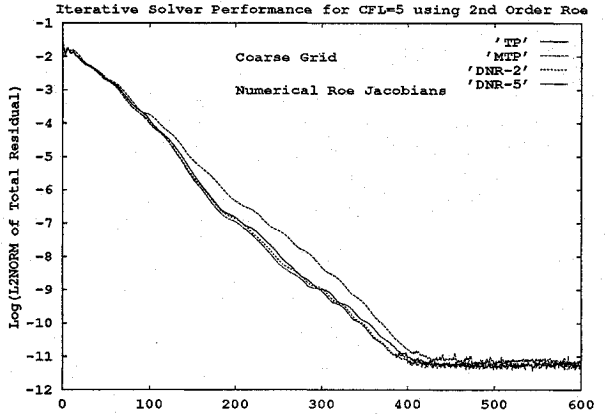

Fig. 6 Convergence rate of two-pass, modified two-pass, DNR with two cycles, and DNR with five cycles for CFL = 5.

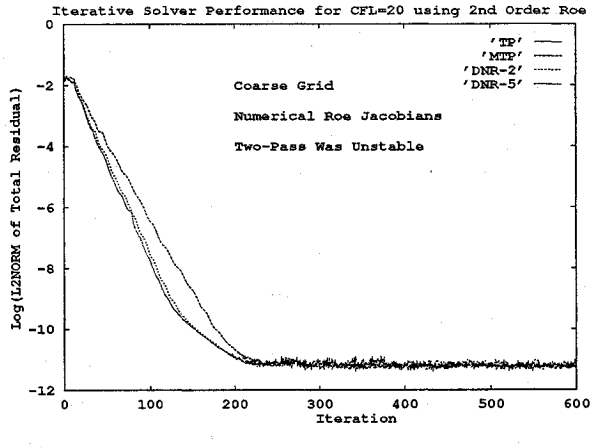

Fig. 7 Convergence rate of two-pass, modified two-pass, DNR with two cycles, and DNR with five cycles for CFL = 20.

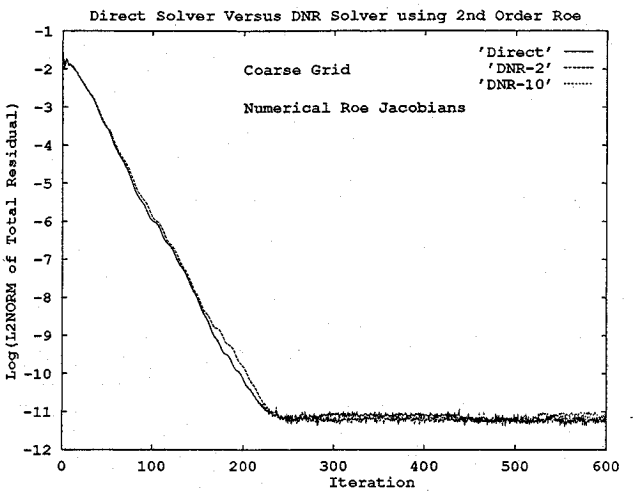

Fig. 8 Convergence rate of direct solver, DNR solver with two cycles, and DNR with 10 cycles for CFL = 10.

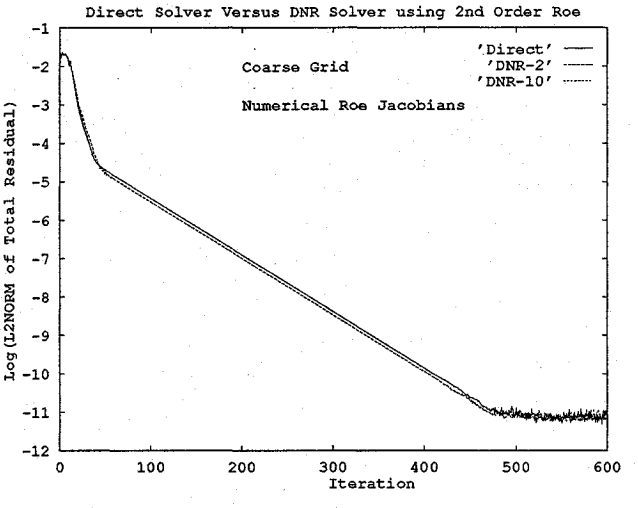

Fig. 9 Convergence rate of direct solver, DNR solver with two cycles, and DNR with 10 cycles for CFL = 50.

As far as computational efficiency is concerned, the iterative solvers were significantly more efficient than the direct solver. For the coarse grid the direct solver required 6.7×10^6 words of memory and 6100 CPU s for 300 iterations whereas the DNR solver with 10 cycles per iteration only required 1.9×10^6 words of memory and approximately 420 CPU s. Figures 8 and 9 show the convergence performance of the DNR algorithm vs the direct solver for CFL numbers of 10 and 50, respectively, where it can be seen that the DNR algorithm with 10 cycles per iteration has the same convergence rate as the direct solver for both CFL numbers. Thus, direct solver performance can be obtained with an iterative algorithm at a fraction of the computational cost. This motivates one to wonder whether the DNR algorithm, with implicit boundary conditions, can achieve quadratic convergence.

Quadratic Convergence

The DNR and NR algorithms were able to achieve quadratic convergence with implicit characteristic variable far-field boundary conditions and an implicit zero pressure gradient boundary condition for the wing surface. In addition, the use of an implicit zero pressure gradient condition for the wing surface along with zeroth-order extrapolated implicit far-field boundary conditions also allowed the DNR and NR algorithms to converge quadratically. All of the quadratic convergence results are using first-order fluxes. The choice of ϵ in the calculation of the numerical derivatives was kept constant at 1.0×10^{-7}. It was noted in Eq. (13) that, in general, it is required that the ϵ value be sequenced to approach zero for a discretized Newton method to achieve quadratic

Table 1 CPU seconds required to reach machine zero with DNR algorithm

CPU time for convergence		
Cycles per iteration	Seconds per iteration	Total CPU seconds
5	0.89	23.39
10	1.42	18.46
20	2.47	19.76
40	4.58	27.48
60	6.68	40.48
154	15.60	93.60

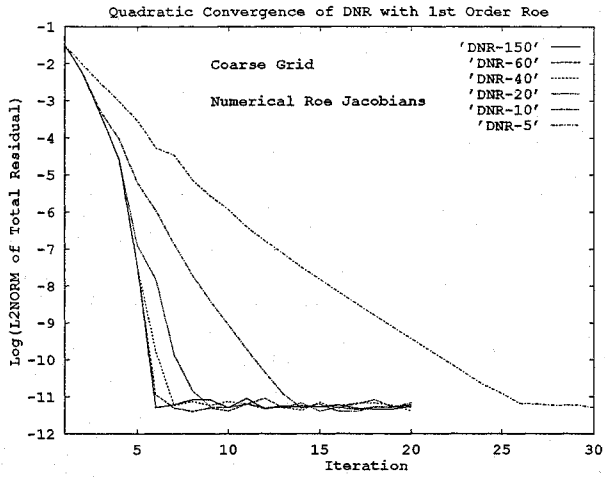

Fig. 10 Convergence of DNR algorithm with implicit characteristic varibale boundary conditions and first-order Roe with numerical Jacobians.

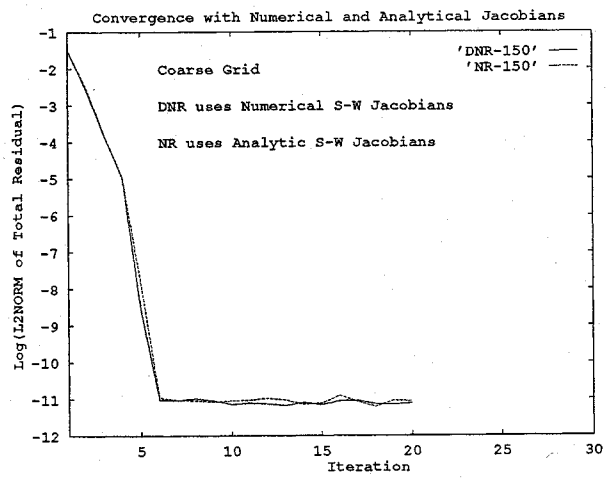

Fig. 11 Convergence of NR algorithm with analytic Jacobians and DNR algorithm with numerical Jacobians using first-order flux vector and 150 cycles.

convergence. It was found that no such sequencing was needed to achieve quadratic convergence for the Euler calculations given in this study. Quadratic convergence using a second-order numerical flux is not possible with Eq. (7) because of the first-order spatial discretizations on the left-hand side.

Figure 10 shows the convergence rate for the DNR algorithm using first-order Roe fluxes, implicit characteristic variable far-field boundary conditions, and implicit zero pressure gradient for the wing surface. The quadratic convergence was obtained by running 150 cycles of the DNR algorithm per iteration. The time step for all cells was kept at a constant 7000 for the first three iterations and then increased to 1.0×10^{99} resulting in quadratic convergence. Recall that before it was noted that as $\Delta\tau \to \infty$ the discretized unsteady Euler equation is the same as a Newton's method for the steady Euler equations. Table 1 gives the number of CPU seconds

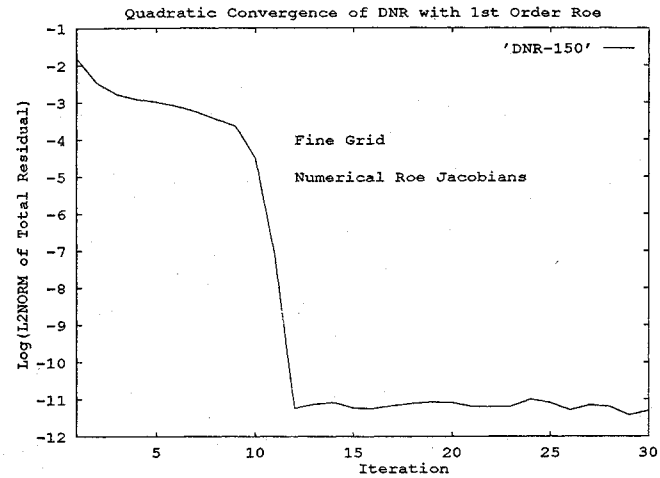

Fig. 12 Convergence of DNR algorithm with subsonic implicit characteristic varibale boundary conditions and first-order Roe with numerical Jacobians.

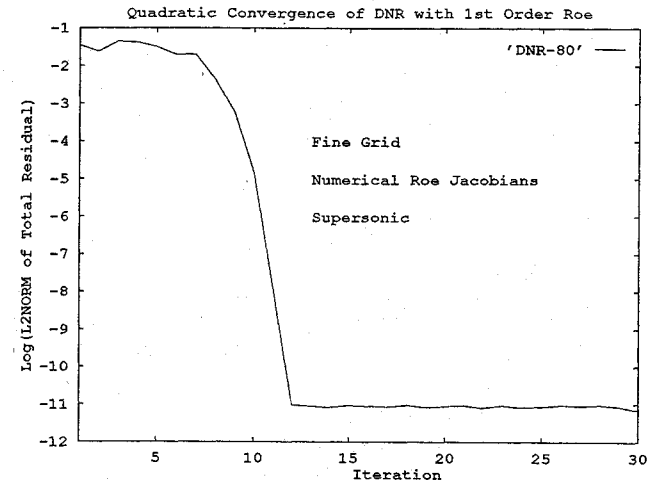

Fig. 13 Convergence of DNR algorithm with supersonic implicit characteristic varibale boundary conditions and first-order Roe with numerical Jacobians.

per iteration for the results compared in Fig. 10 and the CPU time that is required for the solution to converge to machine zero for each of these cases. Note that the DNR algorithm with 10 cycles per iteration was the best performer with the quadratic run taking approximately 500% longer. As with the direct solver results, the extra time needed to calculate the numerical Jacobians compared to using Steger-Warming analytical Jacobians is small compared to the total CPU time for the best performing case of the DNR algorithm with 10 cycles. Figure 11 compares the convergence performance of the NR vs the DNR algorithms with 150 cycles per iteration using first-order Steger-Warming fluxes and Jacobians. There is little difference between the convergence rates with either the Steger-Warming analytical Jacobians or Steger-Warming numerical Jacobians. This indicates that the method used for forming the numerical Jacobians is satisfactory.

It was also possible to obtain quadratic convergence for the $97 \times 17 \times 17$ fine grid which is illustrated in Fig. 12. It was found that to achieve quadratic convergence a few more iterations had to be run before setting $\Delta\tau$ to 1.0×10^{99}. For the first seven iterations $\Delta\tau$ was kept constant at 5000 for each computational cell. When this value was held for only the first four iterations quadratic convergence did not occur, but the convergence to machine zero took approximately the same number of iterations. Figure 13 shows the quadratic convergence of the DNR algorithm for a freestream Mach number of 1.50 and an angle of attack of 3.06 deg. Here, supersonic characteristic variable far-field boundary conditions were used. Until iteration number 5, $\Delta\tau$ was kept constant at 8000 at which time

it was increased to 1.0×10^{99}. Only 80 cycles per iteration were required to obtain quadratic convergence compared to 150 cycles with the subsonic implicit characteristic variable boundary conditions.

Conclusions

A discretization has been developed for first- and second-order spatially accurate solutions of the three-dimensional Euler equations. The advantage of this discretization is that the Jacobian terms are computed numerically to remain consistent with the numerical flux vector. Complex derivations of analytical Jacobians using symbolic mathematics software is not needed. The increased convergence rate with Jacobians that are consistent with the numerical flux far outweighs the extra CPU time needed to calculate the derivatives.

A two-pass factorization, modified two-pass factorization, NR, and DNR algorithms were compared, with the best performer being the DNR and NR algorithms. The DNR algorithm then was compared to the direct solver where it was observed that the Newton-relaxation approach can provide direct solver performance although needing only a fraction of the memory and CPU time.

When run with implicit boundary conditions the NR and DNR algorithms were found to be capable of quadratic convergence, without the need to sequence the perturbation used in the numerical differentiation. However, the overall best convergence performance vs CPU time was found to be when the DNR and NR algorithms were run with significantly less cycles per iteration than required for quadratic convergence. Although not the most efficient means of convergence, three-dimensional quadratic convergence demonstrates the robustness of the NR and DNR algorithms.

References

[1]Venkatakrishnan, V., "Newton Solution of Inviscid and Viscous Problems," *AIAA Journal*, Vol. 27, No. 7, 1989, pp. 885–891.

[2]Bailey, H. E., and Beam, R. M., "Newton's Method Applied to Finite-Difference Approximations for the Steady-State Compressible Navier–Stokes Equations," *Journal of Computational Physics*, Vol. 93, March 1991, pp. 108–127.

[3]Orkwis, P., "A Newton's Method Solver for the Two-Dimensional and Axisymmetric Navier–Stokes Equations," Ph.D. Dissertation, Dept. of Aerospace Engineering, North Carolina State Univ., Raleigh, NC, 1990.

[4]Whitfield, D. L., "Newton-Relaxation Schemes for Nonlinear Hyperbolic Systems," Engineering and Industrial Research Station Rept., MSSU-EIRS-ASE-90-3, Mississippi State Univ., Mississippi State, MS, Oct. 1990.

[5]Taylor, A. C., Ng, W., and Walters, R. W., "Upwind Relaxation Methods for the Navier–Stokes Equations Using Inner Iterations," *Journal of Computational Physics*, Vol. 99, No. 1, 1992, pp. 68–78.

[6]Steger, J. L., and Warming, R. F., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite-Difference Methods," *Journal of Computational Physics*, Vol. 40, No. 2, 1981, pp. 263–293.

[7]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372.

[8]Anderson, W. K., Thomas, J. L., and Van Leer, B., "A Comparison of Finite Volume Flux Vector Splittings for the Euler Equations," AIAA Paper 85-0122, Jan. 1985.

[9]Osher, S., and Chakravarthy, S., "Very High Order Accurate TVD Schemes," Institute for Computer Applications in Science and Engineering, ICASE Rept. No. 84-44, Sept. 1984.

[10]Dennis, J. E., and Schnabel, R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983, pp. 86–110.

[11]Ortega, J. M., and Rheinboldt, W. C., *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970, pp. 181–187.

[12]Van der Vorst, H. A., "The Performance of FORTRAN Implementations for Preconditioned Conjugate Gradient Methods on Vector Computers," *Parallel Computing*, Vol. 3, 1986, pp. 49–58.

[13]Saad, Y., and Schultz, M. H., "Parallel Implementations of Preconditioned Conjugate Gradient Methods," Research Report 425, Dept. of Computer Science, Yale Univ., New Haven, CT, 1985.

[14]Yoon, S., and Kwak, D., "Implicit Navier–Stokes Solver for Three-Dimensional Compressible Flows," *AIAA Journal*, Vol. 30, No. 11, 1992, pp. 2653–2659.

[15]Vanden, K. J., "Direct and Iterative Algorithms for the Three-Dimensional Euler Equations," Ph.D. Dissertation, Dept. of Aerospace Engineering, Mississippi State Univ., Mississippi State, MS, Dec. 1992.

[16]Schmitt, V., and Charpin, F., "Pressure Distributions On The Onera-M6-Wing At Transonic Mach Numbers," AGARD Rept. AR-138, May 1979.

[17]Van Leer, B., "Towards the Ultimate Conservative Difference Scheme, II. Monotonicity and Conservation Combined in a Second-Order Scheme," *Journal of Computational Physics*, Vol. 14, No. 2, 1974, pp. 361–376.

[18]Barth, T. J., "Analysis of Implicit Local Linearization Techniques for Upwind and TVD Algorithms," AIAA Paper 87-0595, Jan. 1987.

[19]Knight, D., "A Fully Implicit Navier–Stokes Algorithm Using an Unstructured Grid and Flux Difference Splitting," AIAA Paper 93-0875, Jan. 1983.

[20]Orkwis, P. D., and Vanden, K. J., "On the Accuracy of Numerical Versus Analytical Jacobians," AIAA Paper 94-0176, Jan. 1994.