AIAA JOURNAL
Vol. 36, No. 11, November 1998

Efficient Newton–Krylov Solver for Aerodynamic Computations

Alberto Pueyo* and David W. Zingg†
University of Toronto, Downsview, Ontario M3H 5T6, Canada

An efficient inexact Newton–Krylov algorithm is presented for the computation of steady two-dimensional aerodynamic flows. The algorithm uses the preconditioned, restarted generalized minimal residual method in matrix-free form to solve the linear system arising at each Newton iteration. The preconditioner is formed using a block-fill incomplete lower–upper factorization of an approximate Jacobian matrix with two levels of fill after applying the reverse Cuthill–McKee reordering. The algorithm has been successfully applied to a wide range of test cases, which include inviscid, laminar, and turbulent aerodynamic flows. In all cases, convergence of the residual to 10^{-12} is achieved with a computing cost equivalent to fewer than 1000 function evaluations. The matrix-free inexact Newton–Krylov algorithm is shown to converge faster and more reliably than an approximate Newton algorithm and an approximately factored multigrid algorithm.

Introduction

MUCH of the effort in the development of efficient solvers for aerodynamic computations is currently concentrated on the use of the multigrid technique. References 1–3 give a good indication of the state of the art. In all three papers, full approximation storage multigrid is used, driven by either an approximately factored implicit algorithm[1] or an explicit multistage method.[2,3] Although multigrid techniques can be very effective, especially with the improvements described in Refs. 1–3, optimal convergence rates have proven elusive for the types of grids required for high-Reynolds-number turbulent flows, which have cells with very high aspect ratios.

This has prompted several researchers to develop quasi-Newton methods, e.g., Refs. 4–14. Quasi-Newton methods can be classified as inexact Newton methods or approximate Newton methods. In an inexact-Newton method, the large linear system arising at each Newton step is solved approximately, using an iterative solver. Hence, there are two levels of iteration, the Newton, or outer, iterations and the inner iterations required to solve the linear system at each Newton iteration. In an approximate Newton method, the functional Jacobian matrix is simplified, thus producing an approximate linearization. The linear system is again solved iteratively.

Within the framework of an inexact Newton solver, there are various strategies that can be adopted and various parameters that must be carefully selected. Some of the important components of an inexact Newton solver include 1) a strategy for choosing the degree of convergence of the inner iterations, 2) an iterative solver for linear systems, 3) a technique for preconditioning the linear system, 4) a method for reordering the unknowns, and 5) a strategy for dealing with the large initial transients in highly nonlinear problems. The choices made can significantly impact the efficiency of the solver, including its speed and memory use.

We present the development and optimization of an inexact Newton solver using the generalized minimal residual algorithm (GMRES) developed by Saad and Schultz[15] as the iterative solver for the linear system. A matrix-free approach is used to form the matrix-vector products required by GMRES. Issues such as optimal choices of preconditioning strategy, ordering of the unknowns, and tolerance level at each Newton step for specific problem classes are addressed. The solver, which is known as PROBE, is evaluated for a wide range of flows over two-dimensional airfoils and compared to other efficient implicit solvers. The objective is to demonstrate that the present solver is both efficient and robust and, thus, provides a promising alternative to solvers currently in use.

In comparing different algorithms, quantities such as the convergence rate or the number of iterations (or multigrid cycles) for a given reduction in the residual can be misleading. The relevant quantity for comparison is the CPU time required for a given level of convergence. Because this is dependent on the computer and the compiler used, we normalize the CPU time by the cost of a single evaluation of the functional or right-hand side. Shortcomings of this choice are that it tends to favor expensive flux evaluation methods (overhead appears smaller) and there is some arbitrariness as to what is included in a function evaluation. For example, local time stepping and the circulation correction are optional. We include the following in defining the cost of one function evaluation: all flux derivative evaluations, the pressure field update, the computation of the artificial dissipation, the computation of the molecular and turbulent viscosity, and the right-hand-side terms at the boundaries. It is also important to note that the memory bandwidth, which may differ substantially from one computer to another, can have an important effect on the relative cost of an inner iteration to a function evaluation because not all of the operations are equally affected by the speed of the access to memory. Nevertheless, the number of function evaluations required for convergence remains a useful standard for comparison. All cases presented here were run on a Pentium Pro 180.

Algorithm Description

Spatial Discretization

The spatial discretization is the same as that used in the well-known solver ARC2D.[16] It consists of second-order centered-difference operators with second- and fourth-difference scalar artificial dissipation. A far-field circulation correction is also included. The Baldwin–Lomax algebraic model[17] is used for turbulent flows. The steady-state solutions of PROBE, thus, are identical to those computed using ARC2D. Grids with a C topology are used in all of our calculations.

Inexact Newton Solver

The spatial discretization leads to a nonlinear system of equations in the form

$$F(Q) = 0 \tag{1}$$

where Q contains the conservative flow variables at every node in the mesh. An inexact Newton method is a generalization of Newton's method, where, at each step k, we try to find ΔQ_k such that

$$\|F(Q_k) + A(Q_k)\Delta Q_k\| \le \eta_k \|F(Q_k)\| \tag{2}$$

where A is an exact linearization of $F(Q)$ given by

$$A = -\frac{\partial F(Q)}{\partial Q} \tag{3}$$

Received Dec. 9, 1996; presented as Paper 97-0877 at the AIAA 35th Aerospace Sciences Meeting, Reno, NV, Jan. 6–9, 1997; revision received May 10, 1998; accepted for publication July 1, 1998.

*Postdoctoral Fellow, Institute for Aerospace Studies, 4925 Dufferin Street.
†Professor, Institute for Aerospace Studies, 4925 Dufferin Street. Member AIAA.

which implies finding a ΔQ_k such that the initial residual $F(Q_k)$ is reduced by a factor of η_k. If $\eta_k = 0$, we recover Newton's method. The local convergence of the inexact Newton method is controlled by η_k. Strategies for choosing a sequence of η_k leading to an efficient rate of convergence of the inexact Newton method will be discussed in a later section.

Linear Iterative Solver

There are several effective iterative solvers for nonsymmetric linear systems available, as reviewed by Dutto.[18] It is very difficult to establish general rules about which one is the best method. This depends on the particular problem one is attempting to solve. Nevertheless, for the type of systems arising in computational fluid dynamics applications, preconditioned Krylov methods have shown better convergence properties than classical stationary methods such as Jacobi, Gauss–Seidel, or successive overrelaxation (SOR). Among Krylov solvers, GMRES is the most popular one, often being faster than other Krylov solvers. We have not done a systematic study of different Krylov solvers, but in a few tests comparing GMRES with stabilized biconjugate gradient (bi-CGSTAB) and conjugate gradient squared (CGS), we found GMRES faster for our applications.

GMRES has the property of minimizing, at every step, the norm of the residual vector over a Krylov subspace of the form $K_k = \text{span}\{v_1, Av_1, A^2v_1, \ldots, A^{k-1}v_1\}$, where $v_1 = r_0/\|r_0\|$ and r_0 is the initial residual given by $r_0 = b - AQ_0$. The storage requirements of GMRES increase linearly and the CPU expense increases quadratically with the number of search directions in the Krylov subspace. To overcome this problem, GMRES is terminated when the size of the Krylov subspace is equal to $m \ll N$, where N is the number of unknowns. GMRES is then restarted using the most recent solution as the initial guess. This is known as restarted GMRES, denoted GMRES(m). Although GMRES(m) can have inferior convergence properties, it is generally essential to use a finite value of m to keep the memory requirements reasonable. Tradeoffs between convergence rates and memory requirements have to be taken into account when determining the maximum size of the Krylov subspace. After testing different values of m, we found that, for our applications, limiting its value to 20 does not significantly degrade the convergence rate.

GMRES requires only matrix-vector products. Because these can be approximated using finite differences,[19] the algorithm can be implemented without forming the Jacobian matrix explicitly. The product of the Jacobian matrix and a vector v can be approximated by

$$A(Q_k)v \simeq \frac{F(Q_k + \varepsilon v) - F(Q_k)}{\varepsilon} \tag{4}$$

The performance of this technique is sensitive to ε. Following Nielsen et al.,[10] we determine ε using the expression

$$\varepsilon \|v\|_2 \simeq \sqrt{\varepsilon_m} \tag{5}$$

where ε_m is the value of machine zero for the computer used. This matrix-free approach is very appealing considering the savings in storage that are made compared to the standard implementation. It was shown in Ref. 20 that it can be advantageous from the performance point of view as well.

Incomplete Lower–Upper Preconditioner

Incomplete lower–upper (ILU) factorizations can be efficient preconditioners for Krylov solvers. In an incomplete factorization, we approximate the matrix A by a matrix M such that

$$A = M - E = LU - E \tag{6}$$

where L is a lower-triangular matrix, U is an upper-triangular matrix, and E is an error matrix. The factors are computed using a Gaussian elimination process applied to the matrix A or to some reasonable approximation of it.

The factorization may be more or less accurate, depending on how many new nonzero entries we retain in the factorization compared to the original matrix. The cost of forming the preconditioner and storage goes up when we allow more fill-in in L and U; on the other hand, increases in robustness and efficiency often justify more accurate factorizations, particularly when the preconditioner is used in solving several systems, because the cost of forming the preconditioner is amortized. There have been two distinct approaches to forming such incomplete factorizations: level of fill-in and threshold strategies. The first approach, ILU(p), uses only the graph of the matrix to determine which entries to drop. In contrast, ILU threshold [ILUT(P, τ)], developed by Saad,[21] uses two rules to determine which elements should be dropped at a given row. The first rule consists of dropping any element smaller than a relative tolerance determined by τ and a norm of the original matrix. The maximum number of nonzeros in a given row is controlled by the second parameter P.

The matrices arising from the linearization of the Navier–Stokes equations present a block structure, with blocks of size 4×4. That is why block versions of ILU(p) are quite popular in this type of application. In cases where a low level of fill-in is used, i.e., $p \leq 1$, the iterative solver may fail with the scalar version and converge with the block version. An alternative approach is to use the scalar version but treat the zeros within the 4×4 blocks as if they were nonzeros, in other words, allowing fill-in in those positions. We call this strategy block-fill ILU(p) [BFILU(p)]. Because of the storage format that we use, namely, compressed sparse row (CSR),[22] we find the BFILU(p) approach more convenient than block ILU(p). ILUT(P, τ) does not have an equivalent BFILUT because it already considers all of the possible nonzeros in a row and keeps the larger ones.

One might expect that a preconditioner based on the exact Jacobian matrix would be more effective than one based on an approximate Jacobian. However, this assumption is not valid. In the case of nonsymmetric nondiagonally dominant matrices such as the ones arising in our applications, the incomplete factors can be more ill conditioned than the original matrix[23] and the long recurrences associated with backward and forward solves may be unstable.[24,25] Because diagonal dominance tends to alleviate this problem,[26] we can benefit by using an approximate Jacobian matrix with reduced off-diagonal dominance.

The off-diagonal dominance of the Jacobian matrix can be reduced by using only second-difference dissipation in forming it. The coefficient of the second-difference dissipation in the matrix is computed as a linear combination of the second- and fourth-difference artificial dissipation coefficients on the right-hand side, ε_2^r and ε_4^r, respectively, as follows:

$$\varepsilon_2^l = \varepsilon_2^r + \sigma \varepsilon_4^r \tag{7}$$

where σ is a constant left as a free parameter. This approximation has the additional benefit that the storage of this modified matrix, which has five blocks per node, is significantly smaller than the storage required by the exact Jacobian matrix, which has nine blocks per node. Note that these approximations are made only in the matrix used to form the preconditioner. They do not affect the functional; thus the converged solution is not affected.

Ordering of Unknowns

The ordering of the unknowns of the linear system plays an important role in the convergence of the preconditioned iterative solver.[4,27] The natural ordering of the nodes on a C grid does not lead to a fully banded matrix because of coupling between cells on either side of the wake cut arising from the grid topology. In a complete factorization, these off-diagonal elements create a great number of nonzeros in the L and U factors, which may not be retained in an incomplete factorization, reducing the quality of the preconditioner. To overcome this problem, we have studied reordering strategies that are applied before computing the factorization. Besides the natural ordering (NAT), we consider an ordering based on numbering the nodes across the wake cut and the airfoil, designated double bandwidth (DB). We have eliminated the nonzeros outside of the main diagonals, but the bandwidth is now twice as large, which can adversely affect the quality of the factorization. To obtain more consistent performance, we test two of the reordering algorithms typically used for unstructured grids. The reverse Cuthill–McKee (RCM) strategy[28] is a well-known bandwidth reduction algorithm. The minimum neighboring (MN) algorithm[29] is a modification of the minimum-degree reordering of George and Liu.[30] It was designed to minimize the fill-in produced in a factorization.

Table 1 Parameters for the seven flows studied[a]

Case	Airfoil	Ma	α	Re	tr. up	tr. low
1	NACA[b]	0.63	2.00	invisc	——	——
2	NACA	0.80	1.25	invisc	——	——
3	NACA	0.80	5.00	$5.00e2$	——	——
4	NACA	0.30	0.00	$2.88e6$	$0.43c$	$0.43c$
5	NACA	0.30	6.00	$2.88e6$	$0.05c$	$0.80c$
6	NACA	0.70	1.49	$9.00e6$	$0.05c$	$0.05c$
7	RAE[c]	0.729	2.31	$6.50e6$	$0.03c$	$0.03c$

[a]tr. up, transition point on the upper surface; tr. low, transition point on the lower surface. [b]NACA 0012 airfoil. [c]RAE 2822 airfoil.

Startup

A well-known difficulty of Newton's method is that the early iterations may diverge if the initial solution is far from the final solution, especially for flows with shocks. There are several techniques that can be used to overcome this problem. Damping the Newton updates,[31] using a finite time step,[32] and using an approximate Newton method for the first one or two orders of magnitude reduction in the outer residual are some of the options. In our experience, using an approximate Newton strategy is the most efficient of the three.[20] However, it is possible to employ a cheaper relaxation algorithm.[10] In PROBE, the approximately factored algorithm of ARC2D[16] in diagonal form is used with two levels of mesh sequencing. We do 150 iterations or reduce the residual two orders of magnitude, whichever comes first, on a coarse grid, followed by five iterations on the fine grid. This strategy reduces the CPU time of the startup by a factor of two to three compared with the approximate Newton strategy.

Test Cases

Six test cases are studied using the NACA 0012 airfoil and one using the RAE 2822 supercritical airfoil. Two are inviscid flows, one is laminar, and four are turbulent. The parameters defining the test cases are given in Table 1. The initial condition is always freestream flow.

For the inviscid cases, the grid has 249×39 nodes with the wall spacing set to 2×10^{-3} chords. For the laminar case, the grid has 249×49 nodes and a wall spacing of 5×10^{-4} chords. A 331×51 grid with the wall spacing set to 1×10^{-5} chords is used for the NACA 0012 turbulent cases. For the RAE airfoil case, the grid has 321×49 nodes with similar wall spacing. In some tests, which include comparisons with multigrid, the number of nodes may vary slightly to be able to obtain coarser grids by removing every other point in the finer grid. These grids provide reasonable numerical accuracy for the flows considered. Maximum cell aspect ratios for these grids range between 8.7×10^2 and 1.16×10^5.

Algorithm Optimization

Approximate Matrix Inversion

The convergence of the inexact Newton method depends on the values of η_k. As shown by Dembo et al.,[33] certain choices of η_k lead to superlinear or quadratic convergence in terms of iterations. However, these strategies are not necessarily the most efficient in terms of CPU time because they tend to impose a tight tolerance in solving the linear system. During the first few Newton iterations after the startup phase, the solution is still relatively far from the converged solution, and thus the linear approximation made by Newton's method can be very inaccurate. Solving the linear system of equations very accurately does not, in general, equally reduce the residual of the nonlinear system of equations, which causes a waste of CPU time. This is known as oversolving.[34] To illustrate this point, several levels of reduction of the residual of the linear problem arising at each quasi-Newton step have been tested. The convergence histories for six values are shown in Fig. 1. BFILU(2) based on an approximate Jacobian matrix is used as a preconditioner for GMRES. Strict inner tolerances reduce the number of outer iterations, leading to superlinear or quadratic convergence, but there is an increased number of inner iterations due to oversolving. The results indicate that, although is desirable to achieve rapid convergence in terms of outer iterations, it is more important to avoid oversolving.

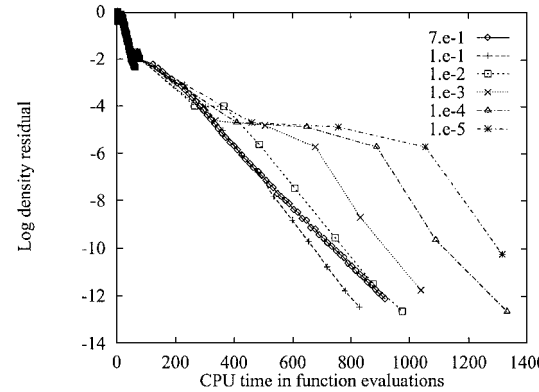

Fig. 1 Convergence history for different levels of reduction of the inner residual using matrix-free Newton GMRES. Each symbol represents one outer iteration.

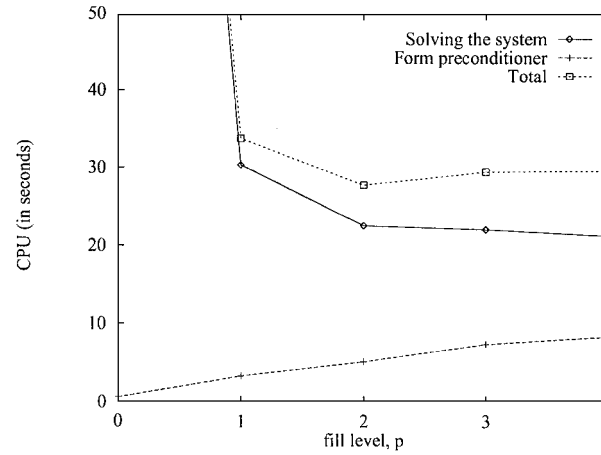

Fig. 2 CPU cost of forming the BFILU(p) factorization and effectiveness in reducing the inner residual by two orders of magnitude.

Eisenstat and Walker[34] proposed some sequences of inner tolerances controlled by safeguards, which prevent η_k from becoming too small too quickly. However, for our applications, we have found the following approach to give slightly better efficiency. We use $\eta_k = 0.5$ for the first 10 outer iterations; we then use $\eta_k = 0.1$ for the remaining outer iterations. We limit the number of GMRES iterations to 40 as a safeguard. This approach, which guarantees no oversolving for most problems, results in linear convergence, but savings in not having to compute η_k and better behavior regarding oversolving make it more efficient in terms of CPU time than the other approaches. Note that undersolving at each Newton step is not particularly wasteful, whereas oversolving should be avoided.

Preconditioning Strategies

Two main issues are considered in this section. 1) Which of the two factorizations, i.e., BFILU or ILUT, is more efficient, and what is the optimal level of fill? 2) Can the first-order Jacobian A_1 produce a better preconditioner than the exact Jacobian A_2, and what is the optimum value of σ for the first-order Jacobian?

The level of fill p allowed in the factorization should be chosen based on storage considerations, the cost of forming the preconditioner, and its effectiveness in reducing the cost of solving the linear system. Figure 2 shows the CPU time required by preconditioned GMRES to reduce the inner residual by two orders of magnitude with different levels of fill for a representative linear system. The approximate Jacobian A_1, which requires roughly half of the storage of A_2, was used to form the preconditioner. Allowing some fill greatly improves the performance of the solver at a very reasonable memory cost. If the CPU time to form the preconditioner is included, $p = 2$ is the most efficient level of fill. It requires 1.73 times more memory than $p = 0$. Therefore, using $p = 2$ seems an optimum choice because it gives the best performance with a storage that is still below that of the exact Jacobian matrix.

Table 2 Relative time to converge for two ILU preconditioners, using RCM and MN reorderings

	ILU(2)		ILUT(15, 0.1)	
Case	RCM	MN	RCM	MN
1	1	1	1.21	1.09
2	1	1.07	1.32	1.52
3	1	1.72	DNC[a]	1.54
4	1	1.20	DNC	1.25
5	1.07	1	1.64	1.35
6	1.07	1	DNC	1.16
7	1	1.18	DNC	DNC

[a]Case did not converge with the given parameters.

Table 3 Frobenius norm of the error matrix and of the preconditioned error matrix for the first and second-order preconditioners

Preconditioner	$\|E\|_F/\|A_2\|_F$	$\|EM_i^{-1}\|_F/\|I\|_F$
M_1	0.4870	4.6003
M_2	0.2576	6676.4449

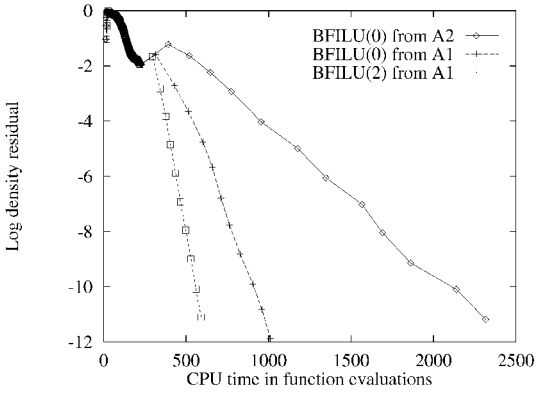

Fig. 3 Convergence histories for case 1 in a grid of 143×20 nodes, using three preconditioners: BFILU(0) formed from the first-order Jacobian A_1, BFILU(0) using the exact Jacobian A_2, and BFILU(2) from the first-order Jacobian.

After testing many combinations, we have also shown that good choices of the parameters P and τ of ILUT are 15 and 0.1, respectively.[35] This produces an efficient preconditioner, whereas the number of nonzeros is similar to that of BFILU(2). Table 2 shows the performance of BFILU(2) and ILUT(15, 0.1) for the seven test cases. Two different reorderings are shown, RCM and MN. These are further discussed subsequently. Table 2 shows that, for this application, BFILU(2) is superior to ILUT(15, 0.1) for both orderings.

Figure 3 shows the convergence histories obtained with three different preconditioners for case 1 on a coarse grid (143×20): BFILU(0) formed from the first-order and exact Jacobians and BFILU(2) formed from the first-order Jacobian. Note that the number of nonzeros given by BFILU(2) using A_1 is about the same as the number of nonzeros given by BFILU(0) using A_2. BFILU(2) applied to A_2 requires excessive storage and is not considered here.

The results in Fig. 3 show that the preconditioners built from A_1 are more efficient than the one built from the exact Jacobian. After preconditioning, Eq. (6) becomes

$$AM^{-1} = I - EM^{-1} \tag{8}$$

Because we are solving the preconditioned system, the matrix E is not as important as the preconditioned error matrix EM^{-1}. As Saad[26] points out, when A is not diagonally dominant, L^{-1} and U^{-1} may have very large norms, causing $EM^{-1} = EU^{-1}L^{-1}$ to be very large and, thus, adding large perturbations to the identity matrix. In that case, the eigenvalues of AM^{-1} will not be nicely clustered around unity, and the iterative solver will show slower convergence.

In Table 3, the Frobenius norms of the error matrices are presented for BFILU(0). The results confirm that M_1, formed from A_1, produces an error matrix E that has a bigger norm than the one from M_2, formed from A_2, but the norm of its preconditioned error matrix is much smaller than the one from M_2. The matrix A_1 is more diagonally dominant than A_2 because we are adding a large amount of second-difference dissipation with stencil $(1, -2, 1)$, whereas A_2 has fourth-difference dissipation with stencil $(-1, 4, -6, 4, -1)$.

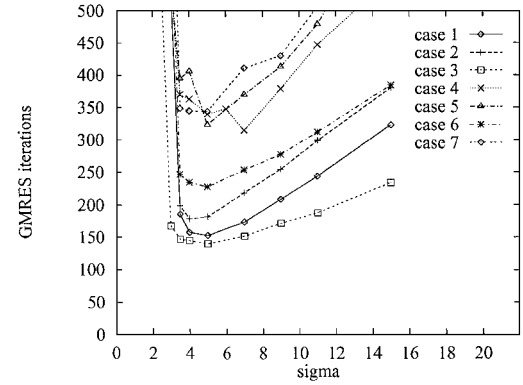

Fig. 4 Total number of GMRES iterations required to converge to machine zero, for different values of σ for the seven cases.

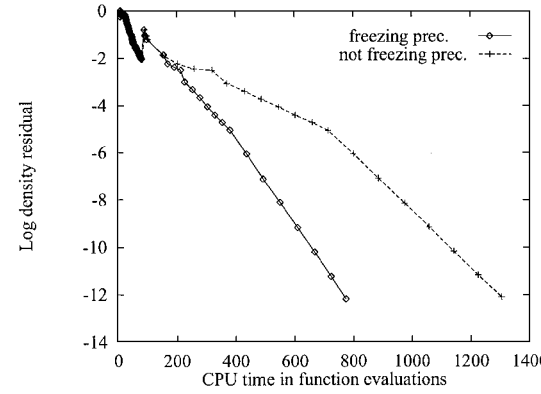

Fig. 5 Convergence history for case 2, freezing the preconditioner and updating the preconditioner at each Newton iteration.

The parameter σ in Eq. (7) controls the amount of second-difference dissipation added to A_1. As we increase its value, the matrix is less nondiagonally dominant and more symmetric, which will benefit the factorization. On the other hand, the matrix will be a less accurate representation of A. Figure 4 shows the total number of GMRES iterations required to converge to machine zero for the seven cases, as we vary the value of σ. As expected, there is an optimum value; for all cases, with the exception of case 4, it is equal to 5. For larger values, the number of GMRES iterations increases gradually, and for smaller values, it increases sharply. In most cases, the code does not converge for $\sigma \leq 3$.

Consideration has been given to computing the preconditioner once, at the first Newton iteration, and then freezing it. To see how this strategy affects the performance of the solver, we have run a case that has strong initial nonlinearities, case 2, with and without updating the preconditioner. Our experiments show that the number of GMRES iterations at each Newton iteration does not increase when we freeze the preconditioner. There is no benefit from updating the preconditioner at each Newton iteration. As a result, there are substantial CPU savings when the preconditioner is frozen, as shown in Fig. 5. This is related to our startup procedure in which an approximately factored algorithm is used to eliminate the most significant transients before initiating the inexact Newton process. Apparently, subsequent changes in the flowfield do not degrade the effectiveness of the preconditioner.

Ordering of Unknowns

The four orderings described, NAT, DB, RCM, and MN, have been tested for the seven cases. In Fig. 6 we show results for case 7

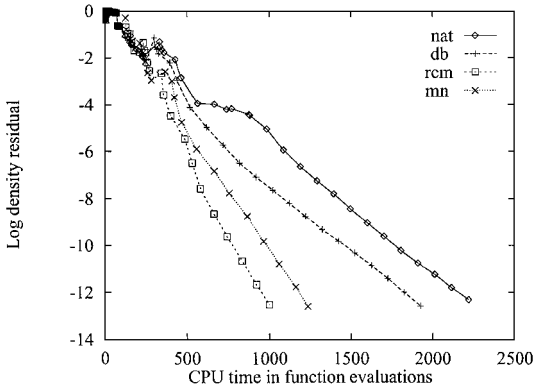

Fig. 6 Convergence history for NAT, DB ordering, RCM, and MN for case 7, using BFILU(2) as preconditioner.

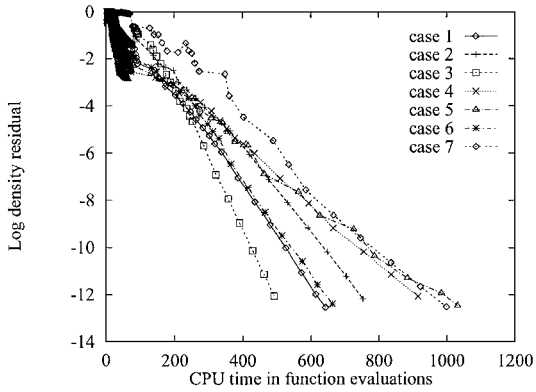

Fig. 7 Residual history for PROBE for the seven cases described in Table 1.

using BFILU(2) as preconditioner. DB is more efficient than NAT. The benefits of applying reordering techniques such as RCM and MN are clearly shown. The performance of MN is virtually independent of the initial ordering, NAT or DB. In contrast, the performance of RCM depends greatly on the initial ordering, with DB preferred. Table 2 shows that, with BFILU(2), faster convergence is achieved with RCM for cases 1, 4, and 7, whereas for cases 5 and 6, MN is faster. Although further study with a wider range of cases is required, we concentrate on RCM for the remainder of the paper. With ILUT(15, 0.1), MN is clearly superior to RCM; however, we do not consider this approach further, as it is slower than the combination of RCM and BFILU(2) for all seven cases.

Optimized Algorithm

The following are the main strategies and parameters of PROBE: 1) inexact Newton strategy; 2) matrix-free GMRES(20); 3) BFILU(2) preconditioner based on the first-order Jacobian formed using $\sigma = 5$; 4) RCM reordering; 5) preconditioner computed at first iteration and not updated; 6) inner tolerance (η_k) set to 0.5 for the first 10 outer iterations, 0.1 for the remainder; as a safeguard, inner iterations are limited to 40; and 7) approximate factorization algorithm used for 150 iterations or to reduce the residual two orders of magnitude, whichever comes first, on a coarse grid, followed by 5 iterations on the fine grid.

Convergence Results

The convergence histories for the seven cases are shown in Fig. 7 as a function of the CPU time normalized by the CPU time required for one function evaluation (FE). Reduction of the residual by 12 orders of magnitude is achieved in 500–1000 FEs for all of the cases. Table 4 shows some statistics for PROBE, including the number of inner and outer iterations required to reduce the residual norm by 12 orders of magnitude. The outer iterations include only those done using the Newton–Krylov solver and not those of the approximately factored algorithm. CPU/FE gives the total run time normalized by

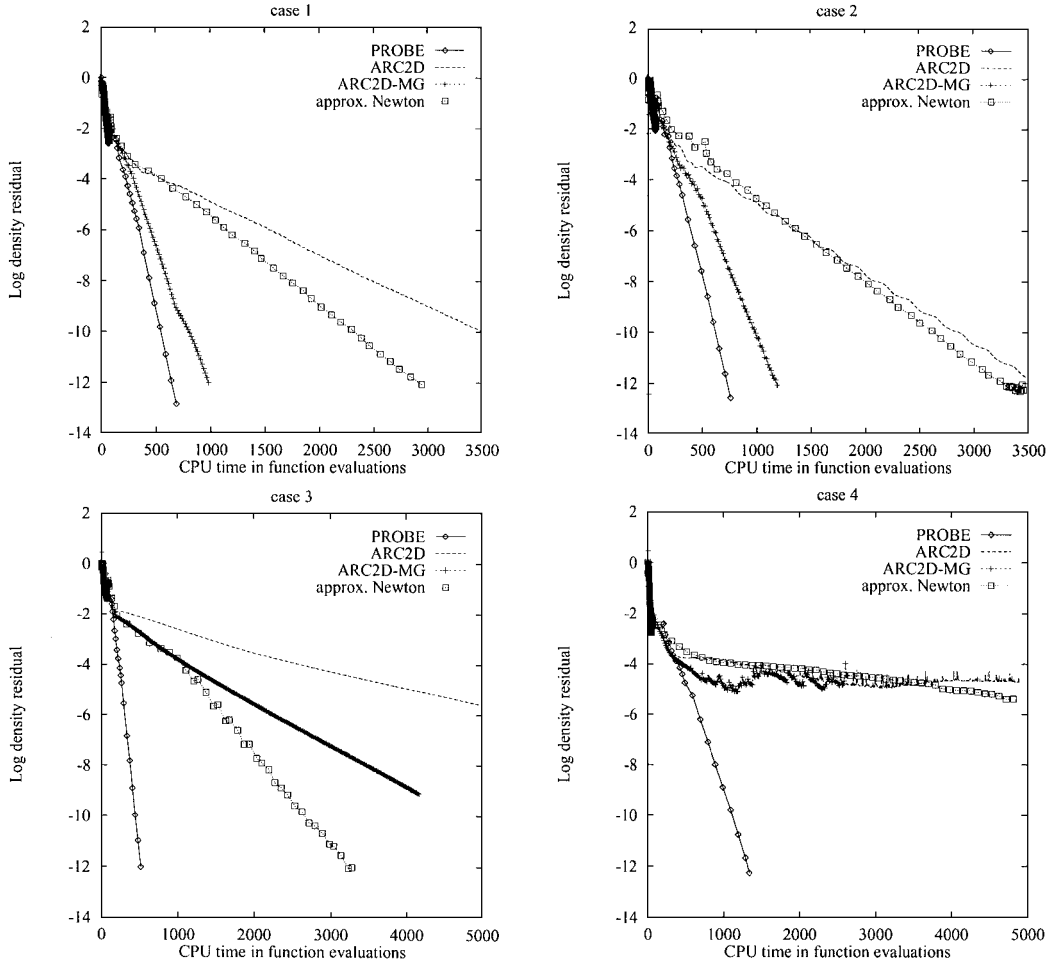

Fig. 8 Cases 1–4: convergence history for the inexact Newton–Krylov method (PROBE), the approximately factored method (ARC2D), the approximately factored method with three-level multigrid (ARC2D-MG), and the approximate Newton method (approx. Newton).

Table 4 Statistics for the Newton–Krylov algorithm for the cases studied

Case	O-it[a]	Σi-it[b]	I-it/o-it[c]	CPU/FE[d]	CPU, min/s
1	18	157	8.7	615.5	1/53.9
2	18	190	10.6	743.8	2/17.6
3	18	129	7.2	490.7	2/27.2
4	17	324	19.1	910.1	8/06.0
5	19	370	19.5	990.8	8/49.1
6	18	227	12.6	642.9	5/43.3
7	22	354	16.1	953.5	7/47.2

[a]Outer iterations.
[b]Total number of inner iterations.
[c]Average number of inner iterations per outer iteration.
[d]CPU time in FE calls to reduce the residual by 12 orders of magnitude.

the CPU time of an FE. The total time in minutes and seconds is also provided.

Comparison with Other Solvers

In this section, we compare our inexact Newton–Krylov strategy with three other implicit solvers, which are briefly described as follows.

Approximate Factorization (ARC2D)

The approximate factorization algorithm in diagonal form, as used in ARC2D, is explained in detail in Ref. 16. In ARC2D, the wake cut can be treated implicitly or explicitly. In this paper we consider only the explicit treatment of the wake cut, which is faster for our test cases.

Approximate Factorization with Multigrid

Other authors have already shown that multigrid can substantially increase the convergence rate of the approximate factorization algorithm.[1,36] For our study, we use a three-level sawtooth cycle with four iterations of the approximately factored algorithm at each level in the cycle. This approach produces optimum convergence in terms of CPU time.[36]

Approximate Newton

Some of the most popular approximate Newton methods use a first-order Jacobian on the left-hand side. One of the original reasons is that this matrix requires less storage than the exact Jacobian. Another reason for using a first-order Jacobian is that it is better conditioned than the exact Jacobian; hence, the inner iterations can converge faster. The penalty is an increased number of outer iterations. Approximate Newton solvers that use a first-order Jacobian are typically preconditioned with a block ILU(0) factorization. Here we use BFILU(0), which is extremely similar. With this preconditioner, the solver requires approximately the same amount of storage as PROBE. An efficient strategy that helps to reduce the overall CPU time without harming the convergence rate consists of freezing the left-hand side after the first 10 iterations. The preconditioner is updated only three times during the first 10 iterations, and then it is also frozen.

Performance Comparison

The residual convergence histories of PROBE are compared with those of the other three solvers in Figs. 8 and 9. The approximate Newton solver is robust and considerably faster than ARC2D for all cases but one, but it is much slower than PROBE, while requiring the same amount of storage. In general, PROBE is significantly faster than the multigrid solver. The difference is particularly striking in cases 3 and 4. For case 5, PROBE does not converge beyond 10^{-9} on this particular grid. This is due to the turbulence model. When we freeze the eddy viscosity, convergence to machine zero is achieved in a similar number of function evaluations as in the other cases.

Because we measure the CPU time in terms of FEs, it is useful to define the convergence rate as the reduction in the residual obtained in the CPU time required for one FE. This provides a convergence rate based on CPU time rather than iterations or multigrid cycles. With this definition, PROBE produces an average convergence rate

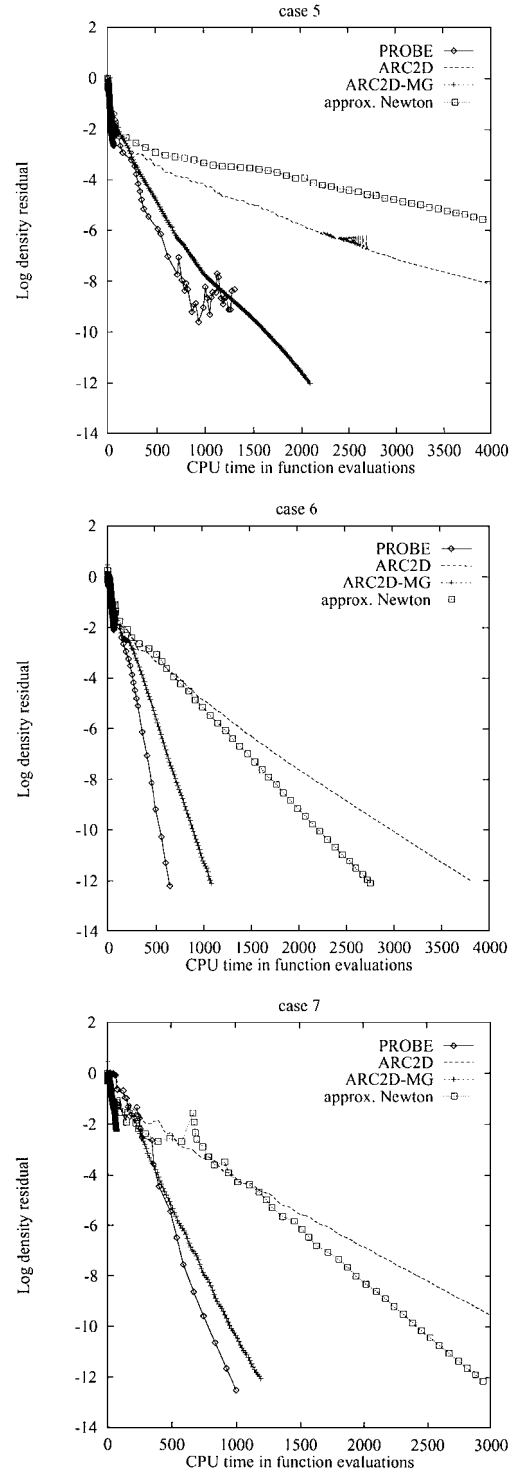

Fig. 9 Cases 5–7: convergence history for the inexact Newton–Krylov method (PROBE), the approximately factored method (ARC2D), the approximately factored method with three-level multigrid (ARC2D-MG), and the approximate Newton method (approx. Newton).

of between 0.945 and 0.972, based on the data in Table 4. For comparison, a multigrid method with a convergence rate of 0.75 per multigrid cycle and a cost per multigrid cycle equal to that of five FEs produces a convergence rate per FE of $0.75^{1/5} = 0.944$. Given the difficulty of achieving a convergence rate of 0.75 per multigrid cycle for turbulent computations, it is clear that the performance of PROBE is excellent.

Conclusions

We have presented, justified, and optimized the basic components of PROBE, a Newton–Krylov solver for aerodynamic flows. The main conclusions can be summarized as follows:

1) Oversolving is avoided and optimum performance is obtained by setting the inner tolerance to 0.5 for the first 10 iterations and to 0.1 for the remainder, once the transients are less severe. As a safeguard, the total number of GMRES iterations at each Newton iteration is limited to 40.

2) For the same storage requirements, the BFILU(p) is a more efficient preconditioner than ILUT(P, τ). The best performance/memory ratio was obtained for BFILU(2).

3) It is possible to produce better-conditioned factorizations with a well-chosen approximate Jacobian matrix than with the exact Jacobian matrix.

4) A parameter was used in the approximation of the Jacobian matrix. It controls the amount of artificial dissipation in the matrix. It shows an optimum value that significantly improves the efficiency of the preconditioner.

5) The use of a fixed preconditioner does not adversely affect convergence of the inner iterations. Therefore, the factorization is computed only once, which reduces the CPU cost substantially.

6) RCM reordering applied to the DB ordering produces the fastest convergence of the four ordering strategies that have been studied.

For a range of flows, the CPU time required for a residual reduction of 12 orders of magnitude is equivalent to the cost of 500–1000 FEs. This corresponds to a convergence rate per FE between 0.945 and 0.972. We have demonstrated that our inexact Newton–Krylov algorithm converges faster and more reliably than an approximate Newton algorithm and an approximately factored multigrid algorithm.

Acknowledgments

The first author was supported by a Fonds pour la Formation de Chercheurs et l'Aide a la Recherche grant of the Government of Québec and an Ontario Graduate Scholarship of the Government of Ontario. This work was supported by De Havilland, Inc., and the Natural Sciences and Engineering Council of Canada.

References

[1]Jespersen, D., Pulliam, T., and Buning, P., "Recent Enhancements to OVERFLOW," AIAA Paper 97-0644, Jan. 1997.

[2]Mavriplis, D. J., "Multigrid Strategies for Viscous Flow Solvers on Anisotropic Unstructured Meshes," AIAA Paper 97-1952, June 1997.

[3]Pierce, N. A., Giles, M. B., Jameson, A., and Martinelli, L., "Accelerating Three-Dimensional Navier–Stokes Calculations," AIAA Paper 97-1953, June 1997.

[4]Dutto, L. C., "The Effect of Ordering on Preconditioned GMRES Algorithm, for Solving the Compressible Navier–Stokes Equations," *International Journal for Numerical Methods in Engineering*, Vol. 36, 1993, pp. 457–497.

[5]Dutto, L. C., "Etude de préconditionnements pour la résolution, par la méthode des éléments finis, des équations de Navier–Stokes pour un fluide compressible," Ph.D. Thesis, Dept. of Applied Mathematics, Univ. Pierre et Marie Curie, Paris, France, Nov. 1990.

[6]Johan, Z., Hughes, T. J. R., and Shakib, F., "A Globally Convergent Matrix-Free Algorithm for Implicit Time-Marching Schemes Arising in Finite Element Analysis in Fluids," *Computational Methods in Applied Mechanical Engineering*, Vol. 87, 1991, pp. 281–304.

[7]Degrez, G., and Issman, E., "Acceleration of Compressible Flow Solvers by Krylov Subspace Methods," Lecture Notes, von Kármán Inst. for Fluid Dynamics Lecture Series 1994-05, 1994.

[8]Luo, H., Baum, J. D., Löhner, R., and Cabello, J., "Implicit Schemes and Boundary Conditions for Compressible Flows on Unstructured Meshes," AIAA Paper 94-0816, Jan. 1994.

[9]Rogers, S. E., "A Comparison of Implicit Schemes for the Incompressible Navier–Stokes Equations with Artificial Compressibility," AIAA Paper 95-0567, June 1995.

[10]Nielsen, E. J., Anderson, W. K., Walters, R. W., and Keyes, D. E., "Application of Newton–Krylov Methodology to a Three-Dimensional Unstructured Euler Code," AIAA Paper 95-1733, June 1995.

[11]Barth, T. J., and Linton, S. W., "An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation," AIAA Paper 95-0221, Jan. 1995.

[12]Anderson, W. K., Rausch, R. D., and Bonhaus, D. L., "Implicit/Multigrid Algorithms for Incompressible Turbulent Flows on Unstructured Grids," AIAA Paper 95-1740, June 1995.

[13]Delanaye, M., Geuzaine, P., Essers, J. A., and Rogiest, P., "A Second-Order Finite-Volume Scheme Solving Euler and Navier–Stokes Equations on Unstructured Grids," AGARD 77th Fluid Dynamics Panel Symposium on Progress and Challenges in Computational Fluid Dynamics Methods and Algorithms, Seville, Spain, Oct. 1995.

[14]Blanco, M., and Zingg, D. W., "Fast Newton–Krylov Solver for Unstructured Grids," *AIAA Journal*, Vol. 36, No. 4, 1998, pp. 607–612.

[15]Saad, Y., and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, 1986, pp. 856–869.

[16]Pulliam, T. H., "Efficient Solution Methods for the Navier–Stokes Equations," Lecture Notes, von Kármán Inst. for Fluid Dynamics Lecture Series, Jan. 1986.

[17]Baldwin, B. S., and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78-257, June 1978.

[18]Dutto, L. C., "On Iterative Methods for Solving Linear Systems of Equations," *Revue européenne des Éléments finis*, Vol. 2, No. 4. 1993, pp. 423–448.

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

[20]Pueyo, A., and Zingg, D. W., "Airfoil Computations Using a Newton-GMRES Method," *CFD96, Proceedings of the Fourth Annual Conference of the Computational Fluid Dynamics Society of Canada*, June 1996.

[21]Saad, Y., "ILUT: A Dual Threshold Incomplete LU Factorization," Supercomputer Inst., TR UMSI 92/38, Univ. of Minnesota, Minneapolis–St. Paul, MN, March 1992.

[22]Saad, Y., "SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations, version 2," Computer Science Dept., Univ. of Minnesota, Minneapolis, MN, June 1994.

[23]Chow, E., and Saad, Y., "Experimental Study of ILU Preconditioners for Indefinite Matrices," Supercomputing Inst., TR UMSI 97/95, Univ. of Minnesota, Minneapolis–St. Paul, MN, June 1997.

[24]Elman, H. C., "A Stability Analysis of Incomplete LU Factorizations," *Mathematics of Computation*, No. 47, 1986, pp. 191–217.

[25]Bruaset, A. M., Tveito, A., and Winther, R., "On the Stability of Relaxed Incomplete LU Factorizations," *Mathematics of Computation*, No. 54, 1990, pp. 701–719.

[26]Saad, Y., "Krylov Subspace Techniques, Conjugate Gradients, Preconditioning and Sparse Matrix Solvers," von Kármán Inst. for Fluid Dynamics, Lecture Series 1994-05, March 1994.

[27]Venkatakrishnan, V., and Mavriplis, D. J., "Implicit Solvers for Unstructured Meshes," AIAA Paper 91-1537, June 1991.

[28]Cuthill, E. H., and McKee, J. M., "Reducing the Bandwidth of Sparse Symmetric Matrices," *Proceedings of the 24th National Conference of the Association for Computing Machinery*, Brondon Press, 1969, pp. 157–172.

[29]Martin, G., "Méthodes des préconditionnement par factorisation incomplète," Mémoire de Maîtrise, Univ. de Laval, Québec, PQ, Canada, 1991.

[30]George, A., and Liu, J. W. H., "The Evolution of the Minimum Degree Ordering Algorithm," *SIAM Review*, No. 31, 1989, pp. 1–19.

[31]McHugh, P. R., and Knoll, D. A., "Comparison of Standard and Matrix-Free Implementations of Several Newton–Krylov Solvers," *AIAA Journal*, Vol. 12, No. 12, 1994, pp. 2394–2400.

[32]Mulder, W. A., and Van Leer, B., "Implicit Upwind Methods for the Euler Equations," AIAA Paper 83-1930, June 1983.

[33]Dembo, R. S., Eisenstat, S. C., and Steihaug, T., "Inexact Newton Methods," *SIAM Journal on Numerical Analysis*, Vol. 19, No. 2, 1982, pp. 400–408.

[34]Eisenstat, S. C., and Walker, H. F., "Choosing the Forcing Terms in an Inexact Newton Method," *SIAM Journal on Scientific and Statistical Computing*, Vol. 17, No. 1, 1996, pp. 16–32.

[35]Pueyo, A., "An Efficient Newton–Krylov Method for the Euler and Navier–Stokes Equations," Ph.D. Thesis, Inst. for Aerospace Studies, Univ. of Toronto, Downsview, ON, Canada, Feb. 1998.

[36]Chisholm, T., and Zingg, D. W., "Multigrid Acceleration of an Approximately Factored Algorithm for Aerodynamic Flows," 44th Annual Conf. of the Canadian Aeronautics and Space Inst., Toronto, ON, Canada, April 1997.

J. Kallinderis
Associate Editor