IMPROVEMENTS TO A NEWTON-KRYLOV SOLVER FOR AERODYNAMIC FLOWS

Alberto Pueyo* David W. Zingg†

University of Toronto Institute for Aerospace Studies
4925 Dufferin Street, Downsview, Ontario
M3H 5T6 Canada

Abstract

In this paper, we present and justify the strategies and parameters used in PROBE, a Newton-Krylov solver for steady aerodynamic flows. The Krylov solver GMRES is used in matrix-free form. An approximate Jacobian matrix with some modifications to increase the magnitude of the diagonal entries is used to form a preconditioner based on an incomplete lower-upper factorization with two levels of fill. The resulting solver is very efficient, generally reducing the residual twelve orders of magnitude with a CPU expense equivalent to between 500 and 1000 function evaluations. For all cases studied, PROBE converged significantly faster than an approximately-factored multigrid algorithm used for comparison. We propose a convergence rate based on the reduction in the residual obtained in the CPU time required for one function evaluation. With this definition, PROBE achieves a convergence rate per function evaluation between 0.945 and 0.972 for the cases studied, which include inviscid, laminar, and turbulent flows. This is substantially faster than many current algorithms applied to similar flows.

Introduction

Most of the effort in the development of efficient solvers for aerodynamic computations is currently concentrated on the use of the multigrid technique. Refs. [1] to [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 multi-stage method [2, 3]. Although multigrid techniques can be very effective, especially with the improvements described in the above papers, 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] to [17]. 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. In an earlier paper [18], we showed that inexact-Newton methods are more efficient than approximate-Newton methods.

Within the framework of an inexact-Newton solver, there are various strategies which can be adopted, and various parameters which must be carefully selected. Some of the important components of an inexact-Newton solver include

1. an iterative solver for linear systems,
2. a strategy for choosing the degree of convergence of the inner iterations,
3. a technique for preconditioning the linear system,
4. a node reordering technique,
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.

In this paper, we present and justify the strategies and parameters selected for our inexact-Newton solver, known as PROBE. Particular emphasis is given to the approach

*Graduate Student.
†Associate Professor, Member AIAA.

used in preconditioning the linear system. In addition, we compare the performance of PROBE with ARC2D-MG [19], an approximately-factored multigrid algorithm similar to that presented in Ref. [1]. In comparing such different algorithms, quantities such as the convergence rate or the number of iterations (or multigrid cycles) for a given reduction in the residual are not useful. The relevant quantity for comparison is the CPU time required for a given level of convergence. Since this is dependent on the computer used, we normalize the CPU time by the cost of a single evaluation of the functional or "right-hand-side." For algorithms using the same spatial discretization (such as PROBE and ARC2D-MG), this provides a comparison which is directly proportional to CPU time but is independent of the computer used. For algorithms using different spatial discretizations, the number of function evaluations required for convergence remains a useful standard for comparison.

Inexact-Newton solver

In this section, we review the basic strategies chosen for PROBE. Most of these were selected based on testing presented in [18, 20]. Details and justification of the preconditioning technique are given in a later section of the paper.

Start up 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 [21], using a time-marching method [22] or employing a cheaper relaxation algorithm [12] are some of the options. In PROBE, the approximately-factored algorithm of ARC2D [23] 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.

Inexact-Newton strategy After applying a spatial discretization, we obtain a nonlinear system of algebraic equations in the form

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

where Q is the solution vector of conservative variables. An inexact-Newton method is a generalization of Newton's method where, at each step k, we try to find ΔQ_k such that

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

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

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

If $\eta_k = 0$, we recover Newton's method.

During the first few Newton iterations after the start-up 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 non-linear system of equations which causes a waste of CPU time. This is known as oversolving [24]. Strategies for choosing a sequence of inner-tolerances that lead to an efficient rate of convergence of the inexact-Newton method without oversolving have been developed by several authors, e.g. [24]. For our applications, we found that using high inner-tolerances (i.e., not solving the linear system accurately) gives slightly better efficiency. We reduce the inner-residual by a factor of two for the first 10 outer-iterations. This guarantees no oversolving for most problems. We then reduce the inner-residual by a factor of 10 for the remaining iterations.

Linear iterative solver There are several effective iterative solvers for non-symmetric linear systems available, as reviewed by Dutto [25], Barrett et al. [26] and Page [27], among others. 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 CFD applications, preconditioned Krylov methods have shown better convergence properties than classical stationary methods such as Jacobi, Gauss-Seidel or SOR. Among Krylov solvers, GMRES, developed by Saad and Schultz [28], 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 bi-CGSTAB and CGS, we found GMRES faster for our applications.

The storage requirements of GMRES increase linearly and the CPU expense 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 restarted GMRES can have inferior convergence properties, it is generally essential to use a finite value of m in order to keep the memory requirements reasonable. Trade-offs 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. Since these can be approximated using finite-differences [29], 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. [12], 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 [30] that it can be advantageous from the performance point of view as well.

Preconditioner A weakness of iterative solvers, relative to direct solvers, is their lack of robustness. Preconditioning is an effective technique to improve both efficiency and robustness. It consists of transforming the linear system into one that is easier to solve by an iterative solver. Discussion of different types of preconditioners and the strategy adopted for PROBE will follow in a later section.

Ordering of unknowns Several authors have shown that the ordering of the unknowns plays an important role in the convergence of the iterative solver, especially when an incomplete factorization preconditioner is used [4, 31]. A number of ordering strategies have been compared in [32]. The Reverse Cuthill-McKee strategy [33], a well-known bandwidth reduction algorithm, was shown to be the most efficient of those studied, which included two natural orderings, two domain decomposition orderings, and the minimum neighbouring ordering [34].

Spatial discretization

The spatial discretization in PROBE is exactly the same as that used in ARC2D [23]. It consists of second-order centered-difference operators with the second and fourth-difference artificial dissipation model of Jameson et al. [35]. A far-field circulation correction is also included. The Baldwin-Lomax algebraic model [36] is used for turbulent flows. The steady-state solutions of PROBE are thus identical to those computed using ARC2D.

case	airfoil	Ma	α	Re	tr.up	tr.low
1	NAC	0.63	2.00	invisc	—	—
2	NAC	0.80	1.25	invisc	—	—
3	NAC	0.80	5.00	5.00e2	—	—
4	NAC	0.30	0.00	2.88e6	0.43c	0.43c
5	NAC	0.30	6.00	2.88e6	0.05c	0.80c
6	NAC	0.70	1.49	9.00e6	0.05c	0.05c
7	RAE	0.729	2.31	6.50e6	0.03c	0.03c

Table 1: Parameters for the seven flows studied. NAC refers to the NACA 0012 airfoil and RAE to the RAE 2822 airfoil.

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 used has 249×39 nodes with the off-wall spacing set to 2×10^{-3} chords. For the laminar case, the grid used has 249×49 nodes and an off-wall spacing of 5×10^{-4} chords. A 331×51 grid with the off-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 off-wall spacing. In some tests which include comparisons with multigrid, the number of nodes may vary slightly in order to be able to obtain coarser grids by removing every other point in the finer grid. These grids are typical of grids used in practice for such applications. Maximum cell aspect ratios for these grids range between 8.7×10^2 and 1.16×10^5.

Preconditioner

The linear system of algebraic equations obtained at each Newton iteration can be preconditioned by applying an iterative solver or using some sort of an incomplete factorization of the Jacobian matrix (which can also be considered to be a solver; see Ref. [20]). Iterative solvers have been widely used as preconditioners for GMRES. Following Wigton's [29] approach, most authors [37, 38, 39, 40] use existing solvers to precondition the Euler and Navier-Stokes equations. When the preconditioner does only one iteration per GMRES search direction, this approach is equivalent to the standard approach, i.e., solving the unpreconditioned equations with GMRES, using left preconditioning. Ajmani et al. [41] use the iterative solver as a right preconditioner. One of the advantages of using right preconditioning is that the residual of the preconditioned system is the same as the residual of the unpreconditioned system. This can be important since the stopping criterion should be based on the residual of the unpreconditioned system.

In general, iterative solvers used as preconditioners require less memory than incomplete factorization preconditioners. On the other hand, they can be computationally expensive since they must be applied at each inner iteration. Incomplete factorizations are more expensive to produce but need to be updated only once per outer iteration (or less often, as we shall see later). Applying such a preconditioner is relatively inexpensive. Hence this approach tends to be more effective if CPU time is the primary concern. Incomplete factorizations are further discussed in the next subsection.

A second issue arises when using a matrix-free implementation of GMRES. Since one need not form the exact Jacobian matrix explicitly, it is desirable to form the preconditioner from some simpler approximation to the Jacobian matrix which requires less storage. To date, few researchers have exploited the freedom available in defining the approximate Jacobian matrix on which the preconditioner is based.

Type of factorizations For our solver, we consider incomplete LU factorizations, which have shown good efficiency as preconditioners for Krylov solvers. There have been two distinct approaches to forming such incomplete factorizations: level of fill-in and threshold strategies. The first approach, named ILU(n), uses only the graph of the matrix to determine which entries to drop. In contrast, ILUT(p,τ), developed by Saad [42], 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 non-zeros 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 in two dimensions. That is why block versions of ILU(p) are popular in this type of application. As a matter of fact, in cases where a low level of fill-in is used, such as $p = 0$ or $p = 1$, 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)). Orkwis [43] reports than in his application, CGS (Conjugate Gradient Squared) did not converge with ILU(0) but it did converge with BFILU(0). Because of the storage format that we use namely Compressed Sparse Row, CSR [44], we find this approach more convenient than block ILU(p). ILUT(p,τ) does not have an equivalent BFILUT, since it already considers all the possible nonzeros in a row and keeps the larger ones.

With the level-of-fill approach, the nonzero structure of the incomplete factorization needs to be computed only once, since it is independent of the numerical values of the entries. Hence the preconditioner based on the level-of-fill approach is less expensive to form that that using thresholding. In addition, we have also shown that, for a given amount of fill, the BFILU factorization is more effective than the ILUT factorization in accelerating the convergence of GMRES [18, 32].

The level of fill p allowed in the factorization should be chosen taking into account storage considerations, the cost of forming the preconditioner, and its effectiveness in reducing the cost of solving the linear system. Figure 1 shows the CPU time required by preconditioned GMRES to reduce the inner-residual by two orders of magnitude for different levels of fill. An approximate functional-Jacobian that requires roughly half of the storage of the exact functional-Jacobian was used to form the preconditioner (to be discussed in the next subsection). 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, since it gives the best performance with a storage that is still below that of the exact Jacobian matrix.

Approximate Jacobian matrix 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 non-diagonally-dominant matrices such as the ones arising in our applications, the incomplete factors and can be more ill-conditioned than the original matrix, and the long recurrences associated with backward and forward solves may be unstable [45, 46, 47]. Since diagonal dominance tends to alleviate this problem [48], we can benefit by using an approximate Jacobian matrix with reduced off-diagonal dominance. This was shown in Ref. [18].

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

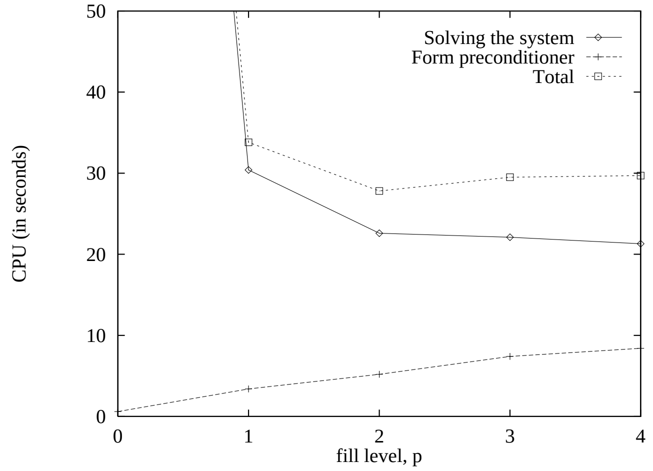

Figure 1: CPU cost of forming the BFILU(p) factorization and effectiveness in reducing the inner-residual by two orders of magnitude.

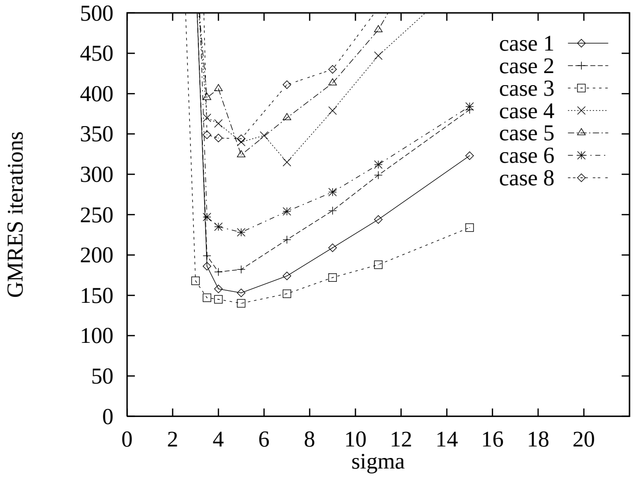

Figure 2: Total number of GMRES iterations required to converge to machine zero, for different values of σ for various cases.

coefficients on the right-hand side given by

$$\varepsilon_2^l = \varepsilon_2^r + \sigma\varepsilon_4^r \tag{6}$$

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.

A second approximation is introduced to enhance the diagonal dominance of the matrix by adding $1/\Delta t$ to the diagonal, Δt being a local time step given by,

$$\Delta t = \frac{\Delta t_0}{1+\sqrt{J}} \tag{7}$$

where J is defined to be the metric Jacobian of the generalized curvilinear coordinate transformations,

$$J = \frac{1}{x_\xi y_\eta - x_\eta y_\xi} \tag{8}$$

and Δt_0 is a constant left as a parameter.

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.

In order to determine appropriate values of σ and Δt_0, we have tried to minimize the sum of the GMRES iterations over the Newton iterations. This is equivalent to minimizing the CPU time to convergence. As we increase the value of σ, the diagonal dominance will benefit the conditioning of the factors. On the other hand, the matrix used to build the factors will be a less accurate representation of the matrix that we are preconditioning. Figure 2 shows that, because of these trade-offs, there is a value of σ that minimizes the amount of work to converge to machine zero. It is interesting to notice that $\sigma = 5$ is optimal or near optimal in all of the cases. 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 $\sigma \leq 3$.

Figure 3 shows that the influence of the second parameter, Δt_0, is, in general, much less beneficial than that of σ. Only cases 2 and 5 show some improvement in convergence when using a finite Δt_0. Values under 200 seriously affect the performance of the solver. Therefore we use a value of 10^4.

Freezing the factorization 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 which has strong initial nonlinearities, case 2, with and without updating the preconditioner. In Figure 4 we 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 Figure 5. This is related to our start-up 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 flow field do not degrade the effectiveness of the preconditioner.

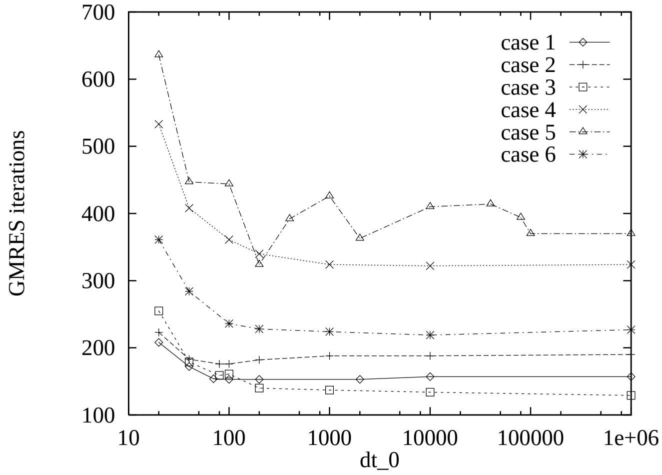

Figure 3: Total number of GMRES iterations required to converge to machine zero, for different values of dt_0 for cases 1 to 6.

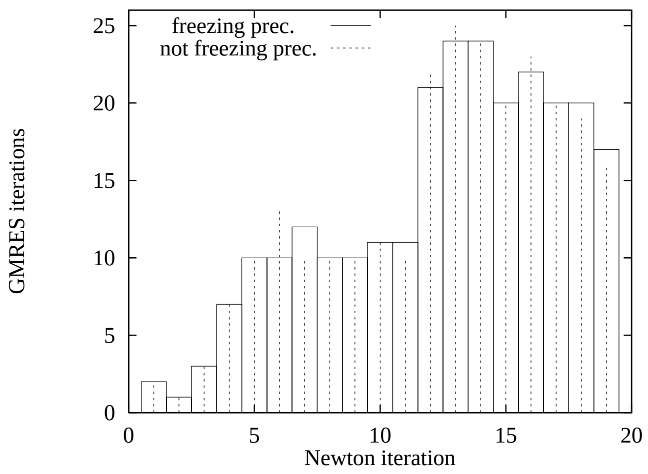

Figure 4: Number of GMRES iterations at each Newton iteration freezing the preconditioner (bars) and updating the preconditioner at each Newton iteration (impulses) for case 2.

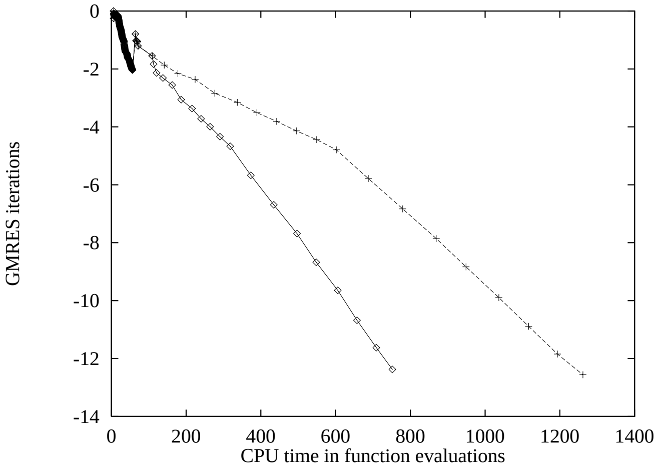

Figure 5: Convergence history for case 2, freezing the preconditioner and updating the preconditioner at each Newton iteration.

Convergence results

The convergence histories for the seven cases are shown in Figure 6 as a function of the CPU time normalized by the CPU time required for one function evaluation. Reduction of the residual by 12 orders of magnitude is achieved in 500 to 1000 function evaluations for all of the cases. Table 2 shows some statistics for PROBE, including the number of inner and outer iterations required to reduce the residual norm by twelve orders of magnitude. The outer iterations include only those done using the Newton-Krylov solver and not those of the approximately-factored algorithm. CPU/f.e. gives the total run time normalized by the CPU time of a function evaluation. The CPU expense per inner iteration is seen to be less than the equivalent of four function evaluations. Application of the BFILU(2) preconditioner requires the equivalent of roughly one function evaluation. The use of a solver as a preconditioner would almost certainly require a substantially greater cost and thus can only be efficient if the total number of inner iterations can be correspondingly reduced using this approach.

Another important consideration is how the CPU time scales with the size of the problem. To this end, we have tested case 1 using several different grids. The results are plotted in Figure 7. We can approximate the points in the plot by the expression

$$\omega = \kappa N^{\beta} \tag{9}$$

where ω is the CPU time in function evaluations, and κ and β are constants. Ideally β should be zero, which would mean that the CPU cost would be linear with respect to the grid size, since the cost of a function evaluation varies linearly with N. For PROBE, $\beta = 0.325$.

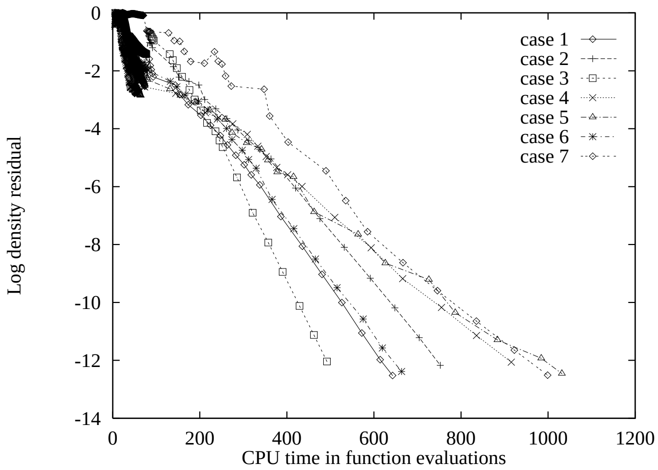

Figure 6: Residual history for PROBE for the 7 cases.

	o-it	Σi-it	i-it/o-it	CPU/f.e.
Case 1	18	157	8.7	615.5
Case 2	18	190	10.6	743.8
Case 3	18	129	7.2	490.7
Case 4	17	324	19.1	910.1
Case 5	19	370	19.5	990.8
Case 6	18	227	12.6	642.9
Case 7	22	354	16.1	953.5

Table 2: Statistics for the Newton-Krylov algorithm for the cases studied; **o-it**: outer iterations, **Σi-it**: total number of inner iterations, **i-it/o-it**: average number of inner iterations per outer iteration; **CPU/f.e.**: CPU time in equivalent function calls to reduce the residual by twelve orders of magnitude.

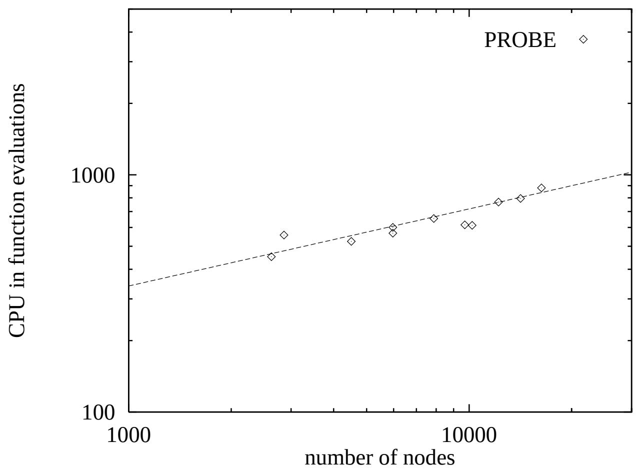

Figure 7: CPU time in function evaluations required to converge to machine zero for case 1 as a function of grid size.

Thus, the scalability of PROBE appears to be reasonably good.

Comparison with multigrid

In this section, we compare the performance of PROBE with ARC2D-MG, which uses the approximately-factored algorithm of ARC2D accelerated with multigrid. PROBE and ARC2D-MG produce identical steady-state solutions. In ARC2D-MG, we use a three-level sawtooth cycle with four iterations of the approximately-factored algorithm at each grid level in the cycle. This approach produces optimum convergence in terms of CPU time [19].

The residual convergence histories of PROBE are compared with ARC2D-MG in Figures 8 and 9. In general, PROBE is significantly faster. 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.

Since we measure the CPU time in terms of function evaluations, it is useful to define the convergence rate as the reduction in the residual obtained in the CPU time required for one function evaluation. This provides a convergence rate based on CPU time rather than iterations or multigrid cycles. With this definition, PROBE produces an average convergence rate of between 0.945 and 0.972, based on the data in Table 2. For comparison, a multigrid method with a convergence rate of 0.75 *per multigrid cycle* with a cost per multigrid cycle equal to that of five function evaluations, for example, produces a convergence rate *per function evaluation* of $0.75^{\frac{1}{5}} = 0.944$. 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.

In order to have an idea of the relative memory needs of the solvers that we are considering, we use case 6, comparing the amount of resident memory that the codes occupy in the computer. This is not a perfect measure, since it depends on many factors foreign to the algorithm, but it serves as a reference. Our test shows that PROBE requires 3.5 times more memory than ARC2D-MG. This value can be reduced through a more careful allocation of memory on our part. Nevertheless, for two-dimensional calculations, this does not constitute a problem.

Conclusions

We have presented and justified the basic components of PROBE, a Newton-Krylov solver for aerodynamic flows.

In particular, the preconditioner used to improve the efficiency of the linear solver has been described in detail. For a range of flows, the CPU time required for a residual reduction of twelve orders of magnitude is equivalent to the cost of 500 to 1000 function evaluations. This corresponds to a convergence rate per function evaluation between 0.945 and 0.972.

Acknowledgments

The first author was supported by an OGS grant 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, January 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. Venkatakrishnan, V., and Mavriplis, D.J., "Implicit Solvers for Unstructured Meshes." AIAA Paper 91-1537, June 1991.

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." Thèse de Doctorat, Université Pierre et Marie Curie, Paris VI, France, 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 Fuids," *Comput. Methods Appl. Mech. Engrg.*, vol. 87, pp. 281–304, 1991.

7. Ajmani, K., "Preconditioned Conjugate Gradient Methods for the Navier-Stokes Equations." Ph.D. thesis, Virginia Polytechnic Institute and State University, 1991.

8. Degrez, G., and Issman, E., "Acceleration of Compressible Flow Solvers by Krylov Subspace Methods." Lecture Notes For The Von Karman Institute For Fluid Dynamics Lecture Series 1994-05, 1994.

9. 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, January 1994.

10. Rogers, S.E., "A Comparison of Implicit Schemes for the Incompressible Navier-Stokes Equations with Artificial Compressibility." AIAA Paper 95-0567, January 1995.

11. Forsyth, P.A., and Jiang, H., "Iterative Methods for Full Newton Solution of the Euler Equations." Sixth International Symposium on Computational Fluid Dynamics, pp. 318–323, September 1995.

12. 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.

13. Barth, T.J., and Linton, S.W., "An unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation." AIAA Paper 95-0221, January 1995.

14. Cai, X., Keyes, D.E., and Venkatakrishnan, V., "Newton-Krylov-Schwarz: an Implicit Solver for CFD," Tech. Rep. No. 95-87, ICASE, December 1995.

15. 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.

16. Delanaye, M., Geuzaine, Ph., 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 CFD Methods and Algorithms, Seville, Spain, 2-5 October, 1995.

17. Blanco, M., and Zingg, D.W., "A Fast Solver for the Euler Equations on Unstructured Grids Using a Newton-GMRES Method." AIAA Paper 97-0331, January 1997.

18. Pueyo, A., and Zingg, D.W., "Progress in Newton-Krylov Methods for Aerodynamic Calculations." AIAA Paper 97-0877, January 1997.

19. Chisholm, T., and Zingg, D.W., "Multigrid Acceleration of an Approximately Factored Algorithm for Aerodynamic Flows." 44th Annual Conference of the Canadian Aeronautics and Space Institute, April 1997.

20. Pueyo, A., and Zingg, D.W., "An Efficient Newton-GMRES Solver for Aerodynamic Computations." AIAA Paper 97-1955, June 1997.

21. McHugh, P.R., and Knoll, D.A., "Comparison of Standard and Matrix-Free Implementations of Several Newton-Krylov Solvers," *AIAA J.*, vol. 12, pp. 2394–2400, December 1994.

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

23. Pulliam, T.H., "Efficient Solution Methods for the Navier-Stokes Equations." Lecture Notes For The Von Karman Institute For Fluid Dynamics Lecture Series, January 1986.

24. Eisenstat, S.C., and Walker, H.F., "Choosing the Forcing Terms in an Inexact Newton Method," *SIAM J. Sci. Comput.*, vol. 17, pp. 16–32, January 1996.

25. Dutto, L.C., "On Iterative Methods for Solving Linear Systems of Equations," *Revue européenne des éléments finis*, vol. 2, 1993.

26. Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H., "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods," *SIAM*, 1993.

27. Page, M., "Étude de Méthodes Itératives pour la Résolution des Équations de Navier-Stokes." Thèse de doctorat (Ph.D.), École Polytechnique de Montréal, Québec, Canada, June 1995.

28. Saad, Y., and Schultz, M.H., "GMRES: A Generalized Minimal Residual Algorithm For Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, July 1986.

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

30. Pueyo, A. and Zingg, D.W., "Airfoil Computations Using a Newton-GMRES Method." CFD96, Proceedings of the Fourth Annual Conference of the CFD Society of Canada, June 1996.

31. 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, pp. 457–497, 1993.

32. Pueyo, A., "An Efficient Newton-Krylov Method for the Euler and Navier-Stokes Equations." Ph. D. Thesis, Institute for Aerospace Studies, University of Toronto, 1998.

33. Cuthill, E.H., and McKee, J.M., "Reducing the bandwidth of sparse symmetric matrices." Proc. 24th National Conference of the Association for Computing Machinery, Brondon Press.

34. Martin, G., "Méthodes des préconditionnement par factorisation incomplète." Mémoire de Maîtrise, Université de Laval, Québec, Canada, 1991.

35. Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes." AIAA Paper 81-1259, June 1981.

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

37. Hixon, R., and Sankar, L.N., "Application of a Generalized Minimal Residual Method to 2D Unsteady Flows." AIAA Paper 92-0422, January 1992.

38. Hager, J.O., and Lee, K.D., "The Behavior of Some Solution Acceleration Techniques in CFD." AIAA Paper 94-0175, January 1994.

39. Ollivier-Gooch, C.F., "Towards Problem Independent Multigrid Convergence Rates For Unstructured Mesh Methods I: Inviscid and Laminar Viscous Flows." In Proceedings of the Sixth International Symposium on Computational Fluid Dynamics, September 1995.

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

41. Ajmani, K., Ng, W., Liou, M., "Preconditioned Conjugate Gradien Methods for the Navier-Stokes Equations," *J. Comp. Phys.*, vol. 110, pp. 68–81, 1994.

42. Saad, Y., "ILUT: A Dual Threshold Incomplete LU Factorization," Tech. Rep. UMSI 92/38, University of Minnesota Supercomputer Institute, March 1992.

43. Orkwis, P.D., "Comparison of Newton's and Quasi-Newton's Method Solvers for the Navier-Stokes Equations," *AIAA J.*, vol. 31, pp. 832–836, May 1993.

44. Saad, Y., "SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations, version 2," June 1994.

45. Chow, E., and Saad, Y., “Experimental Study of ILU Preconditioners for Indefinite Matrices,” Tech. Rep. UMSI 97/95, University of Minnesota Supercomputing Institute Research, June 1997.

46. Elman, H.C., “A Stability Analysis of Incomplete LU Factorizations,” *Math. Comp.*, no. 47, pp. 191–217, 1986.

47. Bruaset,, A.M., Tveito, A., and Winther, R., “On the Stability of Relaxed Incomplete LU Factorizations,” *Math. Comp.*, no. 54, pp. 701–719, 1990.

48. Saad, Y., “Krylov Subspace Techniques, Conjugate Gradients, Preconditioning and Sparse Matrix Solvers.” 1994-05, von Karman Institute for Fluid Dynamics, March 1994.

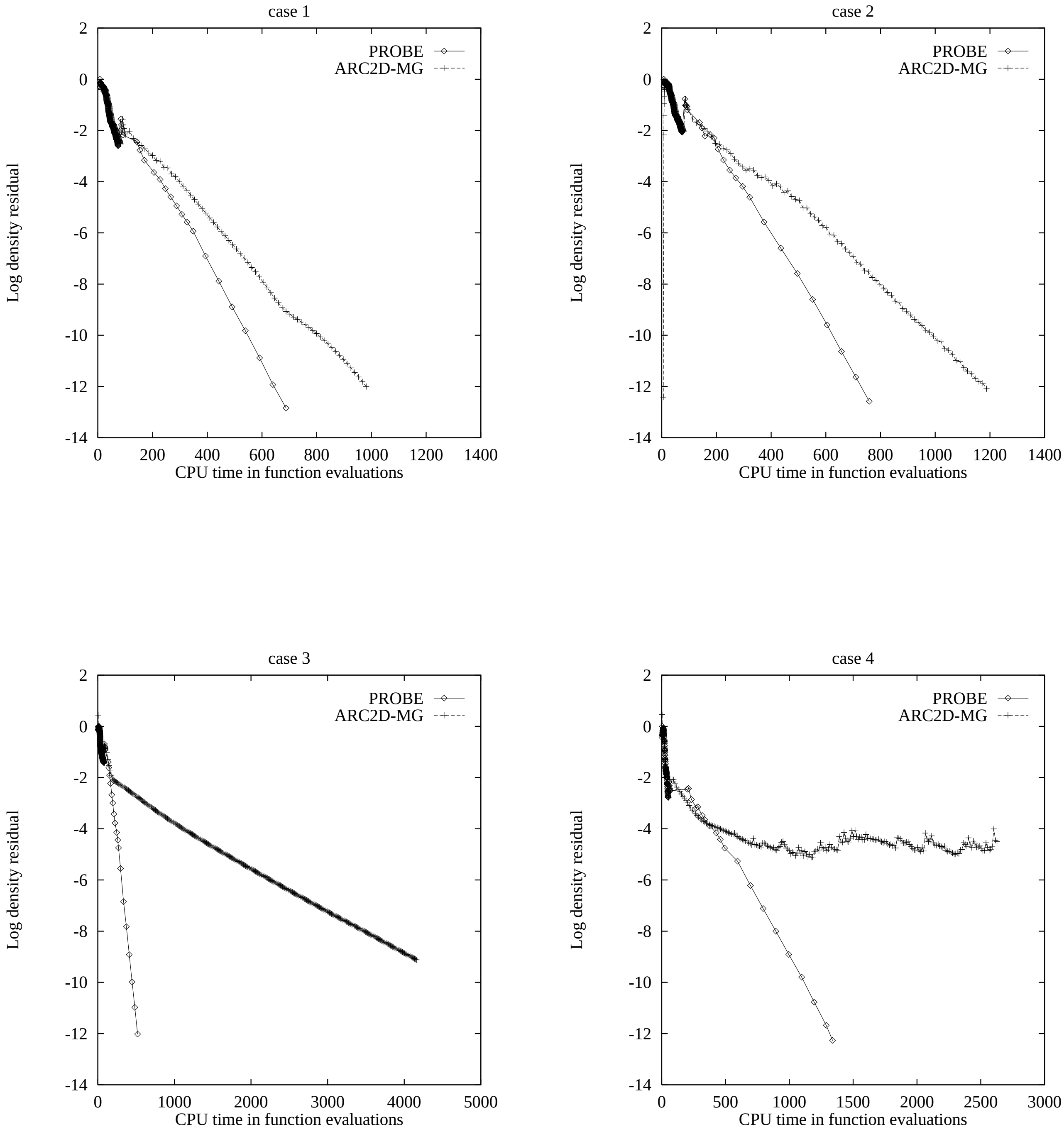

Figure 8: Cases 1 to 4: convergence history for the inexact-Newton-Krylov method (PROBE) and the approximately-factored method with three-level multigrid (ARC2D-MG).

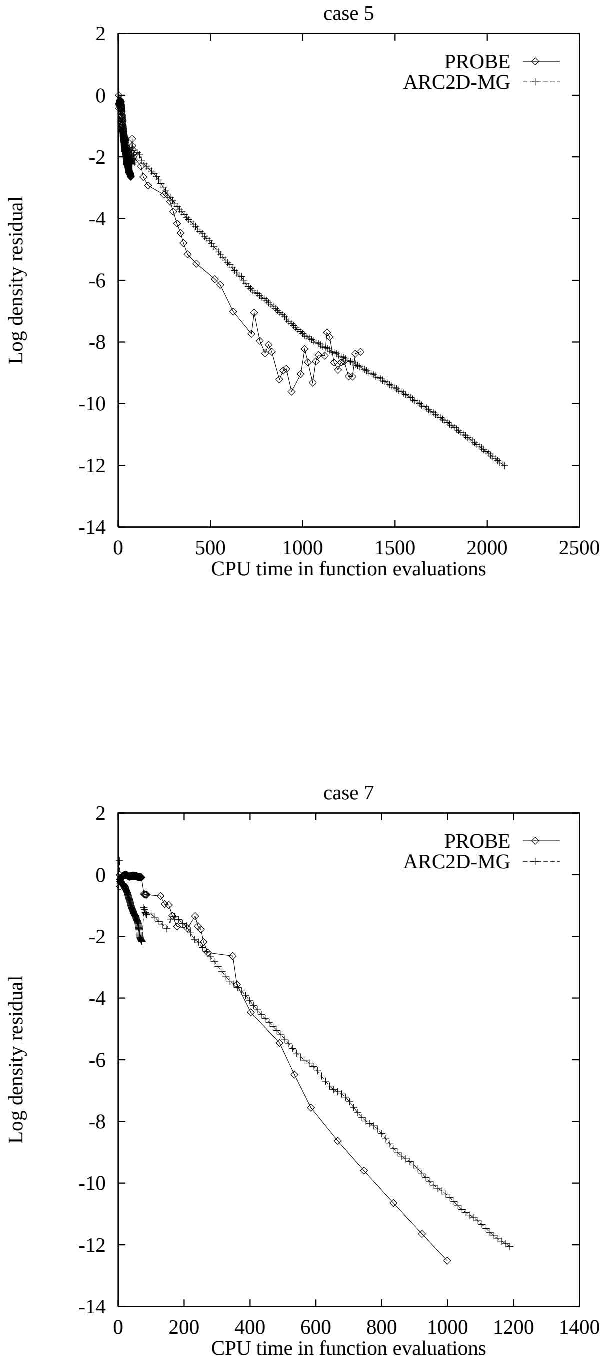

Figure 9: Cases 5 to 7: convergence history for the inexact-Newton-Krylov method (PROBE) and the approximately-factored method with three-level multigrid (ARC2D-MG).