IMPLICIT SOLUTION METHODS IN COMPUTATIONAL FLUID DYNAMICS

Thomas H. PULLIAM

CFD Branch, NASA Ames Research Center, Moffett Field, CA 94035, U.S.A.

A class of implicit finite difference schemes of the Beam and Warming approximate factorization type will be addressed. The development and analysis of various aspects of this class of schemes will be given along with the motivations behind many of the choices. Various acceleration and efficiency modifications such as matrix reduction, diagonalization and flux split schemes will be presented. The methods are demonstrated in fully vectorized codes for a CRAY type architecture. The emphasis will be on the Euler equations in generalized coordinates.

1. Introduction

A wide variety of numerical techniques are in use today. Some have developed to a high enough level to be used in production codes (see [1-6]) while other techniques (for example TVD schemes [7-9]) are just now entering the research code realm. In this presentation we shall concentrate on methods and techniques which have been applied to various computational fluid flow problems. These include, implicit finite differences, central space differencing, upwind differencing, approximate factorization, nonlinear dissipation models, and various acceleration techniques for steady-state and time accurate computations.

The development employed here is based on the implicit approximate factorization algorithm of Beam and Warming [1]. A particular application in two dimensions was first presented by Steger [2] and for three dimensions by Pulliam and Steger [3]. Concurrent with this work has been the paralleled development and application of MacCormack's method [4]. I shall concentrate here on the theoretical development, application and assessment of the implicit algorithm which at this stage has produced two codes, ARC2D a two-dimensional version and ARC3D the three-dimensional code.

There are a number of considerations to weigh when choosing a numerical algorithm to apply to a set of partial differential equations. If we restrict ourselves to finite difference schemes then the possibilities are narrowed somewhat to the two classical approaches for time integration, explicit and implicit techniques. The merits of either of these two have been extensively discussed in the literature. Explicit methods typically require less computational work and are simpler both in derivation and application. Implicit methods, while computationally expensive, have less stringent stability bounds (classical stability analysis shows unconditional stability but in practice on nonlinear problems bounds are encountered).

Implicit numerical schemes are usually chosen because we wish to obtain solutions which require fine grid spacing for numerical resolution, and we do not want to limit the time steps by employing a conditionally stable explicit scheme. Explicit schemes are very useful and schemes such as MacCormack's explicit algorithm [4] are in wide use today. The extra work required for an implicit scheme is usually offset by the advantages obtained by the increased stability limits, and in general implicit schemes have been very useful and successful for a variety of inviscid and viscous flowfield calculations.
With the advent of high-speed vector processors one must also consider the degree to which a certain algorithm can be vectorized when choosing a scheme. As a rule explicit schemes are more easily vectorized than implicit schemes. But implicit schemes can be fully vectorized. This requires though a substantial amount of temporary storage and a commitment to the details of data management, see for instance, [10].

Another consideration is the question of time accuracy versus non-time-accurate steady-state iteration. For unsteady, transient problems we wish to employ time-accurate methods, initialize the flow with some realizable state and integrate forward in time with time steps commensurate with the unsteady phenomenon which is being calculated. Both implicit and explicit methods are capable of computing time accurately. In steady-state calculation we wish to integrate from some arbitrary state to the asymptotic solution in any manner which will get us there in the least amount of computational work. Non-time-accurate techniques (for instance relaxation methods, variable time steps, matrix preconditioning, large time steps) can be employed as long as they are convergent and do not distort the steady-state equations so as to produce inaccurate results. The methods presented below can be employed either for time-accurate calculations or for steady-state rapidly convergent solutions.

The algorithm to be presented is an implicit approximate factorization finite difference scheme which can be either first- or second-order accurate in time. Local time linearizations are applied to the nonlinear terms and an approximate factorization of the two-dimensional implicit operator is used to produce locally one-dimensional operators. This results in block tridiagonal matrices, which are easy to solve. The spatial derivative terms are approximated with second-order central differences. Explicit and implicit artificial dissipation terms are added to achieve nonlinear stability. A spatially variable time step is used to accelerate convergence for steady-state calculations. A diagonal form of the algorithm is also discussed, which produces a computationally efficient modification of the standard algorithm where the diagonalization results in scalar tridiagonal or pentadiagonal operators in place of the block operators. This diagonal form of the algorithm produces a robust, rapid and versatile scheme for steady-state calculations. We also discuss the details of a matrix reduction scheme, due to Barth and Steger [11] where the block matrices of the standard implicit scheme are reduced to sets of lower-rank matrices (e.g. two scalars and a $2 \times 2$ in 2-D).

2. The Euler equations

The starting point is the strong conservation law form of the two-dimensional Euler equations in generalized curvilinear coordinates. The strong conservation law form is chosen because we wish to admit shock capturing.

The Euler equations were transformed from Cartesian coordinates to general curvilinear coordinates where

$$
\tau = t, \quad \xi = \xi(x, y, t), \quad \eta = \eta(x, y, t).
$$

The coordinate transformation introduced here follows the development of Viviand [12] and Vinokur [13].

The transformations are chosen so that the grid spacing in the curvilinear space is uniform and of unit length, see Fig. 1. This produces a computational space $\xi$ and $\eta$ which is a rectangular domain and which has a regular uniform mesh so that standard unweighted differencing schemes can be used in the numerical formulation. The original Cartesian space
will be referred to as the physical domain. There will be a one-to-one correspondence between a physical point in space and a computational point except for regions where there are singularities or cuts due to the topology. In those cases it may be necessary to map one physical point to many computational points (this usually occurs at computational boundaries). With this construction we can produce one computational code for a wide variety of physical geometries and grid systems.

The Euler equations in nondimensional form are

$$\partial_t \hat{Q} + \partial_\xi \hat{E} + \partial_\eta \hat{F} = 0$$

(2.1)

where

$$\hat{Q} = J^{-1} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix}, \quad \hat{E} = J^{-1} \begin{bmatrix} \rho U \\ \rho uU + \xi_x p \\ \rho vU + \xi_y p \\ U(e + p) - \xi_t p \end{bmatrix}, \quad \hat{F} = J^{-1} \begin{bmatrix} \rho V \\ \rho uV + \eta_x p \\ \rho vV + \eta_y p \\ V(e + p) - \eta_t p \end{bmatrix}$$

(2.2a)

with

$$U = \xi_t + \xi_x u + \xi_y v, \quad V = \eta_t + \eta_x u + \eta_y v$$

(2.2b)

the contravariant velocities.

Pressure is related to the conservative flow variables, $Q$, by the equation of state

$$p = (\gamma - 1)(e - \frac{1}{2} \rho (u^2 + v^2))$$

(2.3)

where $\gamma$ is the ratio of specific heats, generally taken as 1.4. The speed of sound is $a$ which for ideal fluids is, $a^2 = \gamma p / \rho$.

The choice of nondimensional parameters is arbitrary. Here we have chosen to scale the
variables \( \rho \) (density), \( u, v \) (the Cartesian velocities), and \( e \) (the total energy) as

\[
\begin{align*}
\tilde{\rho} &= \frac{\rho}{\rho_\infty}, \\
\tilde{u} &= \frac{u}{a_\infty}, \\
\tilde{v} &= \frac{v}{a_\infty}, \\
\tilde{e} &= \frac{e}{\rho_\infty a_\infty^2}
\end{align*}
\]  
(2.4)

where \( \infty \) refers to free stream quantities. Assuming a reference length, \( l \) (usually taken as some characteristic physical length such as chord of an airfoil), time \( t \) scales as \( t = \frac{t}{l} \). For the remainder of this development the - will be dropped for simplicity.

The metric terms are defined as

\[
\begin{align*}
\xi_x &= Jy_y, & \xi_y &= -Jx_x, & \xi_t &= -x_x \xi_x - y_y \xi_y, \\
\eta_x &= -Jy_x, & \eta_y &= Jx_x, & \eta_t &= -x_x \eta_x - y_y \eta_y,
\end{align*}
\]  
(2.5)

where \( J^{-1} = \left( x_x y_y - x_y y_x \right) \) is defined to be the metric Jacobian.

3. Numerical algorithm

3.1. Implicit time differencing

Consider an implicit three-point time differencing scheme of the form [14]

\[
\begin{align*}
\Delta \hat{Q}^n &= \frac{\partial}{1 + \varphi} \frac{\partial}{\partial t} (\Delta \hat{Q}^n) + \frac{\Delta t}{1 + \varphi} \frac{\partial}{\partial t} \hat{Q}^n + \frac{\varphi}{1 + \varphi} \Delta \hat{Q}^{n-1} + O[(\vartheta - \frac{1}{2} - \varphi) \Delta t^2 + \Delta t^3]
\end{align*}
\]  
(3.1)

where \( \Delta \hat{Q}^n = \hat{Q}^{n+1} - \hat{Q}^n \) and \( \hat{Q}^n = \hat{Q}(n \Delta t) \). The parameters \( \vartheta \) and \( \varphi \) can be chosen to produce different schemes of either first- or second-order accuracy in time.

For \( \vartheta = 1 \) and \( \varphi = 0 \), we have the first-order Euler implicit scheme, and for \( \varphi = \frac{1}{2} \), the three-point implicit scheme.

Let us restrict ourselves to the first-order in time scheme (although all of the subsequent development can easily be extended to any second-order scheme formed from (3.1)). Applying (3.1) to (2.1), results in

\[
\hat{Q}^{n+1} - \hat{Q}^n + \hat{h}(\hat{E}^{n+1}_\xi + \hat{F}^{n+1}_\eta) = 0
\]  
(3.2)

where \( h = \Delta t \).

3.2. Local time linearizations

We wish to solve (3.2) for \( \hat{Q}^{n+1} \) given \( \hat{Q}^n \). The flux vectors \( \hat{E} \) and \( \hat{F} \) are nonlinear functions of \( \hat{Q} \) and therefore (3.2) is nonlinear in \( \hat{Q}^{n+1} \). The nonlinear terms are linearized in time about \( \hat{Q}^n \) by a Taylor series such that

\[
\begin{align*}
\hat{E}^{n+1} &= \hat{E}^n + \hat{A}^n \Delta \hat{Q}^n + O(h^2), \\
\hat{F}^{n+1} &= \hat{F}^n + \hat{B}^n \Delta \hat{Q}^n + O(h^2)
\end{align*}
\]  
(3.3)

where \( \hat{A} = \partial \hat{E} / \partial \hat{Q} \) and \( \hat{B} = \partial \hat{F} / \partial \hat{Q} \) are the flux Jacobians and \( \Delta \hat{Q}^n \) is \( O(h) \).

Note that the linearizations are second-order accurate and so if a second-order time scheme had been chosen the linearizations would not degrade the time accuracy.
The Jacobian matrices are

\[
\hat{A} \text{ or } \hat{B} = 
\begin{bmatrix}
\kappa_t & \kappa_x & \kappa_x & \kappa_x & 0 \\
-u\theta + \kappa_x \phi^2 & \kappa_t + \theta - (\gamma - 2) \kappa_x u & \kappa_x u - (\gamma - 1) \kappa_x v & (\gamma - 1) \kappa_x \\
-\theta + \kappa_x \phi^2 & \kappa_x v - (\gamma - 1) \kappa_x u & \kappa_x u - (\gamma - 1) \kappa_x v & (\gamma - 1) \kappa_x \\
\theta [\phi^2 - a_1] & \kappa_x a_1 - (\gamma - 1) u \theta & \kappa_x a_1 - (\gamma - 1) u \theta & \gamma \theta + \kappa_x \\
\end{bmatrix}
\]

(3.4)

with \( a_1 = \gamma (c/\rho) - \phi^2 \), \( \theta = \kappa_x u + \kappa_x v \), \( \phi^2 = \frac{1}{2} (\gamma - 1)(u^2 + v^2) \), and \( \kappa = \xi \) or \( \eta \) for \( \hat{A} \) or \( \hat{B} \), respectively.

Applying (3.3) to (3.2) and combining the \( \Delta \hat{Q}^n \) terms produces the 'delta form' of the algorithm

\[
[I + h \partial_\xi \hat{A}^n + h \partial_\eta \hat{B}^n] \Delta \hat{Q}^n = -h(\partial_\xi \hat{E}^n + \partial_\eta \hat{E}^n).
\]

(3.5)

This is the unfactored form of the block algorithm. We shall call the right-hand side of (3.5) the 'explicit' part and the left-hand side the 'implicit' part of the algorithm.

3.3. Space differencing

The next step is to take the continuous differential operators \( \partial_\xi \) and \( \partial_\eta \) and approximate them with finite difference operators on a discrete mesh.

Introducing a grid of mesh points \((j, k)\), variables are defined at mesh points as

\[
u_{j,k} := u(j\Delta\xi, k\Delta\eta).
\]

(3.6)

The grid spacing in the computational domain is chosen to be unity so that

\[
\Delta\xi = 1 \quad \text{and} \quad \Delta\eta = 1.
\]

(3.7)

Second-order central difference operators can be used where for example,

\[
\delta_\xi u_{j,k} = \frac{1}{2} (u_{j+1,k} - u_{j-1,k}) \quad \text{and} \quad \delta_\eta u_{j,k} = \frac{1}{2} (u_{j,k+1} - u_{j,k-1}).
\]

(3.8)

The choice of the type and order of the spatial differencing is important both in terms of accuracy and stability. In most applications second-order accuracy has proven to be sufficient provided the grid resolution is reasonable. The choices for differencing type include central and upwind operators. These choices are dictated by stability, and in the next section we discuss what motivates certain choices.

3.4. Stability analysis of difference forms

The choice of the type of difference forms to use for the Euler equations can be justified by a linear stability analysis. For simplicity, let us examine a one-dimensional coupled system of linear constant coefficient equations of the form

\[
Q_t + A Q_x = 0
\]

(3.9)

where \( A \) is analogous to the flux Jacobian matrix. Assume that \( A \) has a complete set of real eigenvalues and eigenvectors (a property that the Euler flux Jacobians have) then

\[
\Lambda = X^\dagger A X.
\]

(3.10)
Multiplying (3.9) by $X^{-1}$ and combining terms using (3.10) we have

$$X^{-1}Q + X^{-1}AXX^{-1}Q_x = W_t + AW_x = 0 \quad (3.11)$$

with $W = X^{-1}Q$. Since $A$ is linear and constant the eigenvector matrix $X^{-1}$ can be brought through the derivatives.

The resulting system is now uncoupled and we can examine the representative model equation

$$w_t + \lambda w_x = 0 \quad (3.12)$$

where $\lambda$ represents an eigenvalue of $A$.

We shall apply different finite difference approximations for the spatial derivative and use Fourier analysis to determine conditions on $\lambda$ for stability.

If the second-order central difference operator is applied to the model equation one gets

$$w_j + \lambda(w_{j+1} - w_{j-1})/(2 \Delta x) = 0 \quad (3.13)$$

where $j$ is the spatial index. This is the ODE (ordinary differential equation) or semi-discrete approach to the analysis, since now we are dealing with a system of ODEs.

Classical Fourier analysis can be performed by assuming periodic boundary conditions and a solution of the form

$$w(x, t) = e^{\alpha t} e^{i \beta_j \Delta x} \quad (3.14)$$

with $i = \sqrt{-1}$ and $x = j \Delta x$.

Substituting this into (3.12) yields

$$\alpha e^{\alpha t} e^{i \beta_j \Delta x} + \lambda(e^{\alpha t} e^{i \beta(j+1) \Delta x} - e^{\alpha t} e^{i \beta(j-1) \Delta x})/(2 \Delta x) = 0 \quad (3.15)$$

The stability of the ODE is dependent on the sign of $\Re(\alpha)$ (the real part). Obviously, if $\Re(\alpha) > 0$ then $w(x, t)$ will grow unboundedly with time.

For (3.15)

$$\alpha = -\lambda(e^{i \beta \Delta x} - e^{-i \beta \Delta x})/(2 \Delta x) = -\lambda i \sin(\beta \Delta x)/\Delta x \quad (3.16)$$

Since $\alpha$ is pure imaginary ($\Re(\alpha) = 0$) the scheme is stable in the ODE sense independent of the sign of $\lambda$.

If one-sided difference formulas are employed, conditions on $\lambda$ arise. For simplicity, let us consider first-order one-sided differences.

Applying forward differencing to the model (3.12) gives

$$(w_j)_t + \lambda(w_{j+1} - w_j)/\Delta x = 0 \quad (3.17)$$

Fourier analysis produces

$$\alpha + \lambda(e^{i \beta \Delta x} - 1)/\Delta x = 0 \quad (3.18)$$

so that,

$$\alpha = \lambda(1 - e^{i \beta \Delta x})/\Delta x = \lambda[1 - \cos(\beta \Delta x) + i \sin(\beta \Delta x)]/\Delta x \quad (3.19)$$

Since $\cos(\beta \Delta x)$ is bounded by 1, $\Re(\alpha)$ will be less than zero if $\lambda < 0$. So for forward spatial differencing $\lambda$ must be less than zero for stability. A similar argument for first-order backward differencing shows that $\lambda > 0$ for stability. It can be shown that for higher-order one-sided differences the stability requirements on $\lambda$ remain the same.
These results have a direct application to the choice of differencing for the Euler equations. As we shall see below the inviscid flux Jacobians have eigenvalues (equivalent to \( \lambda \)) with both positive and negative sign. In their basic form the only stable spatial differencing is central differencing, but as we shall see when flux splitting is used or when the eigenvalues can be restricted to one sign then upwind differencing can be employed. A class of upwind schemes shall be discussed in Section 4.

3.5. Matrix form of unfactored algorithm

We now turn to examining the matrices we get when differencing formulas are applied to the implicit algorithm. It is always instructive to examine the matrix structure of any finite difference equation. With the application of central differences to (3.5) it is easy to show that the implicit algorithm produces a large banded system of algebraic equations. Let the mesh size in \( \xi \) be \( J_{\text{max}} \) and in \( \eta \) \( K_{\text{max}} \). Then the banded matrix is a \( (J_{\text{max}} \cdot K_{\text{max}} \cdot 4) \times (J_{\text{max}} \cdot K_{\text{max}} \cdot 4) \) square matrix of the form

\[
\begin{bmatrix}
[ ] & [ ] & [ ] & [B] & [-A] \\
[ ] & [ ] & [ ] & [ ] & [B] \\
\end{bmatrix}
\]

where the variables have been ordered with \( j \) running first and then \( k \).

The matrix is sparse but it would be very expensive (computationally) to solve the algebraic system. For instance, for a reasonable two-dimensional calculation of transonic flow past an airfoil we could use approximately 80 points in the \( \xi \) direction and 40 points in the \( \eta \) direction. The resulting algebraic system has a \( 12,800 \times 12,800 \) matrix problem to be solved and although we could take advantage of its banded sparse structure it would still be very costly in both CPU time and storage.

3.6. Approximate factorization

As we have seen, the integration of the full two-dimensional operator is too expensive. One way to simplify the solution process is to introduce an approximate factorization of the two-dimensional operator into two one-dimensional operators. The implicit side of (3.5) can be written as

\[
[1 + h\delta_{\xi}\tilde{A}^n + h\delta_{\eta}\tilde{B}^n] \Delta \hat{Q}^n = [1 + h\delta_{\xi}\tilde{A}^n][1 + h\delta_{\eta}\tilde{B}^n] \Delta \hat{Q}^n + h^2\delta_{\xi}\tilde{A}^n\delta_{\eta}\tilde{B}^n \Delta \hat{Q}^n.
\]

The cross terms \( (h^2 \text{ terms}) \) are second-order in time errors since \( \Delta \hat{Q}^n \) is \( O(h) \). They can
therefore be neglected without degrading the time accuracy of any second-order scheme which we may choose.

The resulting factored form of the algorithm is

$$[I + h\delta_\xi \hat{A}^n][I + h\delta_\eta \hat{B}^n] \Delta \hat{Q}^n = -h[\delta_\xi \hat{E}^n + \delta_\eta \hat{F}^n].$$  \hspace{1cm} (3.22)

We now have two implicit operators each of which is block tridiagonal. The structure of the block tridiagonal matrix is

$$\begin{bmatrix}
I & I & I & \cdots & I & I & I \\
I & I & I & \cdots & I & I & I \\
& I & I & \cdots & I & I & I \\
& & & \ddots & \cdots & \cdots & \cdots \\
& & & & \cdots & I & I \\
& & & & & I & I \\
& & & & & & I
\end{bmatrix} 
$$  \hspace{1cm} (3.23)

The solution algorithm now consists of two one-dimensional sweeps, one in the $\xi$ and one in the $\eta$ direction. The block matrix size is now at most $(\max[J_{\max}, K_{\max}] - 4) \times (\max[J_{\max}, K_{\max}] - 4)$. Each step requires the solution of a linear system involving a block tridiagonal which is solved by block LUD (lower-upper decomposition). The resulting solution process is much more economical than the unfactored algorithm in terms of computer storage and CPU time.

3.7. Reduced forms of the implicit algorithm

Even though the factorization has improved the efficiency of the block implicit algorithm the major expense of the implicit scheme still resides in the block tridiagonal inversions. Compared to standard explicit algorithms the implicit scheme is still computationally expensive. The increased stability bounds of the implicit scheme offsets some of this disadvantage especially for problems where refined grids are used. In general this holds true for time-accurate applications where refinement would unduly restrict the time steps for explicit schemes, but recent developments in multigrid techniques (see Jespersen [15] for a review), applied to steady-state problems requires us to reexamine the implicit schemes. One way to capture back the advantage is to make the implicit scheme less computationally expensive, we will discuss other ways, such as accelerated convergence and improved accuracy, in later sections.

To improve the efficiency of a numerical scheme we can modify or simplify the algorithm so that the computational work is decreased. Most of the computational work for the implicit algorithm is tied to the block tridiagonal solution process. One way to reduce that work would be to reduce the block size for the tridiagonals. This can be accomplished by reducing the equation set from four variables (density, $x$-momentum, $y$-momentum, and energy) to three variables (density and the two momentums) by assuming constant total enthalpy, $H = (e + p)/\rho = H_0$ or similar thermodynamic approximations.

3.7.1. Diagonal form

The computational work can be decreased by introducing a diagonalization of the blocks in the implicit operators as developed by Pulliam and Chaussee [16]. The eigensystem of the flux Jacobians $\hat{A}$ and $\hat{B}$ are used in this construction.
The flux Jacobians $\hat{A}$ and $\hat{B}$ each have real eigenvalues and a complete set of eigenvectors. Therefore, the Jacobian matrices can be diagonalized [17, 18],

$$
\Lambda_\xi = T^{-1}_\xi \hat{A} T_\xi \quad \text{and} \quad \Lambda_\eta = T^{-1}_\eta \hat{B} T_\eta
$$

(3.24)

with $T_\xi$ the matrix whose columns are the eigenvectors of $\hat{A}$ and $T_\eta$ the corresponding eigenvector matrix for $\hat{B}$. They are written out in the Appendix.

Here we take the factored algorithm in delta form, (3.22), and replace $\hat{A}$ and $\hat{B}$ with their eigensystem decompositions.

$$
[T_\xi T^{-1}_\xi + h\delta_\xi (T_\xi^{-1} A_\xi T_\xi^{-1})][T_\eta T^{-1}_\eta + h\delta_\eta (T_\eta^{-1} A_\eta T_\eta^{-1})] \Delta Q^n
$$

(3.25)

$$
= \text{the explicit right-hand side of (3.22)} = \hat{R}^n.
$$

At this point (3.22) and (3.25) are equivalent. A modified form of (3.25) can be obtained by factoring the $T_\xi$ and $T_\eta$ eigenvector matrices outside the spatial derivative terms $\delta_\xi$ and $\delta_\eta$. The eigenvector matrices are functions of $\xi$ and $\eta$ and therefore this modification reduces the time accuracy to at most first-order in time. The resulting equations are

$$
T_\xi [I + h\delta_\xi \Lambda_\xi] \hat{N} [I + h\delta_\eta \Lambda_\eta] T^{-1}_\eta \Delta \hat{Q}^n = \hat{R}^n
$$

(3.26)

where $\hat{N} = T^{-1}_\xi T^{-1}_\eta$, see Appendix.

The explicit side of the diagonal algorithm (the steady-state finite difference equations) is exactly the same as in the original algorithm, (3.22). The modifications are restricted to the implicit side and so if the diagonal algorithm converges, the steady-state solution will be identical to one obtained with the unmodified algorithm. In fact, linear stability analysis would show that the diagonal algorithm has exactly the same unconditional stability as the original algorithm. (This is because the linear stability analysis assumes constant coefficients and diagonalizes the blocks to scalars, the diagonal algorithm then reduces to the unmodified algorithm.) The modification (pulling the eigenvector matrices outside the spatial derivatives) of the implicit operator does affect the time accuracy of the algorithm. It reduces the scheme to at most first-order in time and also gives time-accurate shock calculations a nonconservative feature, i.e., errors in shock speeds and shock jumps. But, remember that the steady-state is fully conservative since the steady-state equations are unmodified. Also, computational experiments by Pulliam and Chaussee [16] have shown that the convergence and stability limits of the diagonal algorithm are similar to that of the unmodified algorithm.

The diagonal algorithm reduces the block tridiagonal inversion to $4 \times 4$ matrix multiplies and scalar tridiagonal inversions. The operation count associated with the implicit side of the full block algorithm is 410 multiplies, 356 adds, and 10 divides, a total of 776 operations, while the diagonal algorithm requires 233 multiplies, 125 adds, and 26 divides or 384 operations. Adding in the explicit side and other overhead such as I/O (input/output) and initialization, the overall savings in computational work can be as high as 40%. In fact the computational work can be further decreased by noting that the first two eigenvalues of the system are identical (see Appendix). This allows us to combine the coefficient calculations and part of the inversion work for the first two scalar operators.

3.7.2. Pressure–velocity splitting

Another way is to reduce the block size by similarity transformations as proposed by Steger [18]. Until recently this was restricted to Cartesian variables. Recent work by Barth and
Steger [11] has removed some of this restriction and they developed an algorithm where two scalar tridiagonals and one block two-by-two tridiagonal inversion is required. The basic concept can be demonstrated in two-dimensional Cartesian coordinates (see [11] for the extension to generalized coordinates).

The development of the sound speed–velocity splitting begins by considering the nonconservative form of the Euler equations

\[ \partial_t R + M \partial_x R + N \partial_y R = 0 \]  

where

\[ R = \begin{pmatrix} \rho \\ u \\ v \\ \rho \end{pmatrix}, \quad M = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & \rho^{-1} \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{pmatrix}, \quad N = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & \rho \end{pmatrix}. \]  

The eigenvalues of coefficient matrices, \( M \) and \( N \), are the usual characteristic speeds

\[ \Lambda_M = [u, u, u + c, u - c], \quad \Lambda_N = [v, v, v - c, v + c]. \]  

These coefficient matrices can each be split into two submatrices: one contains all velocity eigenvalues and the other contains sound speed eigenvalues. A particular matrix splitting (there are many possibilities) was chosen to satisfy the following conditions

\[ \Lambda(M) = \Lambda(M_u) + \Lambda(M_c), \quad \Lambda(M_u) = (u, u, u, u), \quad \Lambda(M_c) = (0, 0, c, -c), \]  

\[ \Lambda(N) = \Lambda(N_v) + \Lambda(N_c), \quad \Lambda(N_v) = (v, v, v, v), \quad \Lambda(N_c) = (0, 0, c, -c). \]  

Specifically, \( M \) and \( N \) are split as

\[ M = M_u + M_c = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & 0 \\ 0 & 0 & u & 0 \\ 0 & 0 & 0 & u \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho^{-1} \\ 0 & 0 & 0 & 0 \\ 0 & \gamma p & 0 & 0 \end{pmatrix}, \]  

\[ N = N_v + N_c = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & 0 \\ 0 & 0 & 0 & v \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho^{-1} \\ 0 & \gamma p & 0 & 0 \end{pmatrix}. \]  

Given the coefficient matrices \( M \) and \( N \), a similarity transformation exists that transforms these matrices into their conservative counterparts, the flux Jacobians \( A \) and \( B \). \( A = SMS^{-1} \), \( B = SNS^{-1} \) where \( S = \partial Q/\partial R \). Using this similarity transformation, \( M_c \) and \( N_c \) transform to \( A_c = SM_cS^{-1} \) and \( B_c = SN_cS^{-1} \) written out as

\[ A_c = (\gamma - 1) \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2}(u^2 + v^2) & -u & -v & 1 \\ 0 & 0 & 0 & 0 \\ a_{41}^c & a_{42}^c & -uv & u \end{pmatrix}, \]  

\[ B_c = (\gamma - 1) \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2}(u^2 + v^2) & -u & -v & 1 \\ 0 & 0 & 0 & 0 \\ b_{41}^c & b_{43}^c & -uv & v \end{pmatrix}. \]
where
\[ a^c_{41} = [\frac{1}{2}u(u^2 + v^2)] - yup/[\rho(\gamma - 1)^2], \quad a^c_{42} = yp/[\rho(\gamma - 1)^2] - u^2, \]
\[ b^c_{41} = [\frac{1}{2}(u^2 + u^2)] - yuv/[\rho(\gamma - 1)^2], \quad b^c_{43} = yp/[\rho(\gamma - 1)^2] - v^2, \]

while \( A_u \) and \( B_v \) are
\[
A_u = A - A_c, \quad B_v = B - B_c. \quad (3.31)
\]
This splitting produces matrices \( A_u \) and \( B_v \) that are more complex than \( A \) and \( B \). But it is found that \( Q \) is an eigenvector of \( A_u \) and \( B_v \), i.e.
\[
A_u Q = uQ, \quad B_v Q = vQ \quad (3.32a)
\]
which motivates the following substitution
\[
A Q = (uI + A_c)Q, \quad B Q = (vI + B_c)Q. \quad (3.32b)
\]
Insertion of (3.32b) into the equations for local linearization of the Jacobians, the Cartesian equivalent of (3.3), produces
\[
E^{n+1} = E^n + (uI + A_c)^n(Q^{n+1} - Q^n), \quad (3.33a)
\]
\[
F^{n+1} = F^n + (vI + B_c)^n(Q^{n+1} - Q^n). \quad (3.33b)
\]
Utilizing these linearizations in the basic algorithm equation (3.22) yields
\[
L_x L_y \Delta Q = -\Delta t[\delta_x E^n + \delta_y F^n] \quad (3.34a)
\]
with
\[
L_x = [I + \theta \Delta t \delta_x (uI + A_c)^n], \quad L_y = [I + \theta \Delta t \delta_y (vI + B_c)^n]. \quad (3.34b,c)
\]
The end result of this splitting is that the new operators \( L_x \) and \( L_y \) form matrices that no longer require \( 4 \times 4 \) block tridiagonal inversions. In matrix operator form, we have
\[
L_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \theta \Delta t \delta_x \begin{bmatrix} u & 0 & 0 & 0 \\ u + a^c_{21} & a^c_{22} & a^c_{23} & a^c_{24} \\ 0 & 0 & u & 0 \\ a^c_{41} & a^c_{42} & a^c_{43} & u + a^c_{44} \end{bmatrix}, \quad (3.35a)
\]
\[
L_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \theta \Delta t \delta_y \begin{bmatrix} v & 0 & 0 & 0 \\ 0 & v & 0 & 0 \\ b^c_{31} & b^c_{32} & v + b^c_{33} & b^c_{34} \\ b^c_{41} & b^c_{42} & b^c_{43} & v + b^c_{44} \end{bmatrix}, \quad (3.35b)
\]
where \( a^c \) and \( b^c \) are the respective elements of \( A_c \) and \( B_c \) given by (3.30).

In the \( L_x \) operator, for example, the first and third rows decouple from the system and can be solved as scalar tridiagonal matrices with their respective right-hand sides. Once these rows are solved, the elements of the first and third columns can be moved to the right-hand side. The second and fourth equations remain coupled and are solved as a \( 2 \times 2 \) block tridiagonal matrix. The block decoupling of the \( L_y \) operator is even more conspicuous and is inverted (i.e., solved for) in a similar manner.

The use of the pressure–velocity splitting has substantially reduced the computational work over the basic block implicit scheme. A typical \( 2 \times 2 \) block tridiagonal requires 55 operations per point, so the overall inversion, including the two scalar tridiagonals, requires 73 operations.
per entry. Because the two scalar tridiagonals have identical coefficients, this work can be even further cut by solving them together.

The matrix splitting produces the flux vectors.

\[ E = AQ = uIQ + A_cQ = E_u + E_c, \quad F = BQ = vIQ + B_cQ = F_v + F_c \]  

(3.36)

where

\[ E_u = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho u v \\ \rho u v \\ \rho v u \\ \rho v^2 \end{pmatrix}, \quad E_c = \begin{pmatrix} 0 \\ p \\ 0 \\ up \end{pmatrix}, \quad F_v = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 \\ \rho v e \end{pmatrix}, \quad F_c = \begin{pmatrix} 0 \\ p \\ 0 \end{pmatrix}. \]

Note that the Jacobians of \( E_c \) and \( F_c \) are not \( A_c \) and \( B_c \) as defined above. Usually, the use of implicit linearizations which are not the Jacobians of the explicit flux vectors leads to restricted stability bounds or unconditionally instability. Linear stability analysis presented by Barth and Steger, as well as numerical experiments have shown though that the use of \( A_c \) and \( B_c \) leads to unconditional stability.

The generalized coordinate form of pressure–velocity splitting is developed in [11]. A rotation transformation is used to align the momentum equations with generalized coordinate directions, e.g. in the \( \xi \) direction they use

\[ C_\xi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \xi_2/L_1 & \xi_3/L_1 & 0 \\ 0 & -\xi_2/L_1 & \xi_3/L_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \]  

(3.37)

with \( L_1 = \sqrt{\xi_x^2 + \xi_y^2} \). This produces the transformed splitting matrix

\[ \hat{A}_c = (\gamma - 1) \begin{pmatrix} \frac{1}{2}L_1(u^2 + v^2) & 0 & 0 & U \\ 0 & 0 & 0 & L_1 \\ a_{41}^c & a_{42}^c & -UV/L_1 & U \end{pmatrix} \]  

(3.38)

where

\[ a_{41}^c = U \left[ \frac{1}{2}(u^2 + v^2) - \frac{c^2}{(\gamma - 1)^2} \right], \quad a_{42}^c = L_1 \frac{c^2}{(\gamma - 1)^2} = \frac{U^2}{L_1} \]

with \( \hat{U} = \eta_u u - \eta_v v \).

The structure of (3.38) is identical to (3.30) so that the implicit operators in generalized coordinates are again reducible to 2 scalars and one \( 2 \times 2 \) block operator for each direction. Barth and Steger also discuss the application of pressure–velocity splitting to the Navier–Stokes equations.

4. Artificial dissipation added to implicit schemes

Even though linear stability analysis shows unconditional stability for the implicit algorithm, in practice stability bounds are encountered. This is especially true in strongly nonlinear cases such as flows with shocks.
Whenever discrete methods are used to ‘capture’ shocks (as opposed to fitting them), or to compute high Reynolds number viscous behavior, scales of motion appear which cannot be resolved by the numerics. These can be brought about by the nonlinear interactions in the convection terms of the momentum equations. If scale is represented by wave length or frequency, it can be easily shown that two waves interact as products to form a wave of higher frequency (the sum of the original two) and one of lower frequency (the difference). The lower frequencies do not cause a problem, but the continual cascading into higher and higher frequencies does. It is accounted for physically by shock formation (the harmonic analysis of a discontinuity contains all frequencies) or by viscous dissipation of the very high wave numbers. In numerical computations it can not be ignored and must be accounted for in the algorithm constructed. In any finite discrete mesh the cascading frequencies can eventually exceed the capacity of the mesh resolution at which point they can either (a) alias back into the lower frequencies or (b) pile up at the higher frequency side. In either case, if uncontrolled, these terms can lead to serious inaccuracies and possible numerical instability.

The most common way of coping with the high-frequency cascade is to add to the complete algorithm some form of numerical dissipation with an error level that does not interfere with the accuracy of any physical viscous effects. This can be done in a variety of ways.

4.1. Constant coefficient implicit and explicit dissipation

The most common procedure is to add explicit fourth-order artificial dissipation to the central difference algorithm of the form

\[ -\epsilon_c \Delta t J^{-1} [ (\nabla_\xi \Delta_\xi)^2 + (\nabla_\eta \Delta_\eta)^2 ] J Q^{n+1} \]  

(4.1)

which is added to the right-hand side of (3.22) and implicit second-order smoothing

\[ -\epsilon_i \Delta t J^{-1} \nabla_\xi \Delta_\xi J, \quad -\epsilon_i \Delta t J^{-1} \nabla_\eta \Delta_\eta J \]  

(4.2)

which is inserted into the respective implicit block operators. Second-order implicit dissipation is used to keep the block implicit operators tridiagonal. The difference operators are defined as

\[ \nabla_\xi q_{j,k} = q_{j,k} - q_{j-1,k}, \quad \Delta_\xi q_{j,k} = q_{j+1,k} - q_{j,k}, \]

\[ \nabla_\eta q_{j,k} = q_{j,k} - q_{j,k-1}, \quad \Delta_\eta q_{j,k} = q_{j,k+1} - q_{j,k} \]  

(4.3)

and are applied at all interior points. The parameter \( \epsilon_c \) is chosen to be \( O(1) \) and \( \epsilon_i = 2\epsilon_c \). The smoothing terms are scaled with \( \Delta t \) which makes the steady state independent of the time step.

It is important to assess the effect on stability when these terms are added. In Section 4, we provide a linear analysis of the effect of added dissipation on stability. We summarize the results here. In the original development of the implicit algorithm we only added in the explicit dissipation, but this led to a linear stability bound which was dependent on the magnitude of \( \epsilon_c \Delta t \). The implicit second-order term was added to eliminate this stability bound. The proper approach would be to make the fourth-order dissipation implicit. This would then necessitate the use of block pentadiagonal solvers which is too computationally expensive. The second-order implicit dissipation stabilizes the algorithm and allows us to retain block tridiagonal inversions. Linear analysis shows that if \( \epsilon_i \gg \epsilon_c \) then unconditional stability is obtained. It should be noted that in practice for nonlinear problems the total algorithm has large but conditional stability bounds.
Recent work by Beam and Bailey [20] suggest that while the implicit second-order dissipation improves the practical stability bound, the use of fourth-order implicit dissipation matching the explicit terms produces larger stability bounds and enhanced convergence. This is consistent with a concept which I will discuss in more detail below. That is, maximum stability bounds and optimal convergence rates are only achieved if we properly linearize the explicit side of the algorithm. In this case a proper linearization of the explicit dissipation produces improved stability and convergence. Beam and Bailey employed a block pentadiagonal solver which greatly increased the computational work and storage. We take advantage of the diagonal algorithm to produce a much more efficient scheme. Within the framework of the diagonal scheme we can replace the four scalar tridiagonals with scalar pentadiagonals which is just a slight increase in computational work. The resulting scheme has the advantage of increased stability bounds and convergence rates with the total computation work still less than the standard block tridiagonal scheme. Computational experiments demonstrate the increased efficiency and stability.

The approach of adding a constant coefficient fourth-order explicit dissipation can produce some problems which are only evident in the case of refined meshes. Initially, because of computer limitations we only employed coarse grids and this type of dissipation was sufficient to produce stability and limited accuracy. With the advent of more powerful computers we have gone to grid refinement especially to resolve shocks. The use of the above type of fourth-order dissipation with refined meshes produces wild oscillations near shocks even in cases where the computation is completely stable and converged. In Fig. 2, we show a converged solution for a NACA 0012 airfoil at a transonic Mach number, \( M_a = 0.8 \), and angle of attack, \( \alpha = 0^\circ \), isolating the region near the shock.

As can be seen, the solution seems perfectly fine except in the region of the shock where a large every other point oscillation is evident. Varying the coefficient of artificial dissipation over a fairly wide range did not alter the nature of this oscillation. This is obviously an undesirable result which can be eliminated as shown below.

![Graph showing coefficient of pressure oscillations at shock.](image-url)
4.2. The upwind connection to artificial dissipation

In the last few years a number of schemes have been developed based on upwind differencing. The flux split schemes of Steger and Warming [21], Roe [22], and van Leer [23] employ a decomposition of the flux vectors in such a way that each element can be stably differenced in an upwind fashion. Other schemes of a similar nature but based on complicated theories are the flux difference scheme of Osher and Chakravarthy [24] and Harten’s TVD methods [25]. These schemes all claim (with good justification) to be physically consistent since they follow in some sense the characteristics of the flow. They in general can be shown to produce sharp oscillation free shocks without added artificial dissipation. They are, though, complicated schemes which are just now being applied to complicated flowfield situations. Also these schemes have an inherent amount of internal dissipation, due to the one-sided differences, which cannot be modified or decreased. It may be advantageous to have the flexibility of a simple central difference scheme with a controllable amount of artificial dissipation.

It can be shown (as done below) that the upwind schemes have an equivalence to central difference schemes with added dissipation. The central schemes are much simpler and more flexible and are therefore desirable if the dissipation can be added in an analogous fashion to the upwind schemes.

The plus-minus flux split method of Steger and Warming [21] will be used here to demonstrate the dissipative nature of upwind schemes. The approach taken is to split the eigenvalue matrix $\Lambda$ of the flux Jacobians into two matrices, one with all positive elements and the other with all negative elements. Then the similarity transformations $X$ or $Y$ are used to form new matrices $A^+$, $A^-$ and $B^+$, $B^-$. Formally,

$$A = X\Lambda X^{-1} = X(\Lambda_+ + \Lambda_-)X^{-1} = A^+ + A^- \tag{4.4a}$$

with

$$\Lambda_\pm = \frac{1}{2}(\Lambda_\pm \pm |\Lambda_\pm|) \tag{4.4b}$$

Here, $|\Lambda|$ implies that we take the absolute values of the elements of $\Lambda$. The two matrices, $A^+$ and $A^-$, have by construction all positive and all negative eigenvalues, respectively.

New flux vectors can be constructed as

$$E = AQ = (A^+ + A^-)Q = E^+ + E^-,$$
$$F = BQ = (B^+ + B^-)Q = F^+ + F^- \tag{4.5}$$

Different types of spatial differencing can now be used for each of the new flux vectors. One stable form is to use one-sided backward differencing for the positive terms and one-sided forward differencing for the negative terms. The one-sided difference operators are usually either first-order accurate

$$\nabla_\xi u_{j,k} = \frac{u_{j,k} - u_{j-1,k}}{\Delta \xi} \quad \text{and} \quad \Delta_\xi u_{j,k} = \frac{u_{j+1,k} - u_{j,k}}{\Delta \xi} \tag{4.6a}$$

or second-order accurate

$$\delta_\xi u_{j,k} = \frac{3}{2}u_{j,k} - 2u_{j-1,k} + \frac{1}{2}u_{j-2,k} \frac{\Delta \xi}{\Delta \xi}, \quad \delta_\xi u_{j,k} = \frac{-3}{2}u_{j,k} + 2u_{j+1,k} - \frac{1}{2}u_{j+2,k} \quad \frac{\Delta \xi}{\Delta \xi}. \tag{4.6b}$$
Similar expressions are used for the $\eta$ derivatives. Note that even though we generally take $\Delta \xi = 1$, it will appear in formulas where its presence conveys meaning. The plus-minus matrices, $A^+$ and $A^-$, can be written as

$$A^\pm = X[\frac{1}{2} (A \pm |A|)]X^{-1} = \frac{1}{2} (A \pm |A|),$$  \hspace{1cm} (4.7a)

which gives

$$E^\pm = A^\pm Q = \frac{1}{2} A Q \pm \frac{1}{2} |A| Q = \frac{1}{2} E \pm \frac{1}{2} |A| Q.$$  \hspace{1cm} (4.7b)

Similar expressions are obtainable for the $B$ matrices and flux vector $F$.

Examining the flux derivative

$$\delta^b E^+ + \delta^f E^-$$

where second-order one-sided difference approximations are chosen

$$\delta^b = \frac{3I - 4 \varepsilon^{-1} + \varepsilon^{-2}}{2 \Delta \xi}, \quad \delta^f = \frac{-3I + 4 \varepsilon^{+1} - \varepsilon^{+2}}{2 \Delta \xi}$$  \hspace{1cm} (4.8a)

with $\varepsilon^i$ the shift operator, i.e., $\varepsilon^i u_j = u_{j+i}$.

Combining (4.7b) and (4.8) we have

$$\frac{1}{2} [(\delta^b + \delta^f) E + (\delta^b - \delta^f) |A| Q]$$

for the difference equation.

It is easily shown that

$$\frac{1}{2} (\delta^b + \delta^f) = (-\varepsilon^{-2} + 4 \varepsilon^{+1} - 4 \varepsilon^{-1} + \varepsilon^{-2})/(4 \Delta \xi) = \bar{\delta}$$  \hspace{1cm} (4.10a)

which is a second-order central difference operator, but not $\delta^-$. The other term of (4.9) is of more interest, where

$$\frac{1}{2} (\delta^b - \delta^f) = (\varepsilon^{+2} - 4 \varepsilon^{+1} + 6I - 4 \varepsilon^{-1} + \varepsilon^{-2})/(4 \Delta \xi) = \frac{1}{4 \Delta \xi} (\Delta \xi \nabla \xi)^2$$  \hspace{1cm} (4.10b)

which is a fourth-order difference stencil. The difference operators are defined in (4.3).

Now (4.9) can be written as

$$\left( \delta^b E + \frac{1}{4 \Delta \xi} (\Delta \xi \nabla \xi)^2 |A| Q \right).$$  \hspace{1cm} (4.11a)

The form now is a second-order central difference term plus fourth-order dissipation. The dissipative term is a natural consequence of the upwind differencing. It is interesting to note that the central difference term (4.10a) is not the standard three-point difference. If first-order formulas are employed for the upwind differences then a similar analysis would produce the standard second-order three-point central differencing plus a second-order dissipative term. For instance, (4.11a) is replace by

$$\left( \delta^b E - \frac{1}{2 \Delta \xi} (\Delta \xi \nabla \xi) |A| Q \right).$$  \hspace{1cm} (4.11b)

We note a number of things from the form of (4.11) which can guide us in developing artificial dissipation models for a central difference scheme. Adding fourth-order dissipation to a central difference produces the equivalent of some second-order upwind scheme. The use of
second-order dissipation can produce a first-order upwind equivalent. Recent work applying flux limiters to upwind schemes and some of the TVD concepts suggest that the best approach for an upwind algorithm is to use a locally first-order upwind difference at a shock and second-order elsewhere. This can be accomplished by some switching and transitioning of second-order and fourth-order dissipation added to a central scheme. The coefficients for the dissipation parts of (4.11) suggest some sort of flux Jacobian scaling where for instance a spectral radius of the Jacobians could be used.

4.3. Nonlinear artificial dissipation model

As seen from the previous analysis a mixed second- and fourth-order dissipation model with appropriate coefficients should produce a scheme with good shock capturing capabilities. Jameson et al. [6] has employed a dissipative model of such a form where second- and fourth-order dissipation are combined. The model rewritten in our notation is

\[ \nabla_{j}^{s} \left( \sigma_{j+1,k} J_{j+1,k}^{-1} + \sigma_{j,k} J_{j,k}^{-1} \right) \left( \epsilon_{j,k}^{(2)} \Delta_{j,k} Q_{j,k} - \epsilon_{j,k}^{(4)} \nabla_{j}^{s} \Delta_{j,k} Q_{j,k} \right) \]  

with

\[ \epsilon_{j,k}^{(2)} = \kappa_{2} \Delta t \max(Y_{j+1,k}, Y_{j,k}, Y_{j-1,k}) \]  

\[ \epsilon_{j,k}^{(4)} = \max(0, \kappa_{4} \Delta t - \epsilon_{j,k}^{(2)}) \]  

where typical values of the constants are \( \kappa_{2} = \frac{1}{4} \) and \( \kappa_{4} = \frac{1}{100} \). Similar terms are used in the \( \eta \) direction. The term \( \sigma_{j,k} \) is a spectral radius scaling and is defined as

\[ \sigma_{j,k} = |U| + a \sqrt{\xi_{x}^{2} + \xi_{y}^{2}} \]  

which is the spectral radius of \( \hat{A} \), the spectral radius of \( \hat{B} \) is used for the \( \eta \) dissipation.

The first term of (4.12) is a second-order dissipation with an extra pressure gradient coefficient to increase its value near shocks. The second term is a fourth-order term where the logic to compute \( \epsilon_{j,k}^{(4)} \) switches it off when the second-order nonlinear coefficient is larger than the constant fourth-order coefficient. This occurs right near a shock. In Figs. 3 and 4, we show solutions for the flow problem of Fig. 2, using this nonlinear artificial dissipation. For Fig. 3 we employ just the fourth-order term, i.e. \( \kappa_{2} = 0 \).

The oscillations at the shock are eliminated and a sharp shock is obtained. In this case though there is an overshoot and undershoot at the top and bottom of the shock which is eliminated in Fig. 4 by adding the second-order term, \( \kappa_{2} = \frac{1}{4} \).

The results shown are fully converged to machine zero and in the case of Fig. 4 represent the current quality of our shock capturing capabilities. The chosen values of the coefficients have, at least to date, been static and are not changed from case to case.

The implicit dissipation used with (4.12) is the linearization of the equation treating the pressure coefficient \( Y \) and the spectral radius \( \sigma \) as space varying functions but ignoring their functional dependency on \( Q \). Then the dissipation is linear in \( Q_{j,k} \) and is added to the diagonal algorithm again necessitating scalar pentadiagonal solvers. This produces a very efficient, stable and convergent form of the implicit algorithm.

Near computational boundaries we modify the fourth-order dissipation so as to maintain a dissipative term. A derivation and analysis of various boundary treatments is given in [26]. The modification is needed at the first interior point (e.g. \( Q_{j+1,k} \)) where the five-point
fourth-order term \( Q_{j+2,k} - 4Q_{j+1,k} + 6Q_{j,k} - 4Q_{j-1,k} + Q_{j-2,k} \) is to be applied. There the point \( Q_{j+2,k} \) doesn’t exist, the formula is modified to a one-sided second-order term with the differencing stencil \(-2Q_{j+1,k} + 5Q_{j,k} - 4Q_{j-1,k} + Q_{j-2,k}\). Similar formulas are used at other boundaries.

4.4. Total variation diminishing schemes, TVD

The development of monotone, flux vector–difference splitting, TVD and other nonoscillatory schemes can be found in numerous publications, see for example, [7–9, 21–25]. Here we shall just briefly define the conditions for a class of TVD schemes introduced by Harten [25].

The conditions for a scheme to be TVD in Harten’s sense can be developed for the scalar hyperbolic conservation law

\[
\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0
\]

(4.15)

where \( f \) (the flux) is a nonlinear function of \( u \). We can define a characteristic speed \( a(u) = \frac{\partial f}{\partial u} \).

A one parameter family of schemes can be defined.

\[
\frac{u_{j+1}^{n+1} + \lambda a(h_{j+1/2}^{n+1} - h_{j-1/2}^{n+1}) - u_{j}^{n} - \lambda (1 - \theta)(h_{j+1/2}^{n+1} - h_{j-1/2}^{n+1})}{\Delta t} = 0 ,
\]

(4.16a)

rewritten as

\[
Lu_{j}^{n+1} = Ru_{j}^{n}
\]

(4.16b)

where \( u_{j}^{n} = u(j \Delta x, n \Delta t), \lambda = \Delta t/\Delta x, \theta \) parameterizes the equations from the fully explicit to fully implicit forms, and \( h \) is the numerical flux function with \( h_{j+1/2} = h(u_{j+1}, u_{j}, u_{j+1}, u_{j+2}) \).

The total variation of a mesh function \( u^n \) is defined as

\[
TV(u^n) = \sum_{j=-\infty}^{\infty} |u_{j+1}^n - u_{j}^n| = \sum_{j=-\infty}^{\infty} |\Delta_{j+1/2}u^n|
\]

(4.17)

where \( \Delta_{j+1/2} = u_{j+1} - u_{j} \).
A numerical scheme is TVD if
\[ \text{TV}(u^{n+1}) \leq \text{TV}(u^n). \] (4.18)

For (4.16) the conditions due to Harten [25] are
\[ \text{TV}(R u^n) \leq \text{TV}(u^n) \] (4.19a)
and
\[ \text{TV}(L u^n) \geq \text{TV}(u^{n+1}). \] (4.19b)

Rewriting (4.16), assuming \( h \) is Lipschitz continuous,
\[ u_j^{n+1} - \lambda \theta (C^-_{j+1/2} \Delta_{j+1/2} u - C^+_{j-1/2} \Delta_{j-1/2} u)^{n+1} \]
\[ = u_j^n + \lambda (1 - \theta) (C^-_{j+1/2} \Delta_{j+1/2} u - C^+_{j-1/2} \Delta_{j-1/2} u)^n \] (4.20)
with \( C^\pm \) bounded functions. Sufficient conditions for (4.19) are for all \( j \):
\[ \lambda (1 - \theta) C^-_{j+1/2} \geq 0, \quad \lambda (1 - \theta) (C^+_{j+1/2} + C^-_{j+1/2}) \leq 1, \]
\[ -\infty < C \leq -\lambda \theta C^z_{j+1/2} \leq 0 \] (4.21)
for finite \( C \).

These conditions can be used to analyze and construct various TVD schemes. Refer to [7, 8, 25] for two forms of high resolution (at least second-order accurate) TVD schemes applied to hyperbolic conservation law equations.

5. Time accuracy, steady states, convergence and stability

5.1. Time accuracy versus steady-state computation

The implicit algorithm is designed to be time-accurate where second-order accuracy can be maintained and the equations are integrated through time from some meaningful initial condition to the solution at time \( T \). In this case the time step is chosen to be commensurate with some time scale of the problem. The evolution of the solution through time is physically realistic and good solution accuracy is dependent on the mesh spacing and boundary conditions.

The equations can also be applied to steady-state problems. Typically we employ the first-order scheme in time and attempt to accelerate the algorithm by various non-time-like maneuvers. The equations are then integrated from an arbitrary initial condition to a time asymptotic state. Any procedure which drives us to the steady state faster must also be stable and accurate at convergence. It might be expected that large time steps could be used to drive the solution to the steady state faster. As we shall see, based on linear analysis large time steps can increase the convergence rate, but for factored forms the limit of the amplification factor (a measure of the maximum convergence rate) as \( h = \Delta t \to \infty \) is 1.

5.1.1. Effect of factorization errors on convergence

Let us divide the total solution into the transient (time-like) and particular (steady-state) parts. The goal of any fast steady-state algorithm is to eliminate the transient as quickly as
possible. We can examine the ability of the implicit scheme to eliminate transients by investigating the model problem, (3.12). In this case, instead of (3.14) we take
\[ w = \tilde{w}(t) e^{i\beta x} \]  
and treat the spatial derivative \( \partial_x \) analytically, then examine the temporal differencing schemes in one and two dimensions. This gives us the purely transient one-dimensional model problem
\[ \tilde{w}_t + \lambda \tilde{w} = 0 \]  
with \( \lambda = i\beta \).

The delta form of the first-order implicit algorithm is
\[ (1 + h\lambda) \Delta \tilde{w}^n = -h\lambda \tilde{w}^n \]  
which can be rewritten as
\[ \tilde{w}^{n+1} = \left[ \frac{1}{1 + h\lambda} \right] \tilde{w}^n \quad \text{or} \quad \tilde{w}^n = \left[ \frac{1}{1 + h\lambda} \right]^n \tilde{w}^0 \]  
where \( \tilde{w}^0 \) is some initial value. The term in the brackets is the amplification factor, \( \sigma \).

For \( h \to \infty \), \( \tilde{w}^n \to 0 \) and the transient can actually be eliminated directly for large \( h \).

In contrast, let us examine a two-dimensional factored implicit scheme for the two-dimensional transient problem
\[ \tilde{w}_t + \lambda_x \tilde{w} + \lambda_y \tilde{w} = 0 . \]  
This is the two-dimensional counterpart to (5.2). Applying the first-order implicit approximate factorization delta algorithm to (5.5) we have
\[ [1 + h\lambda_x][1 + h\lambda_y] \Delta \tilde{w}^n = -h(\lambda_x + \lambda_y) \tilde{w}^n . \]  
Expanding \( \Delta \tilde{w}^n = \tilde{w}^{n+1} - \tilde{w}^n \) and combining terms we have
\[ \tilde{w}^n = \left[ \frac{(1 + h^2\lambda_x \lambda_y)}{(1 + h\lambda_x + h\lambda_y + h^2\lambda_x \lambda_y)} \right]^n \tilde{w}^0 \]  
and so \( |\sigma| \to 1 \) as \( h \to \infty \).

A close examination of this result shows that the factorization has destroyed the good convergence characteristics at large time steps. The factoring error term has introduced a \( h^2 \) term in the numerator of the amplification factor. Therefore the factored schemes do not have good convergence characteristics for large time steps. Actually, there is a range of moderately large time steps where the amplification factor is a minimum, see for instance [27]. Convergence can therefore be accelerated by using a time step which minimizes the amplification factor.

Note that for the delta form of the algorithm (either factored or unfactored) the steady-state solution is independent of the time step, \( h \). (There are numerical schemes where this is not the case, such as Lax–Wendroff.) Therefore, the time step path to the steady state does not affect the final solution and we can envision using time step sequences or spatially variable time steps to accelerate convergence.
5.1.2. Space varying $\Delta t$

Manipulation of the time step can have a substantial influence on convergence even within
the framework of the factored algorithms. If only a steady-state solution is required, one can
let $h$ (or $\Delta t$) change in space. This approach can be viewed as a way to condition the iteration
matrix of the relaxation scheme defined via (3.22) or (3.26). Use of a space varying $\Delta t$ can
also be interpreted as an attempt to use a more uniform Courant number throughout the field.
In any event, changing $\Delta t$ can be effective for grid spacing that vary from very fine to very
coarse—a situation usually encountered in aerodynamic simulations where grids contain a
wide variety of length scales.

A space varying $\Delta t$ has been used in both explicit and implicit schemes (e.g. by Shang and
Hankey [28], McDonald and Briley [29], Srinivasan et al. [30], Coakley [31], Jameson et al.
[6]). As a rule one wishes to adjust $\Delta t$ at each point proportional to the grid spacing and the
characteristic speed of the flow. Something like the Courant number restriction (which for the
Euler equations in multi-dimensions is a bit of an approximation).

For highly stretched grids the space variation of the grid is the most important parameter to
scale with. In subsonic and transonic flow the characteristic speeds have moderate variation
and we have found that a purely geometric variation of $\Delta t$ is adequate, specifically

$$\Delta t = \frac{\Delta t|_{ref}}{1 + \sqrt{J}}. \quad (5.8a)$$

To illustrate the advantage of using a variable time step, Fig. 5 shows the degradation in
convergence rate when a constant step size is substituted for the variable time step in a NACA
0012 test case. For this comparison all other possible parameters were held constant and no
other changes were employed. Also other forms of the variable step size sometimes perform
better than (5.8a), for example

$$\Delta t = \frac{\Delta t|_{ref}}{|U| + |V| + a\sqrt{\xi_x^2 + \xi_y^2 + \eta_x^2 + \eta_y^2}} \quad (5.8b)$$

which is approximately a constant CFL condition. However, (5.8b) is more costly to compute
than (5.8a).

5.1.3. Mesh sequences

For inviscid airfoil calculations on a grid of $O(250 \times 50)$ practical convergence is usually
obtained in 500–600 fine grid iterations when the flow field has been started from an initial
condition of uniform free stream flow. Typically the first 100 to 200 iterations on the fine mesh
are needed to get past the initial transients which can be a substantial portion of the total
solution time. For instance, in the above test case it takes on the order of 600 fine grid
iterations for a tight convergence criteria (e.g. lift to 5 decimal places), 200 of which are spent
on clearing out the impulsive start. One way to accelerate convergence to a steady state is to
obtain a good initial guess for a fine mesh by first iterating on a sequence of coarse grids and
then interpolating the solution up to the next refined grid. Such a mesh sequence procedure
can often reduce the amount of time required to obtain a solution to plottable accuracy by a
factor of two. Also, because a coarse grid tends to damp high-frequency waves, using a mesh
sequence procedure can improve the overall robustness of the code.

A mesh sequencing procedure can be implemented in an optionally called stand alone
routine. If a sequence of $m$ grids are used, a coarsened grid is cut from each previous grid by
halving the number of points in the $\xi$ direction and by regenerating a new $\eta$ distribution of points in the $\eta$ direction using a fewer number of points. A redistribution in the $\eta$ direction is used instead of halving so that clusterings near surfaces can be maintained. A finite number of iterations (perhaps 50) are carried out on each coarsened grid at which point the approximate solution is interpolated onto a more refined grid. The finest grid is then iterated to convergence. The result is faster convergence to practical levels and a more robust starting procedure.

For a NACA 0012 test case a sequence of 3 grids has been used; 48 by 18 and 96 by 25 and the final grid of 192 by 33 points. The convergence of $C_l$ is shown in Fig. 6 to indicate the overall improvement in convergence due to the use of mesh sequencing in comparison to the use of a fine grid only. Both cases were started with a free stream initial condition.

### 5.2. Effect of dissipation model on convergence and stability

As discussed in Section 4, based on linear theory the use of explicit dissipation produces explicit stability bounds unless implicit dissipation is added. The second-difference dissipation, (4.2), will stabilize the fourth-difference dissipation if the coefficients are chosen properly. Ideally though, it would be better to treat the explicit dissipation in a fully implicit manner. That is, use implicit fourth-difference dissipation which is an exact linearization of the explicit fourth-difference dissipation. In fact, although the implicit second-difference dissipa-
tion stabilizes the fourth-difference dissipation it can have a detrimental effect on the convergence rates of an implicit algorithm for steady-state computations.

Consider the model problem in one-dimension (equivalent to (3.12) with a convenient change in notation),

\[ q_t + aq_x = 0 \]  \hspace{1cm} (5.9)

Applying the first-order time-accurate Euler implicit scheme in delta form to (5.9) and adding explicit fourth-difference dissipation \((\beta_4 > 0)\), implicit second-difference dissipation \((\alpha_2 > 0)\), and implicit fourth-difference dissipation \((\alpha_4 > 0)\) gives the algorithm

\[ [1 + h a \delta_x - h a_2 \nabla_x \Delta_x + h a_4 (\nabla_x \Delta_x)^2] (q^{n+1} - q^n) = -h (a \delta_x + \beta_4 (\nabla_x \Delta_x)^2) q^n . \]  \hspace{1cm} (5.10)

Fourier analysis using \( q^n = w^n e^{ik_j Ax} \) (with \( k_j \) the wave number in \( x \)) produces

\[ [1 + h a \lambda_x - h a_2 \mu_x + h a_4 \mu_x^2] (w^{n+1} - w^n) = -h (a \lambda_x + \beta_4 \mu_x^2) w^n \]  \hspace{1cm} (5.11)

where \( \lambda_x = 2i \sin(k_j \Delta x)/\Delta x \) represents the Fourier signature for the central difference \( \delta_x \), \( \mu_x = -2 + 2 \cos(k_j \Delta x) \) the signature of the second-difference dissipation operator \( \nabla_x \Delta_x \), and \( \mu_x^2 \) for the fourth-difference dissipation.

The amplification factor for \( w^{n+1} = \sigma w^n \) is then

\[ \sigma = \frac{1 + h (\alpha_4 - \beta_4) \mu_x^2 - \alpha_2 \mu_x}{1 + h (a \lambda_x - \alpha_2 \mu_x + \alpha_4 \mu_x^2)} . \]  \hspace{1cm} (5.12)

The choices which will be investigated are

1. \( \beta_4 \neq 0 \) and \( \beta_4 = \alpha_2 = 0 \), explicit dissipation only.
2. \( \beta_4 \neq 0 \), \( \alpha_2 \neq 0 \), and \( \alpha_2 = 0 \), explicit fourth-difference dissipation and implicit second-difference dissipation, no implicit fourth-difference dissipation.
3. \( \beta_4 \neq 0 \), \( \alpha_2 \neq 0 \) and \( \alpha_2 = 0 \), explicit and implicit fourth-difference dissipation with no implicit second-difference dissipation.

For case (1), explicit dissipation only, (5.12) becomes

\[ \sigma = (1 - h \beta_4 \mu_x^2)/(1 + h a \lambda_x) . \]  \hspace{1cm} (5.13a)

Now, since \( \lambda_x \) is pure imaginary and has a minimum of 0, and \(-4 \leq \mu_x \leq 0\) the explicit stability bound is \( h \beta_4 < \frac{1}{8} \). This is a limit on the product of \( h \) and \( \beta_4 \) and therefore one can always find a combination which will be stable. But, for arbitrary \( h \), especially in the case where large \( h \) are used to accelerate convergence, this bound is too restrictive.

In the second case, implicit second-difference dissipation can eliminate the above stability bound. The amplification factor \( \sigma \) is now

\[ \sigma = (1 - h \beta_4 \mu_x^2 - h a_2 \mu_x)/(1 + h a \lambda_x - h a_2 \mu_x) . \]  \hspace{1cm} (5.13b)

The numerator term \( \lambda_x \) can only improve the stability bounds since it is pure imaginary, so it is taken at its minimum, 0. Let \( \alpha_2 = 2 \beta_4 \) and apply the stability condition \(| \sigma | \leq 1 \) which results in the condition \( -2 \leq -h \beta_4 \mu_x (4 + \mu_x) \). Since \( \mu_x \leq 0 \), the condition can be rewritten as \(-2 \leq h \beta_4 \mu_x (4 + \mu_x) \) which is satisfied because \(-4 \leq \mu_x \). Therefore, using \( \alpha_2 = 2 \beta_4 \) leads to unconditional stability. The disadvantage of this form is evident from the amplification factor, (5.13b). Even though the scheme is unconditionally stable, \( \sigma \to 1 \) as \( h \to \infty \). In fact, the amplification factor has a minimum at a finite \( h \) and then asymptotes rapidly to 1 as \( h \to \infty \).
increases. For this reason, large $h$ cannot be used to accelerate convergence even in this simple one-dimensional example.

In contrast, the third case of implicit and explicit dissipation is unconditionally stable and has good convergence characteristics for large $h$. The amplification factor $\sigma$ for $\alpha_4 = \beta_4$ and $\alpha_2 = 0$ is

$$\sigma = \left[1 + h(\alpha_4 + \alpha_4 \mu_x^2)\right]^{-1}$$

which is unconditionally stable and $\sigma \rightarrow 0$ as $h \rightarrow \infty$.

The analysis for two and three dimensions is straightforward and gives similar results for the unfactored forms. The optimal algorithm is a fully implicit one. In general, optimal stability and convergence only occurs for the fully implicit form of the algorithm.

We demonstrate the improved convergence and stability bounds in Fig. 7.

The curves in Fig. 7 are convergence histories for a transonic airfoil computation showing the effect of a fully implicit treatment of the artificial dissipation. The upper curve is the result of second-order constant coefficient implicit dissipation, (4.2), with nonlinear explicit dissipation, (4.12). A much faster convergence rate is obtained in this problem when the second-order implicit dissipation is replaced by an implicit linearization of the nonlinear dissipation of (4.12) (see [26] for more details). Also the maximum allowable time step is at least 10 times larger for the fully implicit scheme.

---

![Fig. 7. Improvement in convergence rate due to implicit treatment of artificial dissipation.](image-url)
6. Application to fluid dynamics code—ARC2D

General purpose centrally space differenced implicit finite difference codes in two [2] and three [3] dimensions have been developed at NASA Ames and have been widely distributed since their introduction in 1977 and 1978. These codes, now referred to as ARC2D and ARC3D, can run either in inviscid or viscous mode for steady or unsteady flow. They use general coordinate systems and can be run on any smoothly varying curvilinear mesh, even a mesh that is quite skew. Because they use well-ordered finite difference grids, the codes can

![Fig. 8. NACA 0012 grid using 192 by 33 grid points.](image)
take advantage of vectorized computer processors and have been implemented on Control Data 205 and the CRAY 1-S and X-MP. On a single processor of the X-MP a vectorized version of the code runs approximately 20 times faster than the original code which was written for the Control Data 7600.

Traditionally gains in computational efficiency due to improved numerical algorithms have kept pace with gains due to increased computer power. Since the ARC2D and ARC3D codes were introduced, a variety of algorithmic changes have been individually tested and have been shown to improve overall computational efficiency. These include use of the spatially varying time step (Δt), use of a sequence of mesh refinements to establish approximate solutions, implementation of various ways to reduce inversion work, improved numerical dissipation terms, and more implicit treatment of terms. Although the various individual algorithm improvements can interact with each other, sometimes adversely making optimization difficult, their combined effect has led to an order of magnitude gain in computational efficiency for steady-state applications. This is a gain equivalent to that achieved with computer hardware. Unsteady flow calculations have also benefited from some of the above improvements.

Fig. 9. ARC2D results for 192 by 33 grid.
Details of the development and application in two and three dimensions can be found in [32, 33]. For this paper, we restrict ourselves to ARC2D in its inviscid mode. The examples given below demonstrate the accuracy, efficiency and flexibility of algorithms and codes which employ the theories presented above.

6.1. Inviscid airfoils

The code ARC2D has the option of computing the inviscid equations (the Euler equations). The basic version of the code is written specifically for airfoil computations. The code has been applied to a wide variety of airfoil shapes, flow conditions, and other geometries. We have validated the code against other computational methods, [2, 34]. A wide variety of applications can be found in [35–41]. To demonstrate the accuracy and efficiency we have chosen two test cases, a NACA 0012 airfoil at $M_a = 0.8$, $\alpha = 1.25^\circ$ on a coarse grid (192 by 33 points) and a fine grid (248 by 49 points). For comparison purposes we use results from Jameson's multigrid Euler code FLO52R [42]. FLO52R is an Euler code using a multistage Runge–Kutta like algorithm with a multigrid scheme to accelerate convergence. The code

![Fig. 10. FLO52R results for 192 by 33 grid.](image-url)
employs enthalpy damping, residual averaging and an artificial dissipation model of the same form as presented in Section 4.3. In fact the boundary conditions and artificial dissipation model used in ARC2D were modified to be the same as in FLO52R so that quantitative as well as qualitative comparisons could be made. The two codes were run on the same machine, the CRAY X-MP at NASA Ames, on the same meshes and at the same flow conditions.

The first case is the NACA 0012 airfoil at $M_{\infty} = 0.8$ and $\alpha = 1.25^\circ$. The grid used is an ‘O’ mesh topology with 192 points on the airfoil surface (running from the lower trailing edge around the nose to the upper trailing edge) and 33 point in the normal direction. The grid which is clustered at the leading and trailing edges, near the expected shock locations on the upper and lower surfaces and in the normal direction is shown in Fig. 8.

Results from this case using ARC2D are shown in Fig. 9. We show here the coefficient of pressure, Mach contours, pressure contours and contours of entropy. In Fig. 10 we show similar results for FLO52R. Computed lift for ARC2D is $C_L = 0.33957$ and for FLO52R $C_L = 0.32408$. The comparison between the two codes is quite good.
We have established a number of accuracy checks and convergence criteria for comparison purposes. In terms of accuracy we recommend comparison of pressure coefficients, lift and other flow quantities. It is also important to establish the accuracy of certain flow regions. The stagnation region near the nose of the airfoil is particularly susceptible to errors due to poor boundary conditions, resolution, or physical assumptions. The best measure of this error is the entropy field. For inviscid flow there should be no generation of entropy at the leading edge of an airfoil in the absence of a leading edge shock. Examination of the entropy at the leading edge for the above case shows, see Fig. 11, that both codes give rise to some error at the leading edge, although the magnitude is rather small.

A number of convergence criteria have been chosen to assess the efficiency and convergence rates of the codes. We have chosen to use computer times as our measure of relative speed. Since the two codes are run on the same machines and with the same meshes this is an adequate measure. Other measures such as operation count, work or iteration are usually programming dependent or susceptible to misinterpretation. The convergence criteria used here are:

![Coeficient of Pressure and Mach Contours](image1)

![Pressure Contours and Entropy Contours](image2)

Fig. 13. ARC2D results for 248 by 49 grid.
(1) Coefficient of lift \( (C_L) \) to 1% of converged value.
(2) Coefficient of lift \( (C_L) \) to \( \frac{1}{2} \% \) of converged value.
(3) Coefficient of lift \( (C_L) \) to 5 decimal places.
(4) Number of supersonic points to converged value.
(5) Residual to machine zero. \( (10^{-13} \) on the Cray X-MP.)

The residual is the \( l_2 \) norm of the explicit or right-hand side of (6.1). We use just the component from the continuity equation, the other components behave similarly. For the above case on the 192 by 33 mesh the computer times for the convergence criteria are given in Table 1.

As can be seen for this case FLO52R is up to twice as fast as ARC2D for some criteria. In either event these are fairly good convergence times. In general, these numbers carry over fairly consistently for a wide variety of airfoils and flow conditions for similar meshes.

A more stringent test is obtained with a finer grid and more grid points. A grid of 248 by 49 points is employed as the second study. The mesh is refined more at the nose, tail and near the shocks. Also to reduce the entropy errors at the nose the grid is clustered more tightly in the normal direction by reducing the minimum normal spacing by a factor of 2. The mesh is shown in Fig. 12.

Fig. 14. FL052R results for 248 by 49 grid.
Table 1
Convergence data for 192 by 33 grid. Convergence comparison (seconds)

<table>
<thead>
<tr>
<th>Criteria</th>
<th>ARC2D</th>
<th>FLO2R</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1% \text{ of } C_L)</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>(\frac{1}{2}% \text{ of } C_L)</td>
<td>17</td>
<td>10.5</td>
</tr>
<tr>
<td>(C_L) to 5 places</td>
<td>57</td>
<td>31</td>
</tr>
<tr>
<td>No. S.S. pts</td>
<td>36</td>
<td>17</td>
</tr>
<tr>
<td>Machine zero</td>
<td>120</td>
<td>97</td>
</tr>
</tbody>
</table>

Table 2
Convergence data for 248 by 49 grid. Convergence comparison (seconds)

<table>
<thead>
<tr>
<th>Criteria</th>
<th>ARC2D</th>
<th>FLO52R</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1% \text{ of } C_L)</td>
<td>38</td>
<td>23</td>
</tr>
<tr>
<td>(\frac{1}{2}% \text{ of } C_L)</td>
<td>52.5</td>
<td>25.5</td>
</tr>
<tr>
<td>(C_L) to 5 places</td>
<td>174</td>
<td>168.5</td>
</tr>
<tr>
<td>No. S.S. pts</td>
<td>118</td>
<td>160</td>
</tr>
<tr>
<td>Machine zero</td>
<td>376</td>
<td>800+</td>
</tr>
</tbody>
</table>

Fig. 15. Convergence history versus iteration for ARC2D results.

Computational results for ARC2D and FLO52R are shown in Figs. 13 and 14. In this case the shocks are sharper and entropy errors at the leading edge are eliminated.

Convergence data for this case are contained in Table 2. In Fig. 15, we show convergence history versus iteration for the two ARC2D results. All the results obtained with ARC2D were done using the fully implicit pentadiagonal algorithm. As mentioned above, numerous other cases and airfoils have been computed and perform similarly.

Appendix

The flux Jacobian matrices of (3.4) have real eigenvalues and a complete set of eigenvectors. The similarity transforms are

\[
\hat{A} = T_\xi A_\xi T_\xi^{-1} \quad \text{and} \quad \hat{B} = T_\eta A_\eta T_\eta^{-1}
\]  
(A.1)
where

$$\Lambda_\xi = \begin{bmatrix} U & U + a\sqrt{\xi^2 + \xi_y^2} & U - a\sqrt{\xi^2 + \xi_y^2} \\ \end{bmatrix},$$

(A.2a)

$$\Lambda_\eta = \begin{bmatrix} V & V + a\sqrt{\eta^2 + \eta_y^2} & V - a\sqrt{\eta^2 + \eta_y^2} \\ \end{bmatrix},$$

(A.2b)

with

$$T_\kappa = \begin{bmatrix} 1 & -\tilde{\kappa}_x & 0 \\ u & \tilde{\kappa}_y & (u + \tilde{\kappa}_a) \\ v & -\tilde{\kappa}_x & (v + \tilde{\kappa}_a) \\ \phi^2 & -\phi^2 & \phi^2 + a^2 \\ (\gamma - 1) & (\gamma - 1) & (\gamma - 1) + a\tilde{\theta} \\ \end{bmatrix},$$

(A.3)

$$T^{-1}_\kappa = \begin{bmatrix} (1 - \phi^2/a^2) & (\gamma - 1)u/a^2 & (\gamma - 1)v/a^2 & -(\gamma - 1)/a^2 \\ -(\tilde{\kappa}_y u - \tilde{\kappa}_x v) & \tilde{\kappa}_y & -\tilde{\kappa}_x & 0 \\ \beta(\phi^2 - a\tilde{\theta}) & \beta[\tilde{\kappa}_a - (\gamma - 1)u] & \beta[\tilde{\kappa}_a - (\gamma - 1)v] & \beta(\gamma - 1) \\ \beta(\phi^2 + a\tilde{\theta}) & -\beta[\tilde{\kappa}_a + (\gamma - 1)u] & -\beta[\tilde{\kappa}_a + (\gamma - 1)v] & \beta(\gamma - 1) \\ \end{bmatrix},$$

(A.4)

and $\beta = 1/(2a^2)$, $\tilde{\theta} = \tilde{\kappa}_x u + \tilde{\kappa}_y v$, and, for example, $\tilde{\kappa}_x = \kappa_x/\sqrt{\kappa_x^2 + \kappa_y^2}$.

Relations exist between $T_\xi$ and $T_\eta$ of the form

$$\hat{N} = T^{-1}_\xi T_\eta, \quad \hat{N}^{-1} = T^{-1}_\eta T_\xi$$

(A.5)

where

$$\hat{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & m_1 & -\mu m_2 & \mu m_2 \\ 0 & \mu m_2 & \mu^2(1 + m_1) & \mu^2(1 - m_1) \\ 0 & -\mu m_2 & \mu^2(1 - m_1) & \mu^2(1 + m_1) \\ \end{bmatrix},$$

(A.6a)

and

$$\hat{N}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & m_1 & \mu m_2 & -\mu m_2 \\ 0 & -\mu m_2 & \mu^2(1 + m_1) & \mu^2(1 - m_1) \\ 0 & \mu m_2 & \mu^2(1 - m_1) & \mu^2(1 + m_1) \\ \end{bmatrix},$$

(A.6b)

with $m_1 = (\xi_x \hat{\eta}_x + \xi_y \hat{\eta}_y)$, $m_2 = (\xi_x \hat{\eta}_y - \xi_y \hat{\eta}_x)$ and $\mu = 1/\sqrt{2}$.

It is interesting to note that the matrix $\hat{N}$ is only a function of the metrics and not the flow variables.
References


