JOURNAL OF COMPUTATIONAL PHYSICS 110, 68–81 (1994)

Preconditioned Conjugate Gradient Methods for the Navier–Stokes Equations

KUMUD AJMANI AND WING-FAI NG

Department of Mechanical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061

AND

MENG-SING LIOU

Internal Fluid Mechanics Division, NASA Lewis Research Center, Cleveland, Ohio 44135

Received February 3, 1992; revised April 20, 1993

A preconditioned Krylov subspace method (GMRES) is used to solve the linear systems of equations formed at each time-integration step of the unsteady, two-dimensional, compressible Navier–Stokes equations of fluid flow. The Navier–Stokes equations are cast in an implicit, upwind finite-volume, flux-split formulation. Several preconditioning techniques are investigated to enhance the efficiency and convergence rate of the implicit solver based on the GMRES algorithm. The superiority of the new solver is established by comparisons with a conventional implicit solver, namely line Gauss–Seidel relaxation (LGSR). Computational test results for low-speed (incompressible flow over a backward-facing step at Mach 0.1), transonic flow (trailing edge flow in a transonic turbine cascade), and hypersonic flow (shock-on-shock interactions on a cylindrical leading edge at Mach 6.0) are presented. For the Mach 0.1 case, overall speedup factors of up to 17 (in terms of time-steps) and 15 (in terms of CPU time on a CRAY-YMP/8) are found in favor of the preconditioned GMRES solver, when compared with the LGSR solver. The corresponding speedup factors for the transonic flow case are 17 and 23, respectively. The hypersonic flow case shows slightly lower speedup factors of 9 and 13, respectively. The study of preconditioners conducted in this research reveals that a new LUSGS-type preconditioner is much more efficient than a conventional incomplete LU-type preconditioner.

1. INTRODUCTION

Recent advances in CFD techniques and computer technologies have led to the development of very powerful tools for the simulation and analysis of fluid flow over complex configurations. The 2D, compressible, Navier–Stokes equations are integrated in time by the application of an implicit, finite-volume scheme. The difference equations at each grid point lead to a system of simultaneous linear equations for the entire domain, which has to be solved at each time step.

The system of equations can be solved by direct matrix inversion; however, this requires extensive computer memory and computation effort at each time step. Iterative schemes are a viable alternative because of their computational efficiency and relatively meager memory requirements. These schemes, however, can be slow in converging from the initial guess to the final solution—a problem which has been a topic of research over the years for numerical analysts.

Upwind relaxation methods [1–3] and the spatially split approximate factorization [4–6] (AF) techniques have enjoyed considerable popularity in CFD research over the years. Relaxation schemes are related to the spatial discretization of the convective and diffusive terms. For first-order upwind differencing, the coefficient matrix is diagonally dominant for any time-step, which permits the use of standard relaxation schemes like Jacobi and Gauss–Seidel iteration. Higher-order spatial differencing does not guarantee diagonal dominance of the coefficient matrix, which forces the use of some form of alternate sweeping strategy to ensure stability. Hence, the convergence rate of relaxation schemes strongly depends on the spatial discretization adopted for the implicit operator. In addition, many of the relaxation schemes do not completely vectorize due to recursive operations. The schemes that do vectorize completely (e.g., point Jacobi) have poor rates of convergence to the steady state.

The AF techniques refer to the family of methods where the 2D or 3D implicit operator is factored into a sequence of simpler 1D operators. The convergence properties of these schemes are strongly affected by the factorization error, particularly as the time-step becomes large (optimal convergence is obtained for Courant numbers of order 10). Hence, the AF techniques exhibit an optimal convergence rate that is time-step dependent and problem dependent [2]. In 3D applications, the AF techniques encounter stability restrictions on the time-step. The advantages of AF

0021-9991/94 $6.00

where

$$Q = [\rho, \rho u, \rho v, \rho e_0]^T; \qquad J = \xi_x \eta_y - \xi_y \eta_x$$

$$\hat{F} = \frac{\xi_x F}{J} + \frac{\xi_y G}{J}; \qquad \hat{G} = \frac{\eta_x F}{J} + \frac{\eta_y G}{J}$$

$$\hat{G}_v = \left(\frac{\mu}{\mathrm{Re}_L}\right) [\hat{g}_{v_1}, \hat{g}_{v_2}, \hat{g}_{v_3}, \hat{g}_{v_4}]^T,$$

where ξ_x, ξ_y, η_x, η_y are the metrics of the transformation. In addition,

$$F = F(Q) = [\rho u, \rho u^2 + p, \rho uv, (\rho e_0 + p) u]^T$$

$$G = G(Q) = [\rho v, \rho uv, \rho v^2 + p, (\rho e_0 + p) v]^T$$

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

$$\hat{g}_{v_1} = 0, \qquad \hat{g}_{v_2} = \alpha_1 u_\eta + \alpha_3 v_\eta, \qquad \hat{g}_{v_3} = \alpha_3 u_\eta + \alpha_2 v_\eta$$

$$\hat{g}_{v_4} = \frac{\alpha_1}{2}(u^2)_\eta + \frac{\alpha_2}{2}(v^2)_\eta + \alpha_2 (uv)_\eta + \frac{(a^2)_\eta}{\mathrm{Pr}(\gamma - 1)}$$

$$\alpha_1 = 1 + \frac{1}{3}\frac{\eta_x^2}{J}, \qquad \alpha_2 = 1 + \frac{1}{3}\frac{\eta_y^2}{J}, \qquad \alpha_3 = \frac{1}{3}\frac{\eta_x \eta_y}{J}.$$

The specific heat ratio, γ, is taken to be 1.4. The molecular viscosity is given by μ, a is the speed of sound, and Re_L is the Reynolds number per unit length. Nondimensionalization is with respect to the freestream density and velocity. The physical coordinates (x, y) and viscosity are nondimensionalized by a reference length L and the molecular viscosity of the freestream, respectively.

The governing equations are solved computationally in their integral, conservation law form, using a cell-centered finite volume formulation. Inviscid flux terms are upwinded using Van Leer's [13] flux-splitting scheme. The thin-layer viscous fluxes are evaluated with second-order accurate central differences.

Equation (2) can be rewritten as

$$\frac{1}{J}\frac{\partial Q}{\partial t} = -R, \tag{3}$$

where

$$R = R(Q) = \frac{\partial \hat{F}}{\partial \xi} + \frac{\partial \hat{G}}{\partial \eta} - \frac{\partial \hat{G}_v}{\partial \eta}.$$

R is called the residual and is equal to zero for a steady state solution. The Euler implicit discretization of Eq. (3) in time gives

$$\frac{\Delta Q^n}{J\,\Delta t} = -R^{n+1}, \tag{4}$$

where ΔQ^n is the incremental change in the cell-centered values of the vector Q between the $(n+1)$th time level and the known nth time level, i.e.,

$$\Delta Q^n = Q^{n+1} - Q^n. \tag{5}$$

R^{n+1} is linearized in time about the nth time level which transforms Eq. (4) into

$$\left(\frac{I}{J\,\Delta t} + \frac{\partial R^n}{\partial Q}\right)\Delta Q^n = -R^n \tag{6}$$

or

$$V^n\,\Delta Q^n = -R^n, \tag{7}$$

where $I/J\,\Delta t$ is a block-diagonal matrix and $\partial R^n/\partial Q$ is a large, sparse, block, banded matrix.

The system of linear simultaneous algebraic equations of Eq. (7) can be solved in several efficient ways. In this research, a comparison is made between the efficiency of an alternate sweeping line Gauss–Seidel relaxation (LGSR) solver [3] and that of a preconditioned generalized conjugate-gradient type solver. Results of comparisons with an approximate factorization (AF) solver may be found in Ref. [14].

2.2. *Conjugate Gradient Methods (CGMs)*

Recall, that we are interested in solving the linear system of equations

$$V^n\,\Delta Q^n = -R^n \Leftrightarrow Ax = b. \tag{8}$$

The classical conjugate gradient method was proposed by Hestenes and Steifel [8] to solve the system $Ax = b$, where A is a symmetric and positive definite (SPD) matrix. The idea of using CGMs as iterative methods was first discussed by Reid [15].

If the coefficient matrix is not symmetric and/or positive-definite (which is often the case in CFD applications), the method requires generalization. Several generalizations have been proposed, particularly for non-symmetric systems by Saad and Schultz [11], Young and Jea [16], and Axelsson [17], to name a few. The CGS method of Sonneveld [18], the Bi-CGSTAB method of Van der Vorst [19], and the QMR method of Freund and Nachtigal [20] are some recent additions to the family of generalizations for CGMs. There is no definitive way of determining the best generalization. The particular generalization used in this paper is the generalized minimal residual (GMRES) method of Saad and Schultz [11], which seems to be the most popular method in the literature. The fundamental idea of GMRES is to minimize the norm of the computed residual vector, r_n ($r_n \equiv b - Ax_n$) at each iteration.

The GMRES method is formulated in such a way that it is directly applicable to solve linear systems with nonsymmetric coefficient matrices. The first step in the method is to use Arnoldi's method [21] to construct an orthonormal basis of vectors of the Krylov subspace $K(A, w_1, k) \equiv \text{span}\{w_1, Aw_1, ..., A^{k-1}w_1\}$. Let W_k denote the rectangular matrix of size $N * k$, whose columns consist of the k individual vectors (each of length N) of the orthonormal basis of vectors formed by the Arnoldi process.

In order to solve the linear system $Ax = b$, the method seeks an approximate solution x_k of the form $x_k = x_0 + z_k$, where x_0 is some arbitrary initial guess to the exact solution $\bar{x}$. The vector z_k lies in the Krylov subspace collectively defined by A, $w_1 = r_0/\beta$ ($\beta = \|r_0\|$) and k, i.e.,

$$z_k \in K(A, r_0, k) = \text{span}\{r_0, Ar_0, A^2r_0, ..., A^{k-1}r_0\} = \text{span}\{r_0, r_1, r_2, ..., r_k\} = W_k y_k \quad (9)$$

for any vector y_k. The various methods generated by this approach are called Krylov subspace methods.

The iterative scheme based on Krylov subspace methods will converge the fastest when each successive iterate x_k (or y_k) minimizes the residual norm, $\|r_k\|$. This can be restated as a minimization of $\|r_k\|$ over $z_k \in K(A, r_0, k)$, i.e.,

$$\min_{z_k \in K(A, r_0, k)} \|r_k\| \Leftrightarrow \min_{z_k \in K(A, r_0, k)} \|(b - Ax_k)\|. \quad (10)$$

The function to be minimized can be rewritten as

$$\begin{aligned} \|(b - Ax_k)\| &= \|(b - Ax_0 - Az_k)\| = \|(r_0 - Az_k)\| \\ &= \|(\beta w_1 - AW_k y_k)\| \\ &= \|(W_{k+1}\beta e_1 - W_{k+1}H_k y_k)\| \\ &= \|(\beta e_1 - H_k y_k)\| = \phi(y_k), \end{aligned} \quad (11)$$

where H_k is an upper-Hessenberg matrix defined such that $AW_k = W_{k+1}H_k$ and W_{k+1} is an orthonormal matrix. A QR factorization of H_k is performed ($Q_k H_k = R_k$), and the upper-triangular matrix R_k is used solve a system of the type $R_k y_k = g_k$, to yield the minimization solution y_k. The solution to the linear system is finally computed as $x_k = x_0 + W_k y_k$.

The complete GMRES algorithm can be written as follows:

1. For any starting vector x_0, from $r_0 = b - Ax_0$; $\beta = \|r_0\|_2$; $w_1 = r_0/\beta$.

2. Perform k steps of Arnoldi's method with w_1 to form W_k.

3. From the approximate solution:

(a) Find the vector y_k which solves the minimization of Eq. (11)

(b) Compute $x_k = x_0 + W_k y_k$.

In summary, the GMRES method is a minimization process to solve linear matrix systems like $Ax = b$. The minimization proceeds as a sequence of sub-iterations, k, and the minimizer is obtained by a simple upper-triangular solve in step 3 of the algorithm. One major practical difficulty with GMRES is that when k increases, both storage and operation cost increase as $O(k)$ and $O(k^2)$, respectively. If the available storage is limited, the method may be restarted after k sub-iterations, with x_k replacing x_0 in step 1. The restart version of the algorithm is referred to as GMRES(k). In addition, the total number of sub-iterations can be restricted to minimize the cost of each global iteration. This issue is discussed in detail later in this paper. The GMRES algorithm is used in conjunction with preconditioning, which is discussed in the next section.

Further mathematical details, implementation techniques, and convergence analyses are contained in Refs. [12, 14]. Since this paper is concerned with convergence acceleration, it may be remarked that the speed of convergence of any algorithm depends on the condition number, $\kappa_2(A)$, of the coefficient matrix A and the distribution of singular values of A. $\kappa_2(A)$ is defined as the ratio of the maximum to minimum singular value of the matrix A. If $\kappa_2(A)$ is large and/or the spectrum of singular values of A is wide and scattered, the matrix A is said to be poorly conditioned and convergence may be very slow. Preconditioning is employed to improve the conditioning of the matrix. The preconditioning technique chosen usually has a first-order effect on the convergence characteristics and efficiency of solvers based on GMRES-like algorithms.

2.3. *Preconditioning—Concepts and Techniques*

One of the most effective iterative methods for solving large sparse linear systems is a combination of a generalized conjugate gradient-like procedure with some appropriate preconditioning technique. Assuming that a preconditioner M is used on the left of the original unpreconditioned system, this involves solving the preconditioned linear system

$$M^{-1}Ax = M^{-1}b \Leftrightarrow \tilde{A}x = \tilde{b}, \quad (12)$$

instead of the original system $Ax = b$.

The motivation of preconditioning is to reduce the overall computational effort required to solve linear systems of equations by increasing the convergence rate of the underlying iterative algorithm. Preconditioning will be effective only if the additional computational work incurred per sub-iteration is compensated for by a reduction in the total number of sub-iterations to convergence—so that the total cost of solving the linear system is reduced.

The costs associated with preconditioning can be enumerated as (i) computing the preconditioning matrix M, (ii) matrix–vector multiplies or equivalent linear system

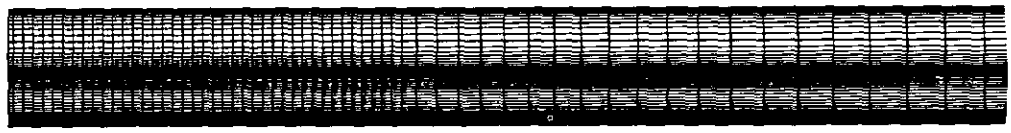

FIG. 1. Computational grid for backward facing step, 61 × 51 grid.

influenced by the skill of the individual programmer, compiler optimization, vectorization, and parallelization. It may be remarked that the "number of time-steps" is independent of all the factors that influence the "CPU time" and thus reflects the true convergence characteristics of a solver. The "CPU time" comparisons are useful and, in conjunction with the "number of time-steps" comparisons, provide an estimate of the practical utility of a solver.

A thorough mathematical analysis of the competing solvers in this paper could provide an in-depth explanation of why one solver or preconditioner is better than another. Such an analysis would require a detailed eigenvalue analysis of each iterative scheme. Unfortunately, it is practical to do such an analysis only for small matrices, as may be encountered in 1D problems or in 2D problems with very coarse meshes. A coarse mesh for the test problems of this paper could be adopted for this purpose, but then there is no theory to provide a one-on-one correspondence between eigenvalues for a coarse and fine mesh. Thus, even though a complete eigenvalue analysis is desirable to fully explain the difference in convergence rates of the solvers, it is impractical for the large problems being tested in this research.

3.1. *Low-Speed Flow over a Backward-Facing Step*

Armaly *et al.* [25] have presented detailed measurements of velocity distribution and reattachment length for the incompressible flow of air downstream of a single backward facing step in a 2D channel. The results show that the various flow regimes (in the Reynolds number range of $70 < \mathrm{Re} < 8000$), are characterized by typical variations of separation length with Reynolds number. The Reynolds number is based on the height of the step and two-thirds of the maximum inflow velocity at the step. The particular test case chosen for this research corresponds to $\mathrm{Re} \approx 400$, since the experimental data suggests that $\mathrm{Re} > 400$ produces 3D variations in the flowfield.

The numerical computations are performed on an H-mesh with 61 and 51 points in the ξ (streamwise) and η (normal) directions, respectively. The grid is shown in Fig. 1. Grid points are clustered, both in the normal and streamwise directions, to resolve the various viscous gradients and the reattachment point of the separated flow. A freestream Mach number of $M_\infty = 0.1$ and Reynolds number of $\mathrm{Re}_\infty = 389$ is specified.

Adiabatic, no-slip boundary conditions are used on the top and bottom walls forming the boundaries of the channel and on the lower portion (which defines the step) of the inflow boundary. For fully developed subsonic flow at the outflow boundary, three variables (ρ, u, and v) are extrapolated and one variable (stagnation enthalpy) is held constant. A parabolic velocity profile at the inflow is simulated by imposing a profile of Reimann invariants at the inflow boundary.

Figure 2 shows Mach number contours obtained from the computations on the 61 * 51 grid. The nature and size of the separation and recirculation behind the step closely matches the physical description of the flow as obtained in the experiments of Armaly *et al.* [25]. It should be remarked that the reattachment point is the primary flow feature used to characterize the flow in the experimental data [25]. For $\mathrm{Re} = 387$, the experimental results suggest a downstream reattachment length (X_R) to step height (S) ratio of $X_R/S = 7.9$. The numerical computation predicts $X_R/S = 8.0$. Thus, the experimental and numerical reattachment lengths are close to each other. A comparison of experimental and computational velocity vectors is shown in Fig. 3.

Figure 4 presents the convergence history comparisons between GMRES with LUSGS preconditioning (GMLUS), GMRES with ILU preconditioning (GMILU), and the line Gauss–Seidel relaxation (LGSR) solver. The "logarithm of the l_2 norm of the residual" has been plotted against the "number of global time-steps." It can be clearly seen that both GMLUS and GMILU converge at a much faster rate than the LGSR solver. The maximum Courant numbers (λ) for GMLUS, GMILU, and LGSR are 100, 100, and 10, respectively. The maximum λ for LGSR is the value of λ above which LGSR becomes unstable. The maximum λ for

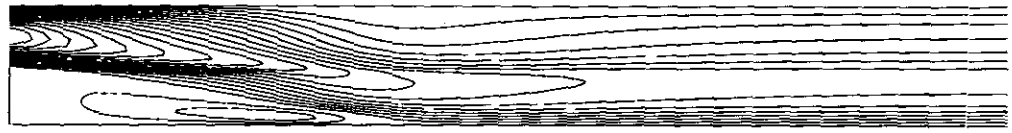

FIG. 2. Mach number contours for backward facing step, range = 0.0, 0.1, interval = 0.003.

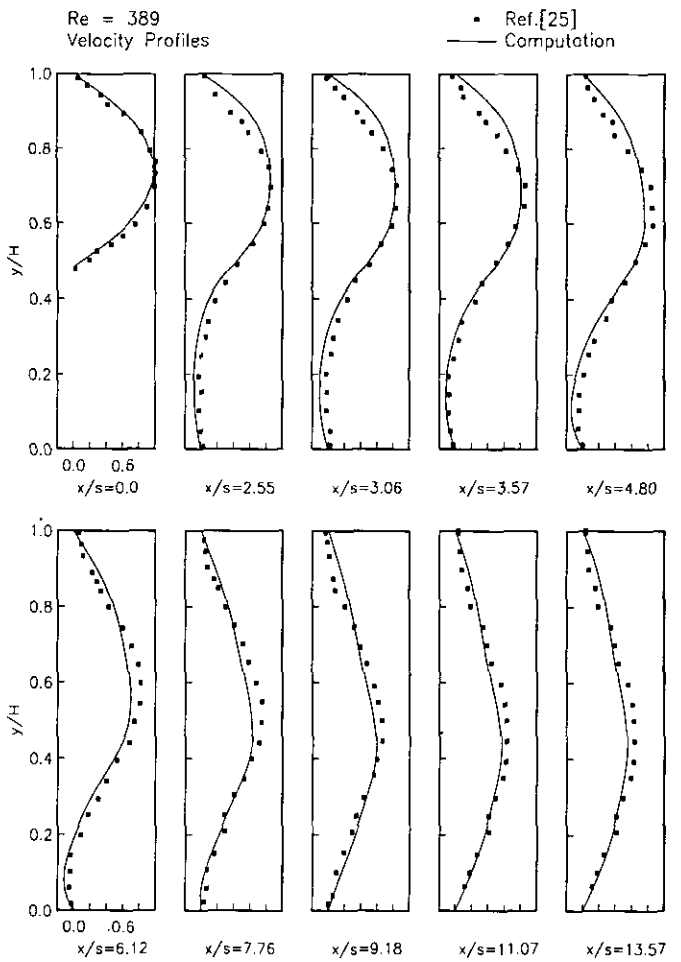

FIG. 3. Velocity vectors for backward facing step.

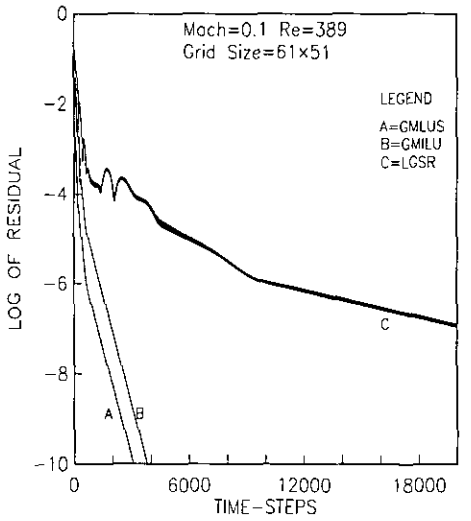

FIG. 4. "Number of time-steps" comparison for backward facing step.

GMLUS and GMILU is uniformly selected to be 100 for all the test cases in this research. This is consistent with the aim of comparing the best available LGSR performance with the GMLUS and GMILU solvers.

The use of different values of λ for the preconditioned GMRES and LGSR solvers may raise some question about the "fairness" of the convergence rate comparisons, particularly since comparatively higher values of λ are used with preconditioned GMRES. The superior convergence rate of the preconditioned GMRES solver can be attributed to two factors—the higher "allowed" values of λ and the use of preconditioning. The role of preconditioning in accelerating convergence can be determined by using preconditioned GMRES at the same λ as the LGSR solver—the result is that the preconditioned GMRES continues to have a superior convergence rate than the LGSR solver, although the speedup factor is reduced.

It must be remarked that the use of preconditioning enables the GMRES solver to accept higher values of λ than those "allowed" by the LGSR solver. It is hence only prudent to take advantage of the increased stability afforded by the use of preconditioning, and increase the convergence rate of the preconditioned GMRES solver by using large Courant numbers. The maximum speedup (in terms of total time-steps to steady state convergence) would thus be obtained by using the maximum allowable values of λ. The GMLUS and GMILU solvers are found to be "stable" at values of $\lambda > 100$. However, a higher λ usually increases the cost per time-step, without being offset by a decrease (in some cases) in the number of time-steps to convergence. Thus, the GMLUS and GMILU solvers perform must efficiently (in terms of CPU time) with Courant numbers of order 100.

The correct physical solution (i.e., the proper reattachment point) is obtained after a six-order reduction of the residual is achieved. The rapid reduction in the initial residual (from zero to three orders) required the use of Reimann invariants at the inflow boundary (as discussed earlier). This rapid reduction represents elimination of part of the initial transient. Curves A (for GMLUS) and B (for GMILU) are generated with a fixed Courant number (λ) of 100, for all time-steps. The LGSR solver (curve C) permits an initial Courant number of $\lambda = 10$ and does not permit any increase in λ (i.e., the iterates diverge at higher values of λ), at any phase of the time integration. GMLUS and GMILU converge in 3806 and 3087 time-steps, with 11,706 and 9720 sub-iterations, respectively. Note that the number of sub-iterations varies at each time-step according to the stopping criteria of Eq. (13) (even though the λ is fixed). LGSR achieves a six-order reduction (i.e., the correct physical solution) in 10,400 time-steps and is estimated to require 51,258 time-steps (according to convergence rate estimates) to attain a 10-order reduction (i.e., convergence). Hence, in terms of number of global time-steps to convergence,

GMLUS and GMILU are 13.5 and 16.5 times faster than the LGSR solver.

The CPU time comparison of the solvers is shown in Fig. 5. The superior efficiency of GMLUS over GMILU and LGSR is seen in this comparison. ILU requires (partially vectorized) computation of the "L" and "U" factors at each global iteration (set-up cost), and subsequent forward and backward (scalar) solves with the "L" and "U" factors at each sub-iteration. The block-LUSGS preconditioner requires virtually no setup time, and involves two (partially vectorized) block-diagonal inversions across the domain at each sub-iteration [14]. This major difference in computational overhead is the reason why GMILU (2173 s) requires five times as much CPU time as GMLUS (440 s). The LGSR solver is again extremely slow in terms of total CPU time, as it was in terms of number of time-steps. LGSR is estimated to require about 6766 s of CPU time for convergence. Thus, LGSR requires 15.4 times more CPU time than GMLUS and 3.1 times more CPU time than GMILU.

One of the goals of this research is to develop effective and efficient preconditioning for the GMRES algorithm for use with the Navier–Stokes equations. The results of Figs. 4 and 5 contribute to this development. The "number of time-steps" comparison shows that both the LUSGS and ILU preconditioners are equally effective in converging to the steady state solution. The CPU time comparison shows that the use of LUSGS preconditioning can afford considerable gains in CPU time over the use of the ILU preconditioner, while maintaining a competitive convergence rate. This is an important step in the identification of the LUSGS scheme as an efficient preconditioner (in terms of storage and CPU time).

The above preconditioner comparison reveals a significant result—either of the two tested preconditioners can be used with GMRES, to obtain a better convergence rate than the conventional LGSR solver. Even though the overhead cost for one preconditioner (ILU) may be more than that for another (LUSGS), the use of either one with the unpreconditioned GMRES solver guarantees better performance than the LGSR solver. Hence, for initial implementations of preconditioned GMRES in new or existing codes, either of the two preconditioners tested in this paper can be employed.

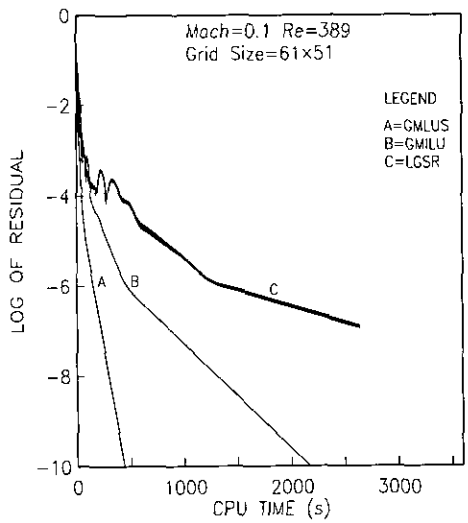

FIG. 5. "CPU time" comparison for backward facing step.

3.2. *Trailing Edge Flow in a Transonic Cascade*

In order to fully understand the flow physics near the trailing-edge of a transonic turbine cascade, Sieverding *et al.* [27] conducted experimental tests on a model simulating the flow in the overhang section of convergent turbine cascades. The experimental setup is shown in Fig. 6. A primary goal of the experiment was to study the flow physics around and behind the blunt trailing edge of the cascade (as modeled by the trailing edge of the flat plate). Correspondingly, the computational results presented here concentrate on resolution of the flow in the trailing edge region. Of the various experimental configurations tested in Ref. [27], the one chosen for numerical simulation in this work corresponds to a flat plate overhang length (l_s) of 37 mm and maximum inclination of the tailboard (simulating complete loading of the cascade). It must be mentioned that the experiment of Ref. [27] was performed with fully turbulent flow. However, the numerical simulation of this work is restricted to laminar flow only and hence detailed comparisons of surface pressure etc. are not presented here. The Reynolds number of the flow is 3×10^7, based on the unit length of the flat plate.

The computational results are obtained on a C-mesh with 207 and 51 points in the ξ and η directions, respectively. The mesh is generated with the GRAPE code [31], and it is shown in Fig. 7. Grid points are clustered around solid surfaces (i.e., the flat plate and the outer walls of the experimental setup) and particularly concentrated around the trailing edge of the flat plate. A freestream Mach number of 0.6 is specified.

Adiabatic, no-slip boundary conditions are specified for all solid surfaces. The inflow boundary (consisting of the

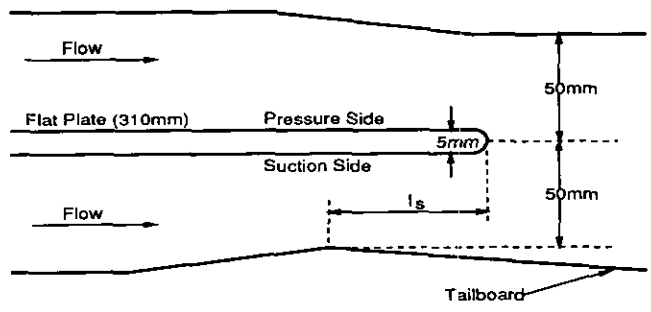

FIG. 6. Experimental setup for transonic cascade [25].

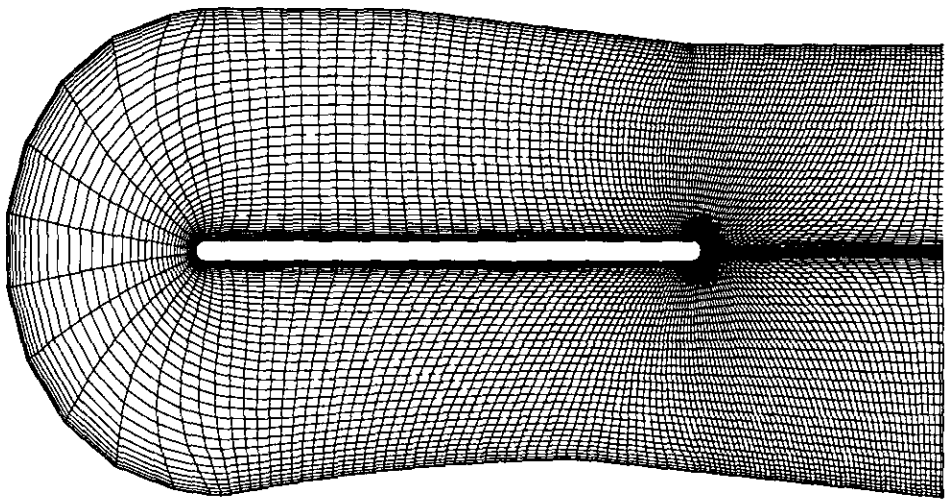

FIG. 7. Computational grid for transonic cascade, 207 × 51 grid.

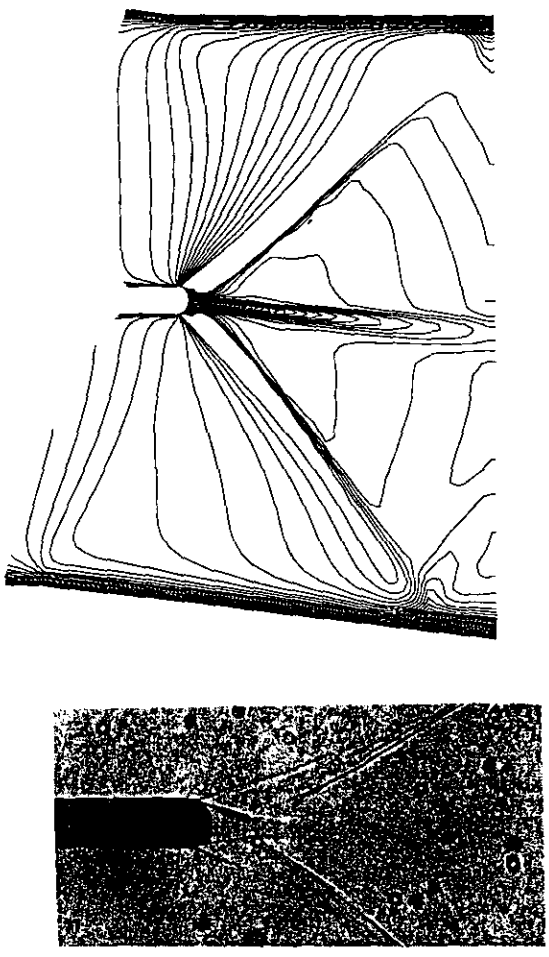

FIG. 8. Mach number contours for transonic cascade, range = 0.0, 1.5, 0.05.

cells forming the semi-circular part of the grid) is specified with subsonic inflow conditions ($v_b = 0$, fixed entropy and stagnation enthalpy, $u_b = u_i$). Standard periodic boundary conditions for C-meshes are applied across the centerline of the wake cut. At the outflow boundary, a back-pressure, $P_b = 0.4P_\infty$ (freestream) is specified to overcome the initial transient. This boundary condition is then changed to enforce extrapolation of all flow variables to the outflow boundary. This change is necessary to maintain stability of the overall solution algorithm.

Figure 8 shows a comparison of Mach number contours generated from computational results and an experimental shadowgraph for the shock system at the trailing edge of the flat plate. The computed trailing edge shocks and expansion waves, along with the location and inclination of the wake centerline, compare very well with the experimental data.

Figure 9 details the convergence characteristics of the GMLUS, GMILU, and LGSR solvers. For this test case, the correct physical solution is obtained after a three-order residual reduction. The Courant numbers are then increase to 100, 100, and 5 for GMLUS, GMILU, and LGSR, respectively. GMLUS and GMILU converge to the steady state solution in 2065 and 2075 time-steps, with 3977 and 3672 sub-iterations, respectively. LGSR is estimated to converge (on the basis of its asymptotic convergence rate) in 34,255 global time-steps. Thus, LGSR requires 16.6 times more time-steps than GMRES (with either preconditioner).

Figure 10 presents the CPU time comparison of the GMLUS, GMILU, and LGSR solvers. The tremendous gains in using preconditioned GMRES instead of LGSR are again clearly evident from this comparison. As observed in the previous test case, ILU preconditioning proves to be much more expensive than LUSGS preconditioning. However, both preconditioners compete will with LGSR,

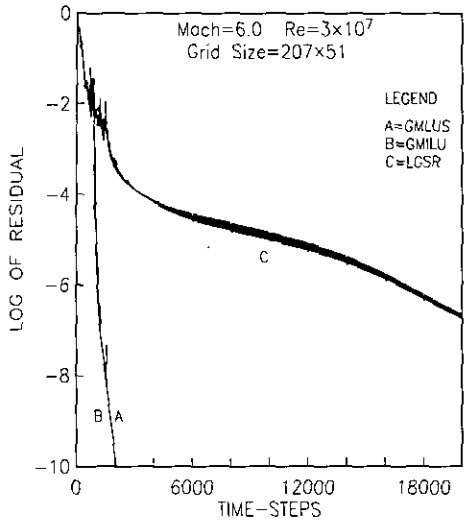

FIG. 9. "Number of time-steps" comparison for transonic cascade.

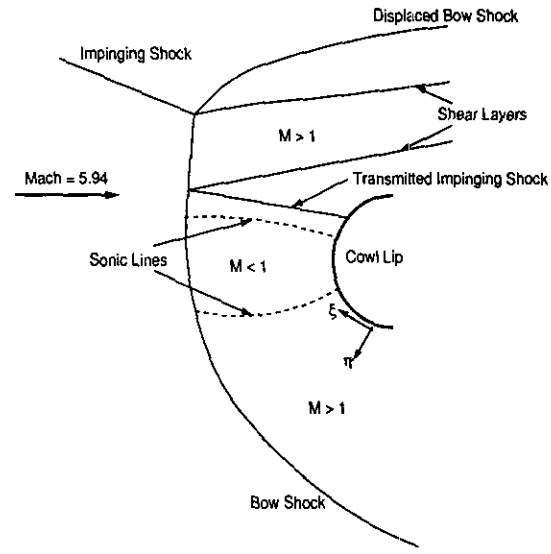

FIG. 11. Shock-on-shock impingement defined by Edney [32] (not to scale).

which is estimated to require 15,072 s of CPU time to converge. Thus, in terms of CPU time to convergence, GMLUS (653 s) and GMILU (4282 s) are roughly 23.1 and 3.5 times faster than LGSR, respectively.

3.3. *Shock-on-Shock Impingement in Hypersonic Flow*

Extensive studies of shock-on-shock impingement and its effect on heat transfer rates and related phenomena on a cylindrical leading edge were conducted by Wieting [28]. The study was motivated by a need to gain insights into the flow physics of (ramp) shocks impinging on a two-dimensional viscous inlet cowl. The flowfield has complex shock–shock and shock–shear layer interactions and is difficult to simulate numerically [29].

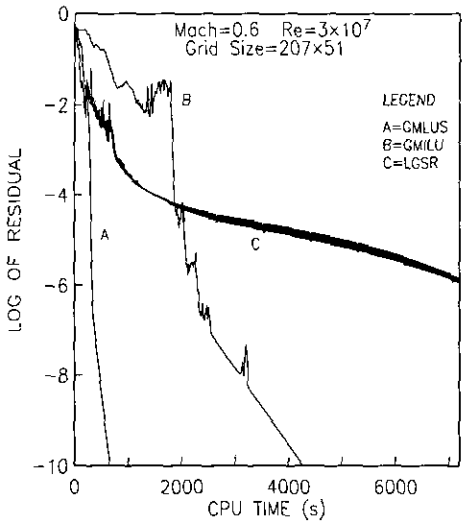

FIG. 10. "CPU time" comparison for transonic cascade.

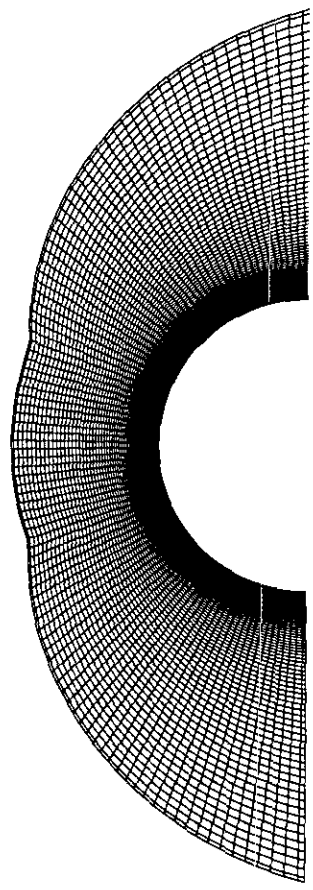

FIG. 12. Computational grid for hypersonic shock (shock-on-shock impingement, 101 × 101 grid).

Edney [32] has classified shock–shock interactions into six categories, based on the interference nature of the impinging shock with the main bow shock. For the purpose of this research, the "Type II" interference pattern of Edney [32] has been chosen for numerical simulation. A schematic of this pattern is shown in Fig. 11. The pattern is created when the impinging shock interacts with the bow shock at a shock angle of 15°. The flow is characterized by a displaced bow shock, a transmitted shock, and formation of supersonic shear layers. The Reynolds number is 1.8×10^5 per unit diameter of the cowl lip.

Computations are performed on an H-mesh with 101 points each in the ξ and η directions. The mesh is shown in Fig. 12. Grid points are clustered near the cowl surface, to resolve the viscous boundary layer. A freestream Mach number of 5.94 is specified for the computation. Since it is difficult to establish the exact location of the impinging shock from the experiment [33], trial-and-error is used until the desired interference pattern for a "Type II" interaction is established.

The incoming oblique shock at the inflow boundary is simulated by imposing the appropriate jump conditions (with respect to freestream values) calculated from inviscid shock theory. The two outflow boundaries are treated with simple extrapolation of all variables to the respective outflow boundaries. Adiabatic, no-slip boundary conditions are imposed on the surface of the cowl.

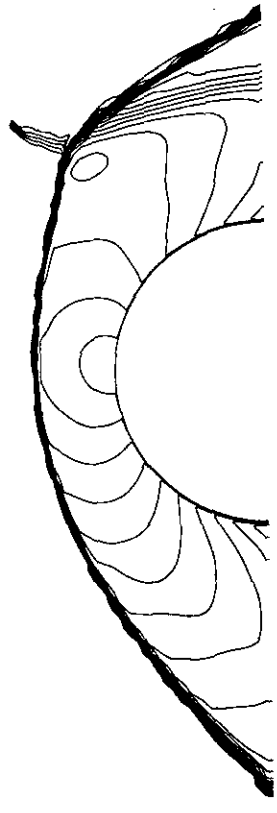

FIG. 13. Mach number contours for hypersonic shock, range = 0.0, 6.0, 0.2.

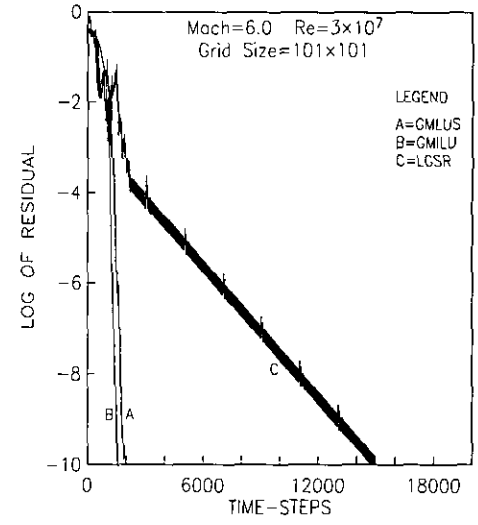

FIG. 14. "Number of time-steps" comparison for hypersonic shock.

The Mach number contours obtained from the computations are shown in Fig. 13. The shear layers and displacement of the bow shock are captured well by the computation. The results compare excellently with similar calculations done by Kloptex and Yee [33]. The transmitted impinging shock (as seen in the schematic of Fig. 11), which is very weak, is not captured. Extensive local grid refinement and grid adaptation is probably required to capture this weak shock.

Figure 14 presents the "number of time-steps" comparison of the GMLUS, GMILU, and LGSR solvers. The "correct" physical solution is established after a four-order reduction in the residual. After the initial transients are resolved, the Courant numbers for the GMLUS, GMILU,

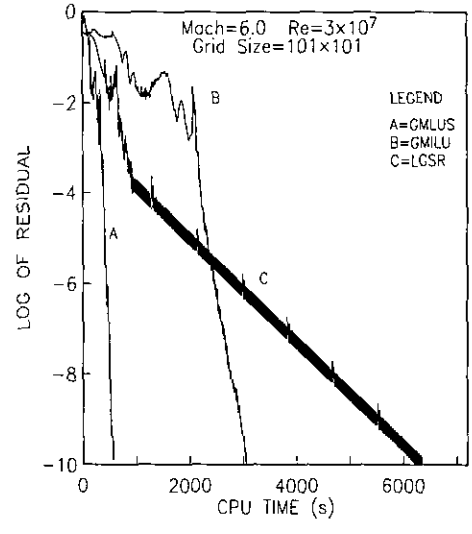

FIG. 15. "CPU time" comparison for hypersonic shock.

and LGSR solvers are increased to 100, 100, and 10, respectively. The LGSR solver converges in 14,225 time-steps for this test case. GMLUS and GMILU require 1900 and 1566 time-steps, with 3463 and 2329 sub-iterations, respectively. Thus GMLUS and GMILU are 7.5 and 9 times faster than LGSR, respectively, in terms of number of time-steps to convergence.

CPU time comparisons for the three solvers are shown in Fig. 15. The results are consistent with the observations for the previous two test cases. GMLUS (475 s) is the most efficient solver in terms of CPU time. It is 6.4 times quicker than GMILU (3058 s) and 12.6 times faster than LGSR (5975 s).

4. CONCLUSIONS

The competitiveness of a proposed preconditioned GMRES solver for CFD applications has been examined in this research. The proposed solver is validated with three diverse fluid-flow test problems—low-speed flow over a backward-facing step, transonic flow through a turbine cascade, and hypersonic flow on a cylindrical leading edge. The proposed preconditioned GMRES solver performs with uniform success with all the test cases and hence presents a viable alternative to existing implicit solvers in CFD research.

This research establishes that the use of an effective, efficient preconditioner is extremely important to the success of GMRES. Recall that an "effective" preconditioner is one that assists the unpreconditioned GMRES solver in obtaining convergence to the steady state solution in the minimum number of time-steps. An "efficient" preconditioner is one which requires minimal overhead (storage and CPU time) when used with the unpreconditioned GMRES solver. An "effective" preconditioner may not necessarily be "efficient," and vice versa. For this research, several preconditioners were investigated for their effectiveness and efficiency. The LUSGS (lower-upper symmetric Gauss–Seidel) scheme of Jameson and Yoon [23] and the ILU (incomplete lower-upper factorization with zero fill-in) preconditioner prove to be good candidates to fill the role of effective and efficient preconditioners. Both ILU and LUSGS are found to be equally effective in their ability to assist GMRES in achieving an excellent convergence rate. Hence, either of the two preconditioners tested in this research could be used with GMRES to construct an implicit solver for new or current CFD codes.

This work represents the first documented use of the LUSGS scheme as a preconditioner in conjunction with Krylov subspace methods (GMRES in particular). GMRES with the LUSGS preconditioner (or GMLUS) is found to be significantly more "efficient" than the GMRES with the ILU preconditioner (or GMILU). This is an important finding related to the development of efficient preconditioners for CFD applications. ILU has been one of the most popular preconditioners used by researchers working with preconditioned Krylov subspace methods. The present research has shown that the LUSGS technique can complete well with ILU in effectiveness, but is remarkably more efficient in terms of the overhead costs associated with the preconditioning effort.

Since the cost of GMRES-like solvers increases with the number of sub-iterations (k), it becomes necessary to "minimize" k at each time-step. The number of sub-iterations (and hence the accuracy to which the linear system is solved) can be restricted without affecting the global convergence of the non-linear CFD problem. An empirical but highly effective "stopping" criteria for determining k has been formulated in this research. The "stopping" criteria is based on the Courant number of the global iteration and can be easily automated for different CFD applications. The use of this criteria allows a variable k and minimizes the computational cost of the preconditioned GMRES solver. In contrast, a fixed k may produce a linear-system solution of very low accuracy (if k is too low), or incur an unnecessarily high cost for each linear-system solve (if k is too high).

One of the goals of this research is to compare the performance of the proposed preconditioned GMRES solvers with a conventional CFD solver. An alternately sweeping, upwind line Gauss–Seidel relaxation (LGSR) algorithm is chosen to represent the school of implicit schemes. The proposed solver provides remarkable speedups over the LGSR solver when converging to a steady-state solution. When the "number of time-steps to convergence" is used as a comparison criterion, the preconditioned GMRES solvers are shown to be 8–17 times faster than the LGSR solver, for the three different test cases (incompressible, transonic, and hypersonic). For "CPU time to convergence" as a comparison criterion, GMLUS is the most efficient solver. Specifically, GMLUS is 13–23 times faster than LGSR and 5–7 times faster than GMILU for the three test cases in this research.

It must be mentioned that "inner iterations" with the LGSR solver [34] can speed up the LGSR solver (by a factor of two in some cases) and thus affect some of the above conclusions. However, Ref. [34] also states that "inner iterations" work better for supersonic flow than for subsonic flow and may thus have limited applicability. The performance of all the solvers researched here could also be improved by the use of concepts like multigrid, mesh sequencing, or similar preconditioning techniques.

As a result of the efforts in this research, an implicit solver based on preconditioned Krylov subspace techniques has been developed for CFD applications. The new solver is demonstrated to be efficient over a wide range of CFD problems and Mach numbers and provides rapid steady state solutions of the Navier–Stokes equations.

ACKNOWLEDGMENTS

This work was sponsored by the Computational Fluid Dynamics Branch, Internal Fluid Mechanics Division of NASA Lewis Research Center. The authors are indebted to Drs. L. A. Povinelli and L. Reid for their support of this research.

REFERENCES

1. S. R. Chakravarthy, *AIAA Pap.* 84-0165 (1984).
2. B. Van Leer and W. A. Mulder, Delft University of Technology Report 84-20, 1984 (unpublished).
3. J. L. Thomas and R. W. Walters, *AIAA Pap.* 85-1501 (1985).
4. J. Douglas and J. E. Gunn, *Numer. Math.* **6** (1964).
5. D. W. Peaceman and H. H. Rachford, *SIAM J.* **3**, 28 (1955).
6. R. M. Beam and R. F. Warming, *AIAA J.* **16**, 393 (1978).
7. J. L. Thomas, B. Van Leer, and R. W. Walters, *AIAA Pap.* 85-1680 (1985).
8. M. R. Hestenes and E. Steifel, *J. Res. Nat. Bur. Stand.* **49**, 409 (1952).
9. Y. S. Wong and M. M. Hafez, ICASE Report 81-15, 1982 (unpublished).
10. L. B. Wigton, N. J. Yu, and D. P. Young, *AIAA J.* **29**, 1092 (1991).
11. Y. Saad and M. H. Schultz, *SIAM J. Sci. Stat. Comput.* 7, 856 (1986).
12. V. Venkatakrishnan, *AIAA Pap.* 90-0586 (1990).
13. B. Van Leer, *Lecture Notes in Physics*, Vol. 170 (Springer-Verlag, New York, 1982), p. 507.
14. K. Ajmani, Ph.D. thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1991 (unpublished).
15. J. K. Reid, in *Large Sparse Sets of Linear Equations* (Academic Press, New York, 1971), p. 231.
16. D. Young and K. C. Jea, *Linear Algebra Appl.* **52**, 399 (1983).
17. O. Axelsson, *Linear Algebra Appl.* **29**, 1 (1980).
18. P. Sonneveld, *SIAM J. Sci. Stat. Comput.* **10**, 36 (1989).
19. H. A. Van der Vorst, *SIAM J. Sci Stat. Comput.* **23**, 631 (1992).
20. R. W. Freund and N. M. Nachtigal, *SIAM J. Sci. Stat. Comput.* **13**, 425 (1992).
21. W. E. Arnoldi, *Quart. Appl. Math.* **9**, 17 (1951).
22. J. A. Meijerink and H. A. Van der Vorst, *J. Comput. Phys.* **44**, 134 (1981).
23. S. Yoon and A. Jameson, *AIAA Pap.* 87-0600 (1987).
24. N. K. Madsen, G. H. Rodrigue, and J. I. Karush, *Inform. Process. Lett.* **5**, 41 (1976).
25. B. F. Armaly, F. Durst, J. C. F. Pereira, and B. Schonung, *J. Fluid Mech.* **127**, 473 (1983).
26. H. J. Dietrichs, J. Hourmouziadis, F. Malzacher, and W. Braunling, in *Proceedings, International Symposium on Airbreathing Engines*, ISABE 87-7031 (1987).
27. C. Sieverding, M. Decuyper, J. Colpin, and O. Amana, *ASME Pap.* 76-GT-30 (1976).
28. A. R. Wieting, NASA TM-100484, 1987 (unpublished).
29. J. C. Tannehill, T. L. Holst, and J. V. Rakich, *AIAA J.* **14**, 204 (1976).
30. W. F. Ng, C. R. Mitchell, K. Ajmani, A. C. Taylor, and J. S. Brock, *AIAA Pap.* 89-0001 (1989).
31. R. L. Sorensen, NASA TM-81198, 1980 (unpublished).
32. B. E. Edney, FFA Report 115, Aeronautical Research Institute of Sweden, 1968 (unpublished).
33. G. M. Kloptex and M. C. Yee, *AIAA Pap.* 88-0233 (1989).
34. A. C. Taylor, Ph.D. thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1989 (unpublished).