JOURNAL OF COMPUTATIONAL PHYSICS 104, 1–13 (1993)

Acceleration of Convergence and Spectrum Transformation of Implicit Finite Difference Operators Associated with Navier–Stokes Equations

M. SALEEM

Department of Mathematics & Computer Science, California State University, San Jose, California 95192

T. PULLIAM

Division of Computational Physics, NASA Ames Research Center, Moffett Field, California 94035

AND

A. Y. CHEER

Department of Mathematics, University of California, Davis, California 95616

Received April 5, 1991; revised January 9, 1992

Eigensystem analysis techniques are applied to finite difference formulations of Euler and Navier–Stokes equations in two dimensions. Spectrums of the resulting implicit difference operators are computed. The convergence and stability properties of the iterative methods are studied by taking into account, the effect of grid geometry, time-step, numerical dissipation, viscosity, boundary conditions, and the physics of the underlying flow. The largest eigenvalues are computed by using the Frechet derivative of the operators and Arnoldi's method. The accuracy of Arnoldi's method is tested by comparing the dominant eigenvalues with the rate of convergence of the iterative method. Based on the pattern of eigenvalue distributions for various flow configurations, the feasability of applying existing convergence-acceleration techniques like *eigenvalue annihilation and relaxation* are discussed. Finally a *shifting* of the implicit operators in question is devised. The idea of shifting is based on the power method of linear algebra and is *very simple to implement. The procedure of shifting the spectrum is* applied to ARC2D, a flow code developed and being used at NASA Ames Research Center. When compared to eigenvalue annihilation, the shifting method clearly establishes its superiority. For the ARC2D code, an efficiency of 20 to 33% has been achieved by this method. © 1993 Academic Press, Inc.

I. INTRODUCTION

As the 21st century approaches, the need for faster and more efficient aircraft becomes more pronounced. In designing an aircraft, the aeronautical engineer must do a lot of experimentation involving a wind tunnel. Such testing is expensive and therefore cannot accomodate the effects of "infinitely many" parameters associated with aircraft flight. The computational fluid dynamicist, on the other hand, is able to perform numerical simulations for many of the situations likely to occur in flight. In addition, numerical experimentation is less costly as compared to wind tunnel testing. This makes CFD a powerful tool for studying aircraft flight. This tool cannot completely replace conventional testing but is extremely desirable because of its versatility.

There are many limitations in present day computers, however. Rounding and truncation on "finite-precision" computers are known to swamp a computation. Speed and storage of a computer are other factors which hinder our calculations in general. This paper addresses a technique for speeding-up certain calculations by shifting the spectrum of finite-difference operators associated with Navier–Stokes solvers.

ARC2D [5, 10] is a computer code that solves Navier–Stokes equations for arbitrary two-dimensional geometries. This code was developed at NASA Ames Research Center and is currently in use there. When Navier–Stokes equations in two dimensions are discretized, the resulting matrix system can be solved iteratively. The iterative process generates the solution, but takes unusually long to converge. The problem of reducing the CPU time in such a computation, by accelerating the iterative process, is attracting many scientists. This paper presents an acceleration-convergence technique for general iterative schemes, particularly ARC2D. The method given here does not depend on a particular Navier–Stokes solver, nor on the computer architecture.

There are various ways of improving the rate of convergence of an iterative method. Some of them are eigenvalue annihilation [14], multigrid [20], and artificial viscosity models [13]. The method described here depends on improving the rate of convergence of a method by *shifting* the dominant eigenvalues of the iterative matrix. This matrix can be modified without altering the steady state solution, by forcing the new eigenvalues to be smaller in magnitude. The process speeds up convergence.

Arnoldi's method is used to compute a few large eigenvalues of the time-dependent matrices appearing in the iterative formulation of Navier–Stokes equations. The patterns of eigenvalues for different flow configurations are studied. Based on these patterns, a *shifting* of the spectrums of implicit operators (matrices) is devised. The amount of *shift* for a specific case depends on the ends of the spectrum, which is available from Arnoldi's method. The *shifted* ARC2D code shows that convergence can be accelerated by a proper choice of the *shift* factor.

The results obtained are compared with the method of explicit eigenvalue annihilation. The shifting strategy is found to be superior, as shown in Sections IV and V. In Section V it is shown that a relaxation scheme will not work for ARC2D, in general. Section VI shows the improvement obtained due to the shifting method. A savings of between 20 and 33% is clearly visible from the tables and graphs.

II. NAVIER–STOKES EQUATIONS

In two dimensions, the equations to be solved are

$$\frac{\partial Q}{\partial t}+\frac{\partial E}{\partial x}+\frac{\partial F}{\partial y}=\mathrm{Re}^{-1}\left(\frac{\partial E_v}{\partial x}+\frac{\partial F_v}{\partial y}\right), \tag{1}$$

where the symbols have their usual meaning [3, 5]. Equations (1) can be transformed from Cartesian coordinates to general curvilinear coordinates by using the transformation,

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

It is assumed that this transformation has sufficient smoothness properties, so that the corresponding Jacobian J is non-singular. With this understanding, the transformed equations are

$$\frac{\partial \hat{Q}}{\partial t}+\frac{\partial \hat{E}}{\partial \xi}+\frac{\partial \hat{F}}{\partial \eta}=\mathrm{Re}^{-1}\left(\frac{\partial \hat{E}_v}{\partial \xi}+\frac{\partial \hat{F}_v}{\partial \eta}\right), \tag{2}$$

where $\hat{Q}=J^{-1}Q$,

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

with $U=\xi_t+\xi_x u+\xi_y v$ and $V=\eta_t+\eta_x u+\eta_y v$ as the contravariant velocities.

The viscous flux terms are $\hat{E}=J^{-1}(\xi_x E_v+\xi_y F_v)$ and $\hat{F}=J^{-1}(\eta_x E_v+\eta_y F_v)$. The stress terms such as τ_{xx} are also transformed in terms of the ξ and η derivatives and can be found in [3, 4]. To be able to include viscous effects which are concentrated near rigid boundaries and in wake regions, a thin-layer approximation is applied to Eqs. (2). The thin layer approximation is similar in philosophy but not the same as the boundary layer theory. All viscous derivatives in the ξ direction are neglected, while terms in the η direction are retained. The inviscid terms are included in their entirety. Equations (2) thus become

$$\frac{\partial \hat{Q}}{\partial \tau}+\frac{\partial \hat{E}}{\partial \xi}+\frac{\partial \hat{F}}{\partial \eta}=\mathrm{Re}^{-1}\frac{\partial \hat{S}}{\partial \eta}. \tag{3}$$

where the form of $\hat{S}$ can be found in [3, 5]. The Beam–Warming scheme [11] is next applied to Eq. (3). This results in

$$\hat{Q}^{n+1}-\hat{Q}^n+h(\hat{E}_\xi^{n+1}+\hat{F}_\eta^{n+1}-\mathrm{Re}^{-1}\hat{S}_\eta^{n+1})=0, \tag{4}$$

where $h=\Delta\tau$.

Equation (4) must now be solved for $\hat{Q}^{n+1}$, once a $\hat{Q}^n$ is given. The flux vectors $\hat{E}$, $\hat{F}$, and $\hat{S}$ are nonlinear functions of $\hat{Q}^n$ and therefore Eq. (4) is nonlinear in $\hat{Q}^{n+1}$. These nonlinear terms are linearized in time about $\hat{Q}^n$, by using a Taylor series. Thus,

$$\begin{aligned}\hat{E}^{n+1}&=\hat{E}^n+\hat{A}^n(\hat{Q}^{n+1}-\hat{Q}^n)+O(h^2)\\ &=\hat{E}^n+\hat{A}^n\,\Delta\hat{Q}^n+O(h^2)\\ \hat{F}^{n+1}&=\hat{F}^n+\hat{B}^n\,\Delta\hat{Q}^n+O(h^2)\\ \mathrm{Re}^{-1}\hat{S}^{n+1}&=\mathrm{Re}^{-1}[\hat{S}^n+J^{-1}\hat{M}^n\,\Delta\hat{Q}^n]+O(h^2),\end{aligned}$$

where $\hat{A}=\partial\hat{E}/\partial\hat{Q}$, $\hat{B}=\partial\hat{F}/\partial\hat{Q}$, $\hat{M}=\partial\hat{S}/\partial\hat{Q}$ are the flux Jacobians, and $\Delta\hat{Q}^n=O(h)$. The explicit forms of the flux Jacobians can be found in [3, 4]. Equation (4) now becomes

$$\begin{aligned}&[I+h\,\partial_\xi\hat{A}^n+h\,\partial_\eta\hat{B}^n-\mathrm{Re}^{-1}J^{-1}h\,\partial_\eta\hat{M}^n]\,\Delta\hat{Q}^n\\ &\quad=-h[\partial_\xi\hat{E}^n+\partial_\eta\hat{F}^n-\mathrm{Re}^{-1}\partial_\eta\hat{S}^n]\end{aligned} \tag{4.1}$$

which in future reference will be called the *unfactored form* of the block algorithm.

Since the integration of the full 2D operator is too expensive, an approximate factorization of the following form is used:

$$\begin{aligned}&[I+h\,\partial_\xi\hat{A}^n+h\,\partial_\eta\hat{B}^n-\mathrm{Re}^{-1}h\,\partial_\eta\hat{M}^n]\,\Delta\hat{Q}^n\\ &\quad=[I+h\,\partial_\xi\hat{A}^n][I+h\,\partial_\eta\hat{B}^n-\mathrm{Re}^{-1}h\,\partial_\eta\hat{M}^n]\,\Delta\hat{Q}^n\\ &\qquad-h^2\,\partial_\xi\hat{A}^n\,\partial_\eta\hat{B}^n-h^2\,\mathrm{Re}^{-1}\,\partial_\xi\hat{A}^n\,\partial_\eta\hat{M}^n\,\Delta\hat{Q}^n.\end{aligned} \tag{5}$$

The cross term is second-order accurate since $\Delta\hat{Q}^n$ is $O(h)$. It can therefore be neglected without degrading the time accuracy of any second-order scheme that may be chosen. The resulting factored form of Eq. (5) is,

$$[I+h\,\partial_\xi\hat{A}^n][I+h\,\partial_\eta\hat{B}^n-\mathrm{Re}^{-1}\,h\,\partial_\eta\hat{M}^n]\,\Delta\hat{Q}^n = -h[\partial_\xi\hat{E}^n+\partial_\eta\hat{F}^n-\mathrm{Re}^{-1}\,\partial_\eta\hat{S}^n]. \qquad (6)$$

This gives two implicit operators, each of which is block tridiagonal. The solution procedure consists of two one-dimensional sweeps: one in the ξ direction and one in the η direction.

As a further simplification, Eq. (6) can be broken into its diagonal version by noting that the flux Jacobians $\hat{A}$ and $\hat{B}$ have real eigenvalues and a complete set of eigenvectors, as given in the appendix of [5]. Thus they can be diagonalized as

$$\Lambda_\xi = T_\xi^{-1}\hat{A}T_\xi, \quad \Lambda_\eta = T_\eta^{-1}\hat{B}T_\eta,$$

where T_ξ and T_η are the matrices of eigenvectors of $\hat{A}$ and $\hat{B}$, respectively. With this choice, Eq. (6) becomes $[T_\xi T_\xi^{-} + h\delta_\xi(T_\xi\Lambda_\xi T_\xi^{-1})][T_\eta T_\eta^{-1}+h\,\delta_\eta(T_\eta\Lambda_\eta T_\eta^{-1})]\,\Delta\hat{Q}^n$ = explicit RHS of (6) $=\hat{R}^n$ or,

$$T_\xi(I+h\,\delta_\xi\Lambda_\xi)\,\hat{N}(I+h\,\delta_\eta\Lambda_\eta)\,T_\eta^{-1}\,\Delta\hat{Q}^n=\hat{R}, \qquad (7)$$

where $\hat{N}=T_\xi^{-1}T_\eta$.

Computational experiments by Pulliam and Chausse [12] show that the convergence and stability limits of the diagonal algorithm are identical to that of the unmodified algorithm. Furthermore, the diagonal algorithm reduces the block tridiagonal inversions to 4×4 matrix multiples and scalar tridiagonal inversions.

In order to improve the stability bounds of the algorithm, numerical dissipation is added. The model to be employed is again from [13].

III. ITERATIVE SOLUTION PROCESS

Our objective here is to write the factored form of ARC2D algorithm in a simple iterative version. This will help us in identifying the iteration matrix and obtaining a subset of its spectrum. Finally, this spectrum will be analyzed and the amount of shift (necessary for accelerating the convergence) will be calculated. Equation (6) can be written in an operator form as

$$L_\xi G_\eta\,\Delta\hat{Q}^n=\hat{R}^n,$$

where L_ξ and G_η are the block ξ and η operators. Dropping the "hats," and inverting the blocks, leads to $\Delta Q^n = G_\eta^{-1}L_\xi^{-1}R^n$, or $Q^{n+1}=Q^n+G_\eta^{-1}L_\xi^{-1}R^n$, with $L(Q^n)=Q^n+G_\eta^{-1}L_\xi^{-1}R^n$. This process can be condensed into an abstract functional equation,

$$Q^{n+1}=L(Q^n). \qquad (8)$$

Clearly, L is a nonlinear transformation in a certain Banach space setting. This L has the effect of flux Jacobians, coordinate geometries, viscosities, numerical dissipations, initial and boundary conditions built into it. Its exact analytic form is not available. Inspite of this difficulty, a study of the convergence properties of Eq. (8) is performed by looking at the eigensystem associated with it. This study will be useful in accelerating the convergence of this equation.

When the ARC2D is put in the iterative form $Q^{n+1}=L(Q^n)$, an application of L on Q^n means that the following steps have been performed:

(a) obtaining a mesh in the ξ, η space.

(b) setting up initial conditions/data for the flow.

(c) setting up boundary conditions.

(d) computing the right hand side $\hat{R}^n$ which has viscous effects and explicit dissipation terms.

(e) filling the block tridiagonal operators L_ξ and G_η, and adding to them implicit dissipation if required.

(f) finally, inverting the factored ξ and η blocks to extract the solution vector Q^{n+1}.

The procedure (a) to (f) outlined above is one iteration of the ARC2D code.

As explained in [16], the Jacobian $\mathring{\mathrm{A}}=dL/dQ^*$, where Q^* is a reference solution of (8), controls the convergence of Eq. (8). With *jmax* grid points in the ξ direction, *kmax* grid points in the η direction, and four unknowns at each grid point, the dimension of matrix $\mathring{\mathrm{A}}$ is $(4\cdot jmax\cdot kmax)$. This is a large number even when *jmax* and *kmax* are small. This study analyzes the spectrum of $\mathring{\mathrm{A}}$ associated with flow past a NACA0012 airfoil, superimposed on a non-periodic 127×32 C-mesh. In this case the Jacobian matrix $\mathring{\mathrm{A}}$ has $(4\times127\times32)^2=(16{,}256)^2$ elements. This is a figure that is larger than 2.6×10^8. The computation and storage of this huge matrix is impossible on the CRAY-XMP48 supercomputer. In order to obtain some insight into the spectrum of $\mathring{\mathrm{A}}$, Arnoldi's method [1, 7] must be used. Its algorithm to compute the dominant eigenvalues of $\mathring{\mathrm{A}}$ is described below:

For an arbitrary vector $\mathbf{q}_1$, define,

$$\hat{\mathbf{q}}_1=\frac{\mathbf{q}_1}{\|\mathbf{q}_1\|}.$$

The algorithm then consists of the following steps:

For $k = 1$ to m, do

$$\mathbf{q}_{k+1} = \mathring{\mathbf{A}}\hat{\mathbf{q}}_k - \sum_{j=1}^{k} c_{jk}\hat{\mathbf{q}}_j$$

$$c_{jk} = \langle \hat{\mathbf{q}}_j, \mathring{\mathbf{A}}\mathbf{q}_k \rangle$$

where $\langle \mathbf{x}, y \rangle$ = Inner product of the vectors $\mathbf{x}$ and $\mathbf{y}$

$$\hat{\mathbf{q}}_{k+1} = \frac{\mathbf{q}_{k+1}}{\|\mathbf{q}_{k+1}\|}$$

next k.

The m eigenvalues of $C = c_{ij}$ are the approximations to "some" m eigenvalues of $\mathring{\mathbf{A}}$. Products like $\mathring{\mathbf{A}}v$ which are needed for arbitrary v can be achieved by using the idea of the Frechet derivative of L as

$$\mathring{\mathbf{A}}v = \frac{dL}{dQ^*}v = \frac{L(Q^* + \varepsilon v) - L(Q^* - \varepsilon v)}{2\varepsilon} + O(\varepsilon^2).$$

It should, however, be noted that even if 10% of the 16,256 eigenvalues are extracted, the upper Hessenberg matrix C of Arnoldi's method should be 163×163. This amounts to demanding a lot in terms of computer CPU time. In extracting 163 eigenvalues of the ARC2D operator, 60 sec of CPU on the CRAY-XMP48 would be used. This includes $2 \times 163 = 326$ iterations of the code and the time for EIGRF [2] to compute the eigensystem.

Keeping this in mind, only 50 eigenvalues of the ARC2D operator were approximated by Arnoldi's method. This was done iteratively by running Arnoldi's process twice, for a variety of problems. Occasionally, 100 eigenvalues, comprising 6% of the entire spectrum were also computed. In forming the Frechet derivative for the one-dimensional case [16],

$$\mathring{\mathbf{A}}v = \frac{dL}{dQ^*}v = \frac{L(Q^* + \varepsilon v) - L(Q^* - \varepsilon v)}{2\varepsilon} + O(\varepsilon^2),$$

the ε was chosen from the requirement, $\varepsilon = 0.001 * \|Q^*\|/\|v\|$). In the present case of ARC2D, this criterion had to be changed. Here ε was taken to be of the same order of magnitude as the residual at that iteration. Choosing a bigger epsilon perturbed the solution to a larger degree than was desired and resulted in an unstable process. Similarly, when a smaller epsilon ($\varepsilon <$ residual) was taken, then the $(Q^* + \varepsilon v)$ was a very slight perturbation of Q^*, and the code did not differentiate $(Q^* + \varepsilon v)$ from $(Q^* - \varepsilon v)$. In that case, cancellation occured while calculating $\{L(Q^* + \varepsilon v) - L(Q^* - \varepsilon v)\}$ in the Frechet derivative. The choice of ε was critical in obtaining the spectrum of $\mathring{\mathbf{A}}$, as explained in the next paragraph.

Assuming a rounding error of δ in one application of the operator L, it is easy to see that a maximum rounding error of $2\delta/2\varepsilon$ is possible in the product $\mathring{\mathbf{A}}\mathrm{v}$. Thus the Frechet derivative has a total error (roundoff and truncation) of $\delta/\varepsilon + O(\varepsilon^2)$. Decreasing epsilon with the hope of obtaining an accurate Frechet derivative will actually destroy it with roundoff errors propagating through the δ/ε term. The computation of an optimal perturbation, ε, is theoretically possible. It will, however, require knowledge of the spectrum of some large sparse matrix—a complex problem in itself, not being addressed here. The dependence of $\mathring{\mathbf{A}}v$ (and hence the eigenvalues of the iterative matrix) on the perturbation ε is therefore quite natural, as will be seen in Section IV.

Moreover, in the viscous calculations, the solution changes so rapidly near the body surface, that a slight change in epsilon excites the solution and hence the Frechet derivative. Thus λ will be less accurate for a viscous case, making it difficult to accelerate. In future studies, as a test, epsilon based on the pointwise residual will be used. For the viscous case, for example, epsilon can be taken to be of the same order as the minimum pointwise residual. This question will not be discussed at the moment.

When the solution vector Q^n at the nth iteration was taken as the starting vector v for Arnoldi's method, the perturbation $(Q^* + \varepsilon v)$ turned out to be a perfectly balanced one. Thus the small components of Q^* were perturbed slightly, whereas the larger ones were perturbed a lot. With this choice, the convergence of Arnoldi's method was remarkable and suggested that Q^n as a starting vector was indeed very appropriate. This choice did not require the construction, $\varepsilon = 0.001 * (\|Q^*\|/\|v\|)$ as proposed by Eriksson and Rizzi [8].

IV. ACCURACY OF ARNOLDI'S METHOD FOR ARC2D

In the one-dimensional nozzle problem [16], establishing the accuracy of Arnoldi's method was easy. This was due to the fact that the complete numerical Jacobian $\mathring{\mathbf{A}} = dL/dQ^*$ and its spectrum were both available even for the largest 300×300 case. Here, in the two-dimensional problem, as already said, the matrices encountered are larger than $16{,}000 \times 16{,}000$. So a new approach is adopted. This section compares the largest eigenvalue of the ARC2D operator, obtained by using Arnoldi's process with the rate of convergence of the code. These quantities agree to within 10^{-3} in most cases, as will be discussed below.

The airfoil under consideration is NACA0012, with a non-periodic 127×32 C-mesh. Other default values of the constants of flow are

TINF = temperature at infinity = 460
DIS2X = DIS2Y = second-order dissipations = 1.0
DIS4X = DIS4Y = fourth-order dissipations = 0.64
ALPHA = angle of attack = 0.

Subsonic Flow

Assuming inviscid flow and a Mach number of 0.4, the code was run to a steady state, and the solution Q^* stored at 600 iterations. For the Frechet derivative, an epsilon of 10^{-7} (which is actually much bigger than the residual at 600 iterations) was taken. On extracting 100 eigenvalues with Arnoldi's method, it was found that

$$\lambda_{\max} = 0.9326 \pm i0.21266$$

$$|\lambda_{\max}| = 0.95656$$

which is exactly the same as the rate of convergence of the code at 600 iterations. The rate of convergence, r, was calculated by the formula [18]

$$r = \left[\frac{\text{residual at } N \text{ iterations}}{\text{residual at } M \text{ iterations}}\right]^{\frac{1}{N-M}}. \qquad (9)$$

Some authors prefer to call it the amplification factor. They define $-\log(r)$ as the rate of convergence. In this study, the rate defined from Eq. (9) will be adopted throughout, because of its similarity with the largest eigenvalue.

For this case, 50 eigenvalues and the corresponding eigenvectors were also extracted. Then using the "dominant" eigenvector as the starting vector for Arnoldi's method, the iterated spectrum was obtained. $\lambda_{\max}$ was found to be $0.93278 \pm i0.21242$ and $|\lambda_{\max}| = 0.95675$, which is within 0.0002 of the previous result. These eigenvalues appear in a circular distribution in the right half plane, but within the unit circle, as shown in Fig. 1.

A similar test was done with viscous flow and a Reynold's number of 10,000. The code was only run to 200 iterations. Laminar conditions were assumed. With a residual of 0.779×10^{-6} and a rate of convergence of 0.986178, 50 eigenvalues were extracted using $\varepsilon = 10^{-7}$. Here $|\lambda_{\max}| = |0.99127 + i0.0| = \text{rate} \pm 0.005$. Since an epsilon of 10^{-6} was also possible in this case, it was used and found that $|\lambda_{\max}| = |0.97803 \pm i0.01817| = 0.9782$. Clearly, in this case, $|\lambda_{\max}| = \text{rate of convergence} - 0.008$.

Using the same conditions of viscous flow at Mach 0.4, a subset of the spectrum at 630 iterations was also computed, to give, $\lambda_{\max} = 0.9914 + i0.0$, which is plotted in Fig. 2. This is a discrepancy of only 0.0019.

Transonic Flow

The results for subsonic flow presented the authors with enough motivation to compute spectrums in the transonic range. The Mach number was set at 0.8, and for inviscid flow the code was run to 200 iterations. Fifty eigenvalues were extracted with epsilon = 10^{-7} to give $|\lambda_{\max}| = |0.97058 \pm i0.0883| = 0.97463536$. The rate of convergence at 200 iterations was 0.9755 which is within 0.0009 of $\lambda_{\max}$. At 400 iterations the rate of convergence was 0.974534, whereas

$$|\lambda_{\max}| = |0.970548 \pm i0.088547| = 0.9745789.$$

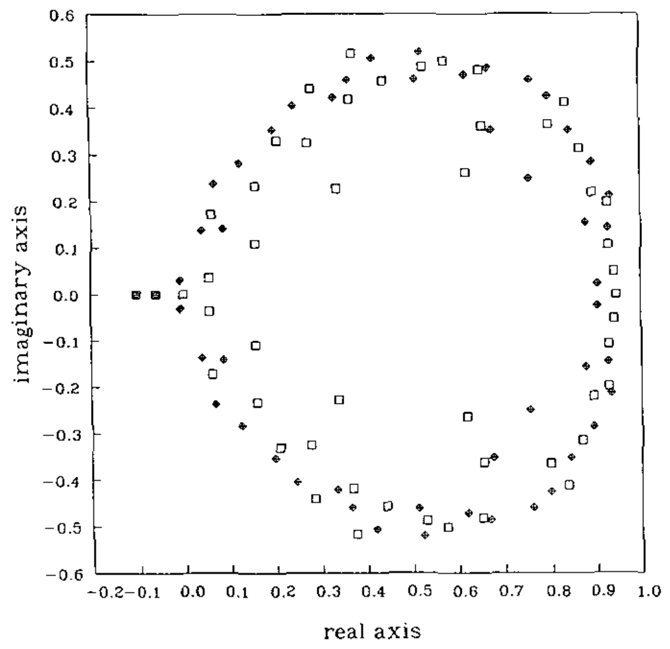

FIG. 1. ARC2D, 127 × 32 C-mesh, 50/16,256 eigenvalues, Mach 0.4, 600 iterations: □ = first iteration of Arnoldi, with epsilon = 1.0*E*-03; ⊕ = second iteration.

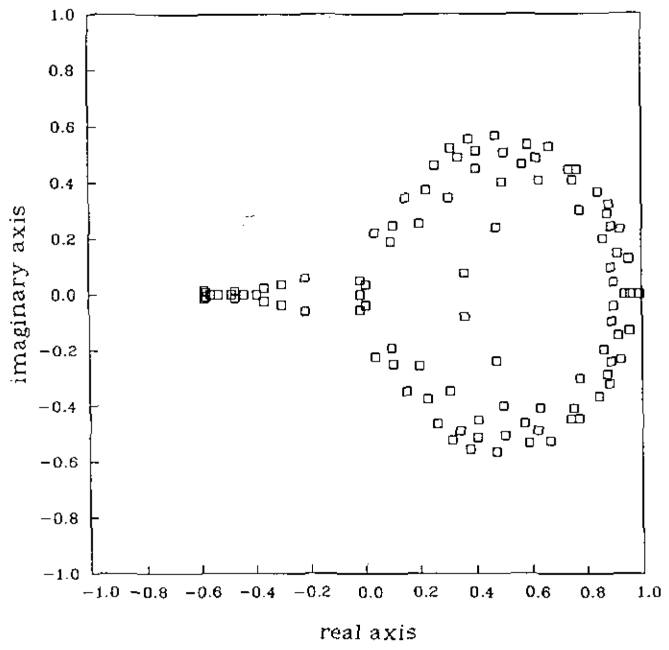

FIG. 2. ARC2D, 127 × 32 viscous C-mesh, Re = 10,000, Mach 0.4, 620 iterations, 100/16,256 eigenvalues: □ = first iteration of Arnoldi.

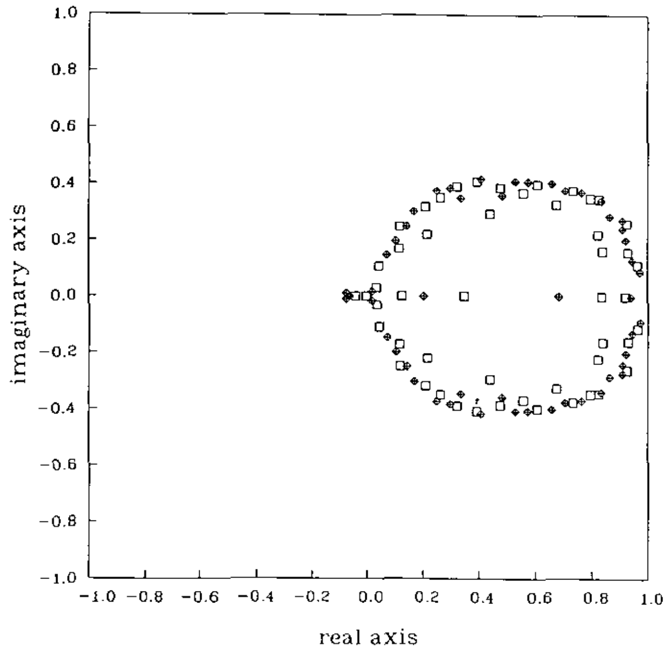

FIG. 3. ARC2D, 127 × 32 inviscid C-mesh, Mach 0.8, 600 iterations, 50/16,256 eigenvalues: □ = first iteration of Arnoldi; ⊕ = second iteration of Arnoldi.

Here the difference between $|\lambda_{\max}|$ and the rate of convergence was less than 0.00005. Similarly, at 600 iterations of the code, the rate of convergence was 0.974866, and as shown in Fig. 3, $|\lambda_{\max}| = |0.9709 \pm i0.08868| = 0.97495273$, a difference of about 0.0008.

With the addition of viscous terms at Mach 0.8 and Reynold's number 10,000, the calculations were carried out at 200 iterations only. The rate of convergence, as computed from Eq. (9) was 0.9876, whereas $|\lambda_{\max}| = |0.99108 \pm i0.0497950| = 0.99233$ as shown in Fig. 4. Here, $|\lambda_{\max}| =$ rate of convergence $+0.004$.

The Effect of Other Algorithm Parameters

In order to see the effect of fourth-order smoothing DIS4X and DIS4Y, the spectrum for various cases was obtained. These smoothing parameters were varied from 0.0 up to 2.56 for the viscous transonic case. The plots are shown in Fig. 4 through 7.

With small values of these parameters, the eigenvalues are large, some with positive real parts and others with negative; $\lambda_{\max} - \lambda_{\min}$ is as large as 1.9 in some cases. As the smoothing becomes larger than 1.0, the smaller circle of eigenvalues in the left half plane begins to disappear. With a smoothing factor of 2.56, a very compact picture (Fig. 7) is obtained with $\lambda_{\max} - \lambda_{\min} = 1.5$.

The variation of smoothing terms was studied in the factored, block tridiagonal version of ARC2D also. The results were similar to the pentadiagonal version, the only difference being that $\lambda_{\max}$ was generally smaller in the block tridiagonal case. With the dissipation taken as DIS4X = DIS4Y = 0.0 and 0.16, there were some eigenvalues larger than 1.0, as expected. Then as the smoothing factors increased from 1.0 to 2.0, the smaller circle of eigenvalues in the left half plane began to disappear as before. This is pictured in the plots. See Fig. 8 through 10.

Among a variety of other options, the code ARC2D has

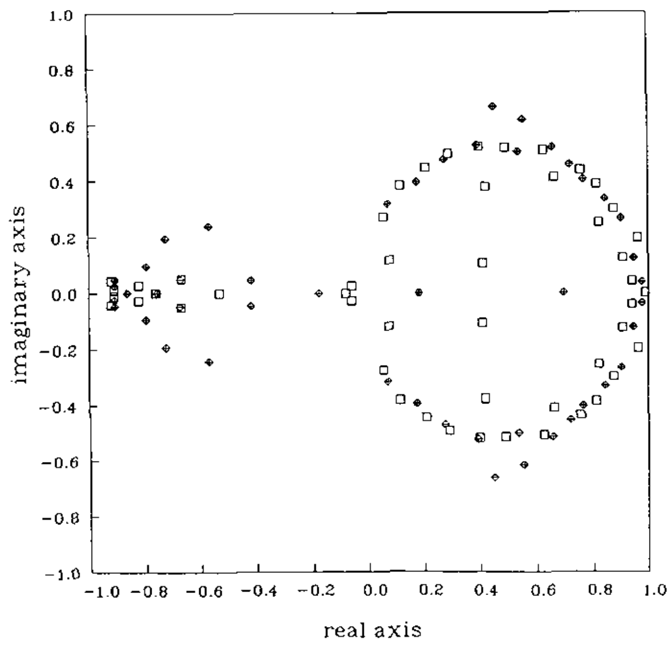

FIG. 4. ARC2D 127 × 32 viscous C-mesh, Re = 10,000, Mach 0.8, diss 4 = 0, 200 iterations: ⊕ = second Arnoldi iteration.

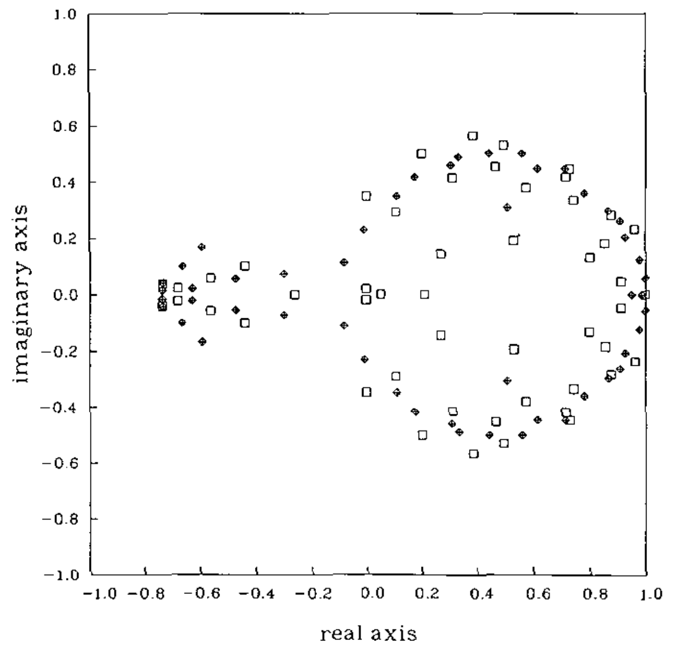

FIG. 5. ARC2D 127 × 32 viscous C-mesh, Re = 10,000, Mach 0.8, diss 4 = 0.64, 200 iterations: ⊕ = second Arnoldi iteration.

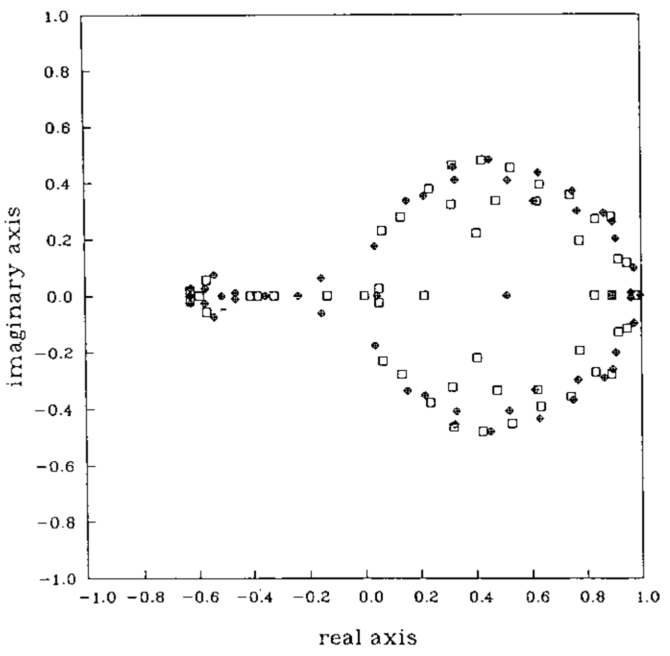

FIG. 6. ARC2D 127 × 32 viscous C-mesh, Re = 10,000, Mach 0.8, diss 4 = 1.28, 210 iterations: ⊕ second Arnoldi iteration.

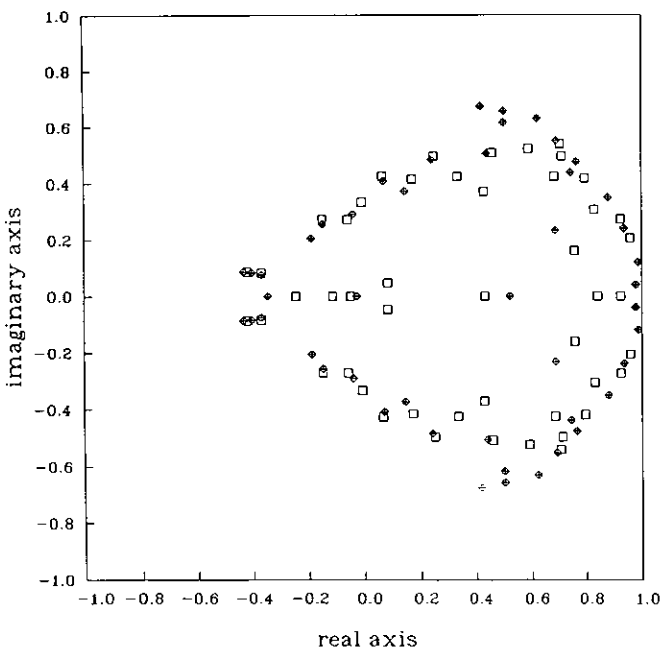

FIG. 8. Block ARC2D 127 × 32 viscous C-mesh, Re = 10,000, Mach 0.8, diss 4 = 0.0, 200 iterations: ⊕ = second Arnoldi iteration.

a number of strategies for choosing a time step procedure. The variable *jacdt* is a switch to turn on constant or scaled step sizes. For instance, *jacdt* = 0 means that a constant step size is to be taken; *jacdt* = 1 means that the time step is scaled according to the metric Jacobian. This choice allows for larger time steps, when the Jacobian is large, especially near the body surface:

jacdt = 2, selects a time step based on the minimum eigenvalue.

jacdt = 3, selects a time step based on constant CFL number at each point.

The default option of the code is *jacdt* = 1. A representative case of inviscid flow at Mach 0.8 was studied with different

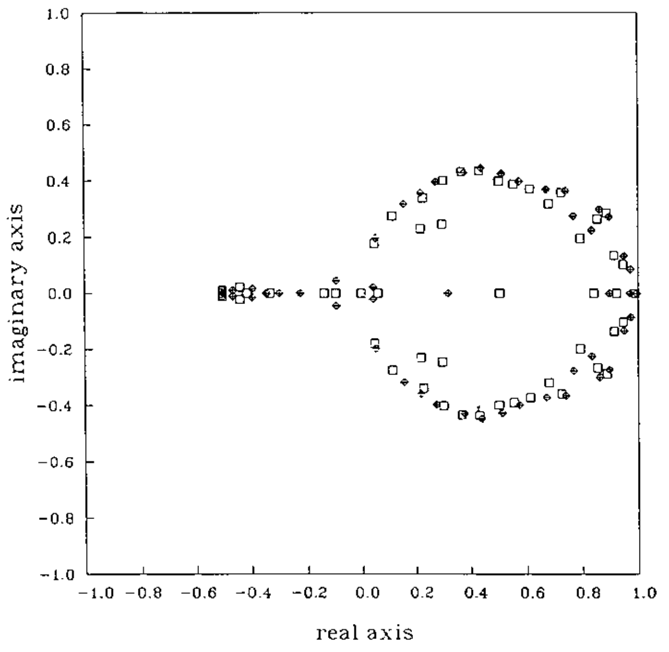

FIG. 7. ARC2D 127 × 32 viscous C-mesh, Re = 10,000, Mach 0.8, diss 4 = 2.56, 210 iterations: ⊕ = second Arnoldi iteration.

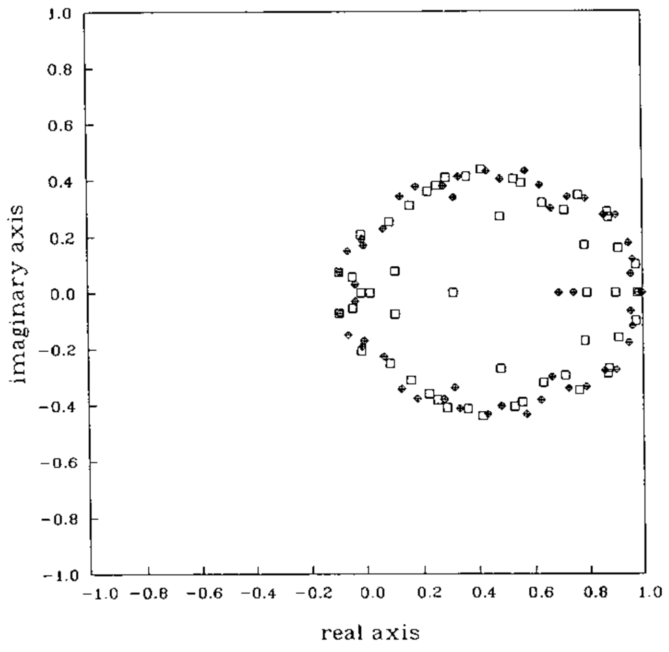

FIG. 9. Block ARC2D 127 × 32 viscous C-mesh, Re = 10,000, Mach 0.8, diss 4 = 1.0, 210 iterations: ⊕ = second Arnoldi iteration.

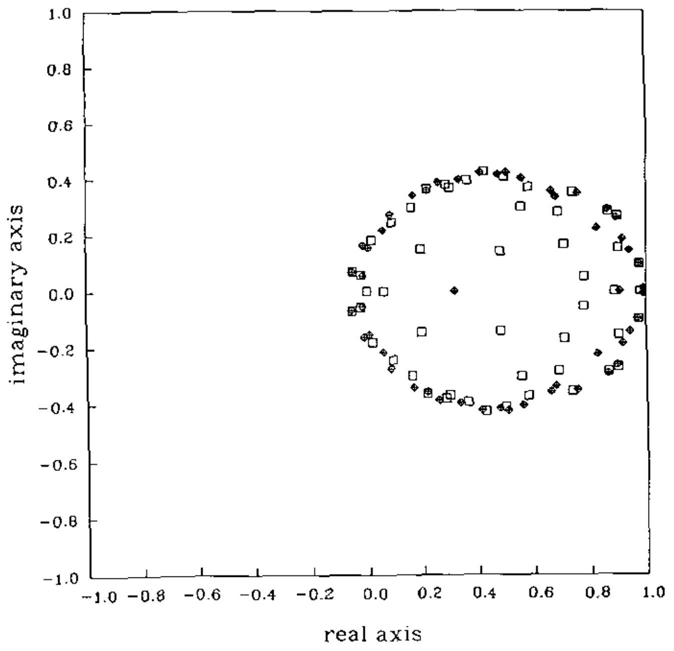

FIG. 10. Block ARC2D 127×32 viscous C-mesh, Re = 10,000, Mach 0.8, diss 4 = 2.0, 200 iterations: ⊕ = second Arnoldi iteration.

values of *jacdt*. First, the code was run to 500 iterations, where the residual fell below 10^{-9}. This converged solution was then saved, and the spectrum of the implicit operator was obtained. The largest eigenvalue was seen to be $0.9705 \pm i0.08$. This eigenvalue will be denoted by Λ, to serve as a reference. The optimal value of dt that the code had picked was 0.816.

Then at 500 iterations, a constant step size of $dt = 0.08$ (with $jacdt = 0$) was introduced into the code and the spectrum was again computed. The largest eigenvalue for this approach was found to be $0.9905 \pm i0.009$, which is considerably larger than the reference eigenvalue Λ, defined above. This test suggested that a time step of 0.08, which is 10 times smaller than 0.816, slowed down the convergence. This was also observed by looking at the increase in residuals with this time step. The effect of increasing dt to a fixed value of 1.0 was tested similarly. The largest eigenvalue came out to be $0.923 \pm i0.636$ which has an absolute value of 1.121. This large value of the time step makes the code unstable. This occurs despite the fully implicit construction of the ARC2D code. It certainly does not reflect as a shortcoming of the code, but rather suggests the difficulty in achieving stability for a nonlinear process with shock waves, as described by the Navier–Stokes equations. It also agrees with the well-known fact that implicit methods are conditionally stable for most nonlinear problems. A simple example is that of backward Euler method (which is implicit) applied to the nonlinear problem $y' = -\lambda y^2$.

When the variable time step was chosen according to the constant CFL number, the dominant eigenvalue was found to be $0.98508 \pm i0.0744$, which is larger than Λ in absolute value. Similarly, with dt based on the minimum eigenvalue, the spectrum of the implicit operator showed $0.998 \pm i0.0407$ as the dominant eigenvalue.

These simple experiments suggested that the time step based on the metric Jacobians is the optimal strategy for choosing dt in the ARC2D code. Very small or very large time steps do not improve the rate of convergence.

As a final experiment, a time step sequencing strategy was tried by fixing the time step as follows:

For 1–100 iterations, $dt = 0.001$
For 101–200 iterations, $dt = 0.005$
For 201–300 iterations, $dt = 0.02$
For 301–600 iterations, $dt = 0.05$.

The largest eigenvalue at 600 iterations was 0.9976 which is bigger than Λ. Small steps like these did not improve the rate of convergence of the code as compared with the case $jacdt = 1$.

Summary of These Results

From Figs. 1 to 10 it is obvious that the inviscid ARC2D operator has an approximately circular distribution of eigenvalues. This circle, in most cases, is centered at $x = 0.5$ and has a radius of 0.5. This definite pattern gave rise to the question of translating the above-mentioned *circle*, so that its new center be at the origin. This means that the new largest eigenvalue would be given by $\lambda = 0.5$. With this translation or *shift*, the rate of convergence can theoretically be improved from around $r = 1.0$ to $r = 0.5$. This idea is addressed in the next section.

The spectrum of the *viscous* ARC2D operator with Re = 10,000 has one noticeable feature as compared to the *inviscid* case. It has two distinct circular distributions of eigenvalues—one with positive real parts and the other with negative real parts. Such a distribution is hard to deal with. A translation which is capable of decreasing the largest eigenvalue from 1.0 to 0.5 in the inviscid case will not be as effective here. The reason is that the largest and smallest eigenvalues of the viscous transonic case are, respectively, given by, $\lambda_{\max} = 0.99108 \pm i0.049795$ and $\lambda_{\min} = -0.834 \pm i0.0408$, which leaves little room for translation.

V. ACCELERATION METHODS

This section deals with some existing techniques for the acceleration of convergence: It gives a brief picture of *eigenvalue annihilation* applied to the ARC2D. The resulting improvement in the rate of convergence by this method is obtained, but the underlying difficulty of the exact computation of a few large eigenvalues is well known. The *classical relaxation* technique is also discussed. This method works very well for one-dimensional problems, but not in the ARC2D. A simple proof to this effect is also given. Finally, a method based on *shifting the spectrum* of the implicit operators is derived. This method, inspite of its simple implementation, has shown interesting results with considerable savings in computer time. These results are presented here.

Whenever the largest few eigenvalues are discretely distributed, eigenvalue annihilation techniques based on Aitken's or Shanks' transformation [14] will work effectively. Explicit eigenvalue annihilation requires that a linear combination of two recent solutions be constructed as [14],

$$Q=\frac{Q^{n+1}-\lambda Q^{n}}{1-\lambda},$$

where $\lambda=\lambda_{\max}$. This improved solution Q clearly converges according to the second largest eigenvalue and not as $\lambda_{\max}$. This method has been applied to a one-dimensional problem [16] to produce impressive results.

The difficulty appears in two dimensions. For the test case of ARC2D, with a 127×32 C-grid, the solution Q is a column vector of dimension 16,256. In order to annihilate the largest p eigenvalues, the previous $(p+2)$ solutions must be saved. Manipulating them means an extra storage of $(p+2)*16{,}256$, which can easily become a problem on small computers. Moreover, the question arises: *How many of these* 16,256 *eigenvalues need to ne annihilated to improve the convergence of the code*? The answer is: *approximately* 800 *for the viscous transonic case, in improving the rate of convergence from* 0.99 *to* 0.96. Theoretically, this is possible by using an 800-term eigenvalue annihilation algorithm. In practice, this suffers from serious storage considerations on machines like the CRAY XMP-48 supercomputer at NASA Ames Research Center. If computer memory were not a problem, a good estimate of the largest 800 eigenvalues is still out of question, even by using Arnoldi's method. This is because Arnoldi's method does not necessarily compute the "largest" eigenvalues in a certain spectrum, as explained in Ref. [16, 17]. Thus, in general, the use of an eigenvalue annihilation procedure in accelerating the ARC2D is not very effective.

Equation (10) can be rewritten as

$$Q=\frac{1}{1-\lambda}\left[Q^{n+1}-\lambda Q^{n}\right]$$
$$=\omega Q^{n+1}+(1-\omega)\,Q^{n},$$

where $\omega=1/(1-\lambda)$. This is the classical relaxation procedure which is equivalent to explicit eigenvalue annihilation defined by Eq. (10). The parameter ω which lies between 0 and 1 can be varied to achieve optimal results. Relaxation works very well in one-dimensional problems. In the two-dimensional case where the largest eigenvalues are clustered together, it does not perform well. In fact, the following shows why a classical relaxation scheme will not work for the ARC2D:

Consider the standard iteration scheme,

$$x_{n+1}=Ax_n+f,$$

where A is a matrix and f is a vector. Suppose the converged solution x^* satisfies

$$x^*=Ax^*+f.$$

The error $e_n=x_n-x^*$ satisfies

$$e_{n+1}=Ae_n.$$

A new iteration can be formed with relaxation parameter ω, such that,

$$\tilde{x}_{n+1}=\omega x_n+(1-\omega)(Ax_n+f)$$
$$=[(1-\omega)A+\omega I]\,x_n+(1-\omega)f.$$

New errors obey the equation

$$\tilde{e}_{n+1}=[(1-\omega)A+\omega I]\,\tilde{e}_n.$$

If λ' denotes the eigenvalues of the new iteration and λ denotes the eigenvalues corresponding to the original iteration, then they are related by

$$\lambda'=(1-\omega)\lambda+\omega.$$

The rate of convergence originally governed by $|\lambda|$ is now governed by $|\lambda'|$, which will be a minimum if

$$\frac{d}{d\omega}|\lambda'|=0.$$

But

$$\frac{d}{d\omega}|\lambda'|^2=\frac{d}{d\omega}\{(1-\omega)^2|\lambda|^2+\omega^2+\omega(1-\omega)\cdot 2\,\mathrm{Re}(\lambda)\}$$

(where Re means "real part of")

$$=-2(1-\omega)|\lambda|^2+2\omega+(1-2\omega)\cdot 2\,\mathrm{Re}(\lambda)$$
$$=\omega[2+2|\lambda|^2-4\,\mathrm{Re}(\lambda)]+2\,\mathrm{Re}(\lambda)-2|\lambda|^2]$$

This vanishes if

$$\omega=\frac{|\lambda|^2-\mathrm{Re}(\lambda)}{1+|\lambda|^2-2\,\mathrm{Re}(\lambda)}.$$

Consequently, there will be no improvement if $\lambda = 1$, which is almost true in many cases of the ARC2D and where there are a lot of eigenvalues with magnitudes close to 1.0.

In fact, explicit eigenvalue annihilation based on $\lambda_{\max}$ was found to be useful for inviscid cases only. For inviscid, subsonic flow with a free stream velocity of Mach 0.4, $\lambda_{\max} = 0.9265 \pm i0.1981$ was annihilated as a complex conjugate pair. This improved the rate of convergence from 0.959 to 0.952. Based on 300 iterations, this is an improvement of one order of magnitude which is about 50 iterations, or a savings of less than 17%.

A similar improvement was obtained for inviscid flow at Mach 0.8. When the largest eigenvalue, given by $\lambda_{\max} = 0.9705 \pm i0.088$ was annihilated at time step 250, the rate of convergence improved from 0.985 to 0.984, which is again a saving of about 50 iterations.

For the viscous case at Mach 0.4, the same explicit eigenvalue annihilation process was applicable to some extent. Based on a run of 300 iterations, a savings of 50 iterations was always possible. The transonic-viscous problem that motivated this entire study could not be accelerated effectively by the annihilation or relaxation methods. Meager achievements in annihilating the largest eigenvalue led to a savings of 10–15 iterations out of a total of 250. This is definitely not enough.

The failure in using these existing methods led to another class of acceleration methods previously not employed in the context of nonlinear fluid flow problems. This new class of methods, as described in the next sections, is based on shifting the spectrum of the ARC2D operator.

VI. SHIFTING THE SPECTRUM OF THE ARC2D OPERATOR

The *unfactored* block form of the ARC2D algorithm is already given in Eq. (4.1). To build a theory for the *shift* process, an approximation is carried out by neglecting the viscous terms. This results in

$$[I + h\,\partial_\xi A^n + h\,\partial_\eta B^n]\,\Delta Q^n = -h[\partial_\xi E^n + \partial_\eta F^n], \tag{11}$$

where A and B are the flux Jacobians. Homogeneous property of the fluxes can now be exploited ($E = AQ$, $F = BQ$) to simplify Eq. (11). Thus,

$$(I + G^n)\,\Delta Q^n = -G^n Q^n,$$

where $G^n = h\,\partial_\xi A^n + h\,\partial_\eta B^n$. Inversion leads to

$$\begin{aligned} Q^{n+1} &= Q^n - (I+G)^{-1}\,GQ^n \\ &= [I - (I+G)^{-1}\,G]\,Q^n. \end{aligned} \tag{12}$$

This suggests that, in general, the shift procedure and analysis for the ARC2D will be similar to the one-dimensional case, discussed in [16]. The only change to be made is because of the approximate factorization of the implicit operator in Eq. (11). This factorization has already been discussed. Thus if

$$\begin{aligned} &[I + h\,\partial_\xi A^n + h\,\partial_\eta B^n]\,\Delta Q^n \\ &\quad \approx [I + h\,\partial_\xi A^n][I + h\,\partial_\eta B^n]\,\Delta Q^n \end{aligned} \tag{13}$$

a shift "s" can be introduced in each of the ξ and η blocks, so that the factored implicit operator becomes

$$[I + sI + h\,\partial_\xi A^n][I + sI + h\,\partial_\eta B^n].$$

Multiplying out, one obtains

$$\begin{aligned} &I + 2sI + s^2 I + h\,\partial_\xi A^n + h\,\partial_\eta B^n \\ &\qquad + sh\,\partial_\xi A^n + sh\,\partial_\eta B^n + h^2\,\partial_\xi \partial_\eta B^n \\ &\quad = (I + 2sI + h\,\partial_\xi A^n + h\,\partial_\eta B^n) \\ &\qquad + sh(\partial_\xi A^n + \partial_\eta B^n) + s^2 I + h^2\,\partial_\xi \partial_\eta B^n. \end{aligned}$$

Thus a shift s in each factored block, produces a shift $2s$ in the implicit operator. In addition, some unwanted terms, $s^2 I$ and $sh(\partial_\xi A^n + \partial_\eta B^n)$ are also introduced. These terms are small enough to be neglected. Moreover, the term $h^2\,\partial_\xi \partial_\eta B^n$ can be dropped, too, because the time derivatives chosen here are accurate up to the first order.

Therefore, to adopt a shifting procedure for the factored form of ARC2D, a shift $s/2$ must be introduced in each factor. The effect of $s/2$ in each factor will automatically shift the implicit operator by an amount s, where s is taken to be

$$s = -\tfrac{1}{2}(\lambda_{\max} + \lambda_{\min}). \tag{14}$$

Consequently a shift of $s/2$ in each factor of Eq. (13) will reduce the largest eigenvalue from $\lambda_{\max}$ to $\mu_{\max}$, as in [16], where

$$\mu_{\max} = \frac{1 - \frac{1}{2}(\lambda_{\max} + \lambda_{\min})}{1 - \frac{1}{2}(\lambda_{\max} + \lambda_{\min}) + g}. \tag{15}$$

where g is an eigenvalue of G.

TABLE I

Iteration	Residual		Rate of convergence	
	W|out shift	With shift	W/out shift	With shift
100	0.193×10^{-5}	0.695×10^{-6}	0.9771	0.975
200	0.169×10^{-6}	0.677×10^{-8}	0.9568	0.9547
300	0.263×10^{-9}	0.740×10^{-10}	0.9592	0.9558
400	0.759×10^{-11}	0.462×10^{-12}	0.965	0.9505
500	0.352×10^{-12}	10^{-14} at iter. 473	0.9697	—
600	0.182×10^{-13}	—	0.9708	—
625	10^{-14}	—	—	—

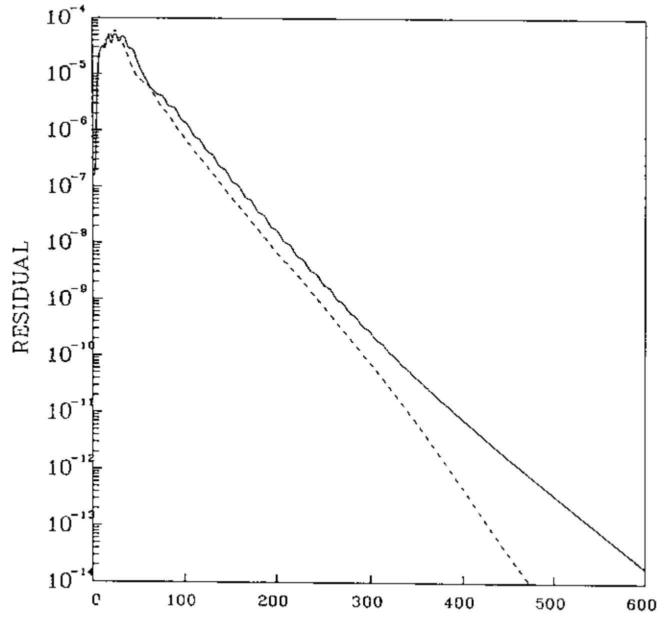

FIG. 11. ARC2D, 127 × 32 C-mesh, Mach 0.4, with and without shift, (without shift residual = 1–.0E – 14 at 625 iterations, ———; shifted each block by −0.2, – – –).

The shift $s/2 = -\frac{1}{4}(\lambda_{\max} + \lambda_{\min})$ was introduced into the implicit factors of Eq. (13) and various flows were computed. In the inviscid, subsonic case with a Mach number of 0.4, $\lambda_{\max}$ and $\lambda_{\min}$ were found to be $0.91326 \pm i0.212$ and -0.102, respectively. Using a shift of $-\frac{1}{4}(\lambda_{\max} + \lambda_{\min})$ which is approximately −0.2 with the imaginary part

TABLE II

Iteration	Residual		Rate of convergence	
	W/out shift	With shift	W/out shift	With shift
100	0.313×10^{-5}	0.214×10^{-5}	0.981	0.9797
200	0.779×10^{-6}	0.284×10^{-6}	0.986	0.9800
300	0.196×10^{-6}	0.489×10^{-7}	0.9861	0.9826
400	0.538×10^{-7}	0.939×10^{-8}	0.9875	0.9836
500	0.153×10^{-7}	0.323×10^{-8}	0.9875	0.9890
600	0.474×10^{-8}	0.200×10^{-8}	0.9881	0.9951
700	0.210×10^{-8}	0.133×10^{-8}	0.9918	0.9943
800	0.148×10^{-8}	0.623×10^{-9}	0.9965	0.9943
900	0.101×10^{-8}	0.340×10^{-9}	0.9962	0.9939
1000	0.653×10^{-9}	0.186×10^{-9}	0.9957	0.9939

supressed, the code was found to converge faster. A comparison of *shifted* and *unshifted* processes is shown in Fig. 11. Based on the interation count, this is a savings of about 150 iterations, or 25%. As already mentioned, explicit eigenvalue annihilation produced a savings of less than 17% for this flow. Table I gives a comparison of the residuals and rates of convergence for both these methods.

For viscous flow at the same Mach number of 0.4 and a Reynold's number of 10,000, based on the chord length, the ends of the spectrum were found to be $\lambda_{\max} = 0.991$ and $\lambda_{\min} = -0.584 \pm i0.04$. With $s/2 = -\frac{1}{4}(\lambda_{\max} + \lambda_{\min}) \approx -0.1$, the performance of the unshifted ARC2D code was compared with the shifted version. The results are plotted in

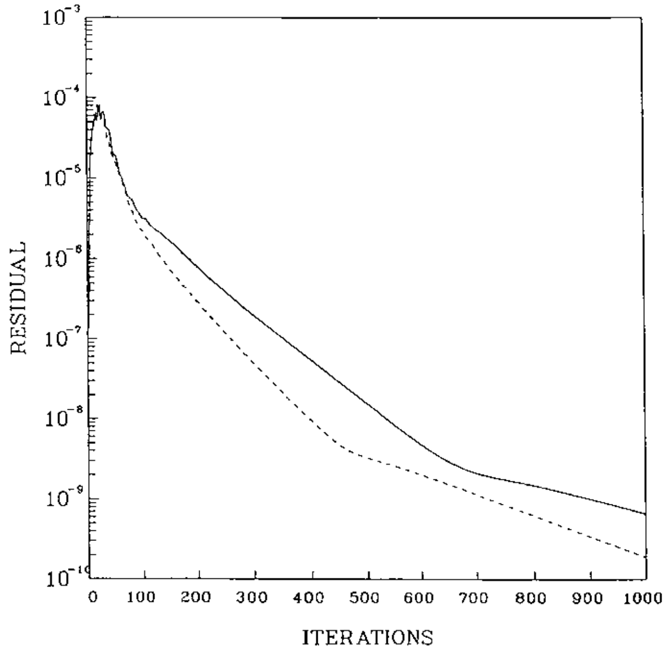

FIG. 12. ARC2D 127 × 32 viscous C-mesh, Re = 10,000, Mach = 0.4 (without shift, ———; each factored block shifted by −0.1, – – –).

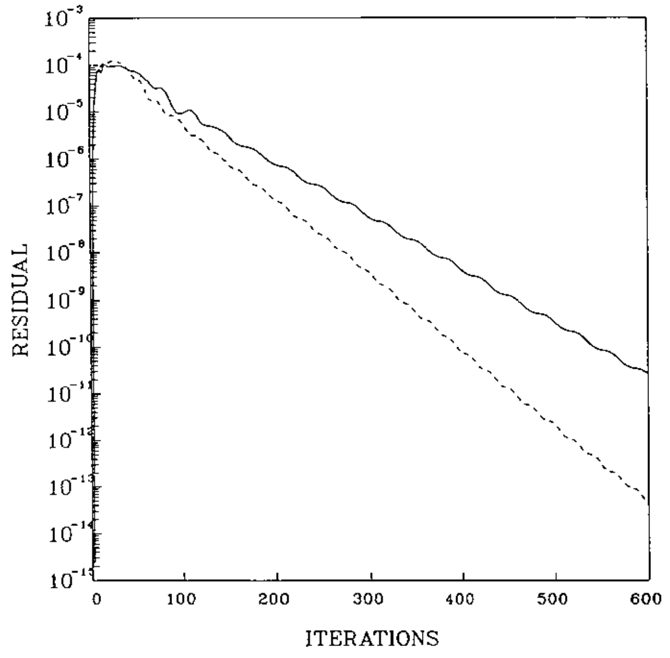

FIG. 13. ARC2D, 127 × 32 inviscid C-mesh, Mach = 0.8, with and without shifting the spectrum (unshifted version, rate of convergence = 0.975, ———, with shift = −0.2, rate of convergence = 0.9617, – – –).

Fig. 12. A savings of about 200 iterations, or 20%, can be observed on a run of 1000 iterations. The residuals and rates of convergence are given in Table II.

In the transonic case (Mach 0.8), inviscid flow calculations were formed by using the ARC2D code and its shifted version. The shift parameter was chosen from the equation

$$\frac{s}{2} = -\frac{1}{4}(-0.9707 \pm i0.088 - 0.07) \approx -0.2.$$

The results shown in Fig. 13 clearly reflect a savings of about 200 iterations, or 33%.

Finally, the shift procedure was applied to a difficult case of transonic flow (Mach 0.85), with an extremely high Reynold's number of eight million. With a 2-degree angle of attack, this was a separated flow with a lot of turbulence. ARC2D uses the Baldwin–Lomax turbulence model [19]. The unshifted version of ARC2D failed to convergence for this case. The residual hung up around 10^{-5} at 200 iterations and stayed the same thereafter. The approximate ends of the spectrum for this flow had been computed at 250 iterations by using Arnoldi's method. They were

$$\lambda_{\max} = 0.9944 \pm i0.0419$$
$$\lambda_{\min} = -0.4.$$

This suggested a shift parameter $s/2 \approx -0.15$. With this shift, the code was run to 1000 iterations. The convergence history in Fig. 14 shows a lot of oscillations in the residual.

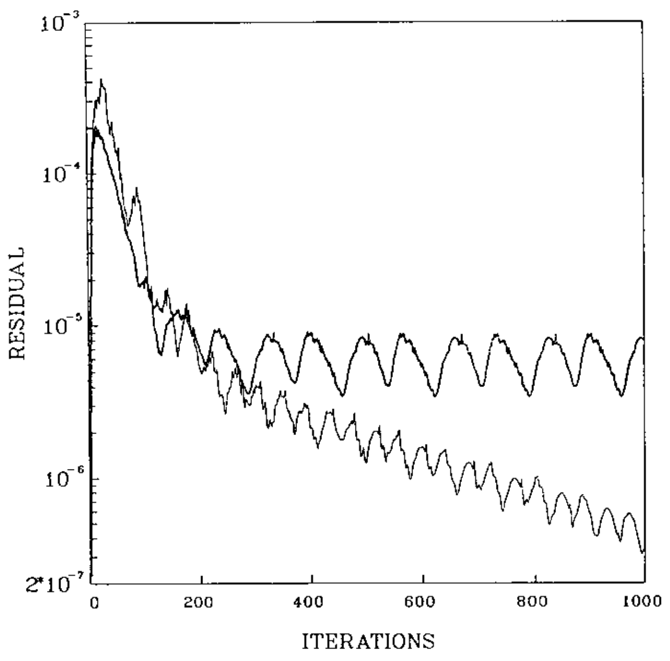

FIG. 14. ARC2D 127 × 32 viscous C-mesh, Mach 0.85, Re = 8E + 06, Alpha = 2°, without shift, ———; shift = −0.3, ———.

This is because of a complex dominant eigenvalue, whose presence was confirmed by Arnoldi's method. With the shifted version, the residual at 300 iterations is smaller than that of the original ARC2D at 1000 iterations. Even though this is an improvement of less than two orders of magnitude, the shifted version saved some 700 iterations. Whereas the coefficient of lift C_L for the NACA0012 airfoil oscillated between 0.165 and 0.225 originally, it converged to three decimal places with the improvement caused by shifting the spectrum of the implicit operator.

VII. SENSITIVITY OF THE SOLUTION W.R.T. SHIFT

The shift parameter is found to depend strongly on the eigenvalue distribution of the operators. This was confirmed by the fact that a different shift was required for each class of problems. Moreover, for different values of "s," the rates of convergence were calculated and plotted. In the case of the ARC2D, only a few values of s were tested each time. This was done inorder to carry out the sensitivity test with maximum economy. The results are shown in Table III. Theoretically, optimal values of the shift for these cases, as given by Eq. (14), were 0.2, −0.15, and −0.2, respectively. These values can be confirmed from the experimental data of Table III.

VII. CONCLUSION

Shifting of the spectrum, presented here, is a simple but powerful idea. It stems from the usual power method [15] of linear algebra. The amount of shift to be employed, depends on the eigenspectrum of the problem in question. This is where the entire difficulty lies, but Arnoldi's method is an economical solution of the problem. Since Arnoldi's method extracts the ends of the spectrum very efficiently and accurately [9], it should be used to compute $\lambda_{\max}$ and $\lambda_{\min}$. In the one-dimensional nozzle problem [16] with 100 grid points, Arnoldi's method gave a good estimate of $\lambda_{\max}$ and $\lambda_{\min}$ by using an additional 20 iterations of the code. Similarly, in the ARC2D with a grid of 127 × 32, where the

TABLE III

Mach 0.4, inviscid at 400 iterations		Mach 0.4, viscous at 1000 iterations		Mach 0.8, inviscid at 600 iterations	
Shift	Rate	Shift	Rate	Shift	Rate
0	0.9651	0	0.99569	0	0.987
−0.2	0.9505	−0.1	0.9945	−0.2	0.981
−0.3	0.9517	−0.15	0.9939	−0.3	0.992
−0.4	1.0	−0.2	1.0	—	—
Does not converge		Does not converge			

implicit operator is a matrix of order 16256, $\lambda_{\max}$ and $\lambda_{\min}$ can be estimated in about 50 extra iterations of the code. In terms of CPU time on the Cray XMP, it costs about 7 s. This is minimal as compared to the amount of savings produced. But the overall reduction in CPU time and the number of iterations to convergence is noticeable. The total gain in CPU time is better than 20 s for a calculation that took 100 CPU s. Arnoldi's method actually pays for itself when used for shifting the spectrum of any linear or nonlinear transformation.

In using Arnoldi's method for computing spectrums of nonlinear operators, the corresponding eigenvectors could always be obtained at no extra cost. These eigenvectors that belonged to a *certain* Krylov subspace formed an orthogonal system under some assumptions. Arnoldi's method gave more information about the operators than was utilized in this paper. A preconditioning of the ARC2D equations by the above-mentioned orthogonal system is still an open question.

ACKNOWLEDGMENTS

The authors are thankful to the referees for suggestions that have improved this paper.

REFERENCES

1. Y. Saad, *Linear Algebra Appl.* **34**, 269 (1980).
2. IMSL, International Mathematical & Statistical Library, Houston, TX.
3. T. H. Pulliam, Efficient solution methods for the Navier–Stokes equations, Lecture Notes for the von Karman Institute for Fluid Dynamics, 1986.
4. T. H. Pulliam, Implicit solution methods in computational fluid dynamics, "Applied Numerical Mathematics," *Trans. IMACS* **2**(6), 441 (1986).
5. T. H. Pulliam, Euler and Thin Layer Navier–Stokes Codes, ARC2D and ARC3D, 1986.
6. A. Cheer, M. Saleem, T. H. Pulliam, and M. Hafez, "Analysis of the Convergence History of Flows through Nozzles with Shocks, in *AIAA (American Institute of Aeronautics and Astronautics), SIAM First National Fluid Dynamics Congress, Cincinnati, Ohio, 1988*, 620 (unpublished).
7. W. E. Arnoldi, *Q. Appl. Math.* **9**, 17 (1951).
8. L. E. Eriksson and A. Rizzi, *J. Comput. Phys.* **57**(1), 263 (1985).
9. J. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Vol. I (Birkhauser, Boston, 1985).
10. T. H. Pulliam, ARC2D Computer Code at NASA Ames Research Center, for Solving Thin Layer Navier–Stokes Equations, 1987 (unpublished).
11. R. Beam and R. F. Warming, *J. Comput. Phys.* **22**, 87 (1976).
12. D. S. Chausse and T. H. Pulliam, A Diagonal Form of an Implicit Approximate Factorization Algorithm with Application to a Two Dimensional Inlet, AIAA (American Institute of Aeronautics and Astronautics, 1981), 83.
13. T. H. Pulliam, AIAA 85-0438, 23rd Aerospace Sciences Meeting, Reno, NV, 1985.
14. D. Shanks, *J. Math. Phys.* **34**, 1 (1955).
15. D. K. Faddeev and V. N. Faddeva, *Computational Methods of Linear Algebra*, translated by R. C. Wiliams (Freeman, San Francisco, 1963).
16. A. Cheer and M. Saleem, *Int. J. Numer. Methods Fluids* **12**, 443 (1990).
17. M. Saleem, Ph.D. Dissertation, University of California, Davis, 1988 (unpublished).
18. E. K. Blum, *Numerical Analysis and Computation. Theory & Practice* (Addison–Wesley, Reading, MA, 1972).
19. B. S. Baldwin and H. Lomax, AIAA Paper No. 78-257, 1978.
20. A. Jameson, *Appl. Math. Comput.* **13**, 327 (1983).