Journal of Computational Physics 228 (2009) 3490–3507

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A Jacobian-free Newton–Krylov algorithm for compressible turbulent fluid flows

Todd T. Chisholm, David W. Zingg *

University of Toronto, Institute for Aerospace Studies, 4925 Dufferin St., Toronto, Ontario, Canada M3H 5T6

ARTICLE INFO

Article history:
Received 13 October 2008
Received in revised form 1 February 2009
Accepted 3 February 2009
Available online 12 February 2009

PACS:
47.11.−j

Keywords:
Newton–Krylov methods
Turbulent flow
Aerodynamic flows

ABSTRACT

Despite becoming increasingly popular in many branches of computational physics, Jacobian-free Newton–Krylov (JFNK) methods have not become the approach of choice in the solution of the compressible Navier–Stokes equations for turbulent aerodynamic flows. To a degree, this is related to some subtle aspects of JFNK methods that are not well understood, and, if poorly handled, can lead to inefficient and unreliable performance. These are described here, along with strategies for addressing them, leading to an efficient JFNK algorithm for turbulent aerodynamic flows applicable to multi-block structured grids and a one-equation turbulence model. Development of globalization strategies for field-equation turbulence models represents one of the key contributions of the paper. Numerous examples of subsonic and transonic flows over single and multi-element airfoils are presented in order to demonstrate the efficiency and reliability of the algorithm. In addition, a number of guidelines are presented to aid in diagnosing problems with JFNK algorithms.

1. Introduction

Jacobian-free Newton–Krylov (JFNK) methods are becoming increasingly popular in many branches of computational physics, as described in the survey paper by Knoll and Keyes [1]. They cite numerous examples in fluid dynamics, plasma physics, reactive flows, flows with phase change, radiation diffusion, radiation hydrodynamics, and geophysical flows. It is interesting to note, however, that JFNK methods have not become the approach of choice in the numerical solution of the compressible Navier–Stokes equations in computational fluid dynamics and aerodynamics. Of the papers cited by Knoll and Keyes in which JFNK methods have been applied to the Euler and compressible Navier–Stokes equations, roughly half are associated with the efforts of a single group during the mid-1990s (e.g. [2]). Of the remaining papers, only a few represent continuing research. The major NASA flow solvers typify the algorithms that are most popular in the computational aerodynamics community. OVERFLOW [3] and CFL3D [4] are both based on implicit approximate factorization methods, TLNS3D [5] and CART3D [6] utilize multi-stage explicit schemes with multigrid, and FUN3D [7], although it includes a Newton–Krylov option, is usually run using either a point implicit procedure or an implicit line relaxation scheme [8]. All of these approaches are more mature than JFNK methods.

The lack of broad acceptance of JFNK methods in the computational fluid dynamics community stems from several factors. Although the Jacobian matrix is not required, some sort of matrix is typically formed in order to precondition the linear system. Together with the Krylov subspace, this can lead to higher memory use than some of the more popular methods. In

* Corresponding author.
E-mail address: dwz@oddjob.utias.utoronto.ca (D.W. Zingg).
URL: http://goldfinger.utias.utoronto.ca/dwz (D.W. Zingg).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.02.004

addition, formation of this matrix can require additional programming effort, e.g. hand linearization of the discrete residual equations. Completely matrix-free algorithms avoid these issues by using a solver as a preconditioner [9] but are typically slower and often inherit the shortcomings of the solver used. Moreover, a JFNK method often involves a number of parameters, and there is some effort involved in ensuring that suitable values are chosen for specific problem classes (e.g. [10]). Poorly chosen parameter values can lead to inefficient and unreliable algorithms. Other algorithms require some parameter selection as well, but often somewhat fewer, and optimal values for specific problem classes are better established than for the relatively newer JFNK algorithms. Furthermore, efficient globalization is sometimes difficult to achieve; the highly nonlinear behavior of some field-equation turbulence models can be particularly problematic. Finally, there are some subtle aspects of Newton–Krylov methods that are not often reported and can also lead to inefficient and unreliable performance if handled improperly.

The above issues notwithstanding, JFNK methods also offer many compelling potential advantages in the solution of compressible flows. They can be the most efficient option for extremely stiff problems, where stiffness can be introduced by multiple scales associated with complex physics, such as chemical reactions [11], or stiff source terms, such as those introduced by some turbulence models. In addition, the Newton-like convergence properties of JFNK methods are ideal when deep convergence is needed. Consequently, they are very effective in the context of aerodynamic shape optimization, as demonstrated by Nemec and Zingg [12]. Moreover, their convergence tends to be insensitive to the properties of the mesh. For example, JFNK methods converge well on meshes with high aspect ratios [13]. Also, high-order methods can lead to reduced stability bounds for explicit iterative methods; hence JFNK methods can be appropriate in this context [14]. Furthermore, JFNK methods can be a very efficient means of solving the nonlinear problem that arises at each time step of a time-accurate implicit algorithm. Isono and Zingg [15] demonstrated that a JFNK algorithm is much more efficient than an approximate factorization algorithm for implicit time-accurate computations. Finally, the parameters involved in JFNK methods can be used to advantage to tune the algorithm to be very efficient for specific problem classes or to adjust the algorithm for particularly difficult problems.

The present paper extends the work of Pueyo and Zingg [13], who developed a JFNK algorithm for the well established solver ARC2D [16], which is based on the same underlying algorithms as OVERFLOW. Pueyo and Zingg demonstrated that their Newton–Krylov algorithm converges much more rapidly in terms of computing time than the original approximate factorization algorithm in ARC2D (and OVERFLOW). They plotted convergence histories in terms of equivalent residual evaluations, determined by dividing the total computing time by the computing time needed for a single evaluation of the residual. In most cases, the residual was reduced by twelve orders of magnitude in fewer than 1000 equivalent residual evaluations. Closely related algorithms were applied to two-dimensional unstructured meshes by Blanco and Zingg [17,18], three-dimensional unstructured meshes by Wong and Zingg [19], three-dimensional inviscid flows on structured meshes by Nichols and Zingg [20], parallel computations of three-dimensional inviscid flows on structured meshes by Hicken and Zingg [21] and shape optimization based on steady [12] and unsteady [22] flows. Although the work of Pueyo and Zingg provided a solid foundation for further development of Newton–Krylov methods for the compressible Reynolds-averaged Navier–Stokes equations, it was restricted in several respects. In particular, their algorithm was limited to two-dimensions, single-block structured meshes, scalar numerical dissipation, and an algebraic turbulence model.

The original goal of the present research was to extend the Newton–Krylov algorithm of Pueyo and Zingg to multi-block structured grids, matrix numerical dissipation, and a field-equation turbulence model. In doing so, a number of interesting and subtle aspects of Newton–Krylov algorithms, which are of general relevance, were encountered and addressed. Hence the objectives of the present paper are as follows:

- To present an efficient JFNK solver for turbulent aerodynamic flows applicable to multi-block structured grids, matrix numerical dissipation, and a field-equation turbulence model.
- To describe some subtle aspects of JFNK algorithms which can greatly affect performance and how to address them.
- To present strategies for identifying the causes of problems with JFNK algorithms.

Full details are available in the thesis of Chisholm [23]. Although the present study is performed in the context of a specific spatial discretization and turbulence model, the ideas presented and conclusions drawn are relevant to JFNK methods in conjunction with a wide range of spatial schemes and turbulence models.

2. Governing equations, turbulence model, and spatial discretization

The governing equations are the two-dimensional, thin-layer, compressible Navier–Stokes equations written in conservative form [16]. They are nondimensionalized based on free stream variable values following Pulliam [16], where the velocity scale used is the free stream speed of sound, and the length scale is, for example, the chord length of an airfoil. The equations are transformed to generalized curvilinear coordinates in order to facilitate the application of finite-difference formulas to the spatial derivatives [16]. The thin-layer approximation neglects all streamwise derivatives in the viscous terms. Turbulence effects are incorporated through an eddy viscosity term computed by solving the partial differential equation defining the turbulence model developed by Spalart and Allmaras [24]. The model is in the form of a single convection–diffusion equation with source terms representing production and destruction. The dependent variable of the turbulence mod-

el, $\tilde{\nu}$, is related to the kinematic eddy viscosity nondimensionalized by the free stream viscosity. Trip terms are included to ensure that laminar-turbulent transition occurs at specified locations. The minor modifications proposed by Ashford [25] are also incorporated in the turbulence model. At a solid surface, standard no-slip boundary conditions are applied. Adiabatic walls are assumed, i.e. zero normal temperature gradient, and a zero normal pressure gradient is also enforced at solid boundaries. At far-field boundaries, Riemann invariants are used to prescribe the boundary conditions, and a circulation correction is included to reduce the effect of artificially truncating the flow domain [16]. At a solid surface, the turbulence variable, $\tilde{\nu}$, is set to zero. At a far-field inflow boundary, it is set to 0.01.

The inviscid and viscous flux derivatives are approximated using second-order centered difference formulas applied in the uniform computational space. First-order upwind differencing is used for the convective terms in the turbulence model equation. Numerical dissipation is added to the inviscid flux discretization through either the scalar dissipation terms described by Pulliam [16] or the matrix dissipation approach of Swanson and Turkel [26]. In the scalar model, the matrix used in the matrix model is replaced by the spectral radius of the flux Jacobian. Hence the scalar model is more dissipative. This is most noticeable in boundary layers, where the matrix model is more accurate for a given grid density. Both dissipation models include a combination of first-order, second-difference dissipation and third-order, fourth-difference dissipation. The amount of first-order dissipation is controlled by a pressure switch designed to sense shock waves. The pressure switch includes both the absolute value and maximum functions, and is therefore not strictly differentiable. Both dissipation models also involve the absolute value function. The fourth-difference operator requires a five-point stencil in each coordinate direction. The coefficient of the fourth-difference dissipation is set to 0.02 for all cases presented here. The second-difference dissipation coefficient is zero for subsonic flows and unity for transonic flows. The matrix dissipation model includes two parameters to avoid zero eigenvalues. The lower bound on the magnitude of the convective eigenvalues is set equal to 0.025 times the spectral radius of the relevant flux Jacobian. The lower bound on the sound speed eigenvalues is set to 0.025 times the spectral radius for subsonic flows and to 0.25 times the spectral radius for transonic flows. These are typical parameter values for these schemes. A complete description of the spatial discretization can be found in Zingg et al. [27]. The accuracy of this spatial discretization has been studied extensively (e.g. [27]), and the grids used here are such that numerical errors are small and the numerical solutions compare well with experimental data (e.g. [28,29]).

3. Algorithm

3.1. Solving the nonlinear system – the outer iterations

After discretizing in space, the governing partial differential equations are reduced to ordinary differential equations in the form

$$\frac{dQ}{dt} = R(Q). \tag{1}$$

Here Q is the vector of conservative variables, including the turbulence variable, at each node of the grid, including the boundary nodes. The vector R includes the discrete residual at each interior node plus the discretized boundary conditions. The equations corresponding to the boundary conditions are included in this system, but for these equations the time derivative is zero, i.e. they are algebraic equations, not differential equations. As a result of the transformation to curvilinear coordinates, the local conservative variables are multiplied by the inverse of the Jacobian of the transformation, J^{-1}, but the turbulence parameter is not. The inverse Jacobian is closely related to the cell area, and hence can vary widely throughout the grid. Similarly, the mass, momentum, and energy equations are multiplied by J^{-1} during the transformation, but the turbulence model equation is not, as it is solved in nonconservative form. The boundary conditions are also not scaled by J^{-1}. As a result of the presence or absence of J^{-1}, there is significant variation in the magnitudes of the entries of both Q and R. For steady flows we seek the solution to the nonlinear algebraic system of equations

$$R(Q) = 0, \tag{2}$$

but we retain the time derivative term for use in globalizing the algorithm.

Applying the implicit Euler time-marching method with local time linearization to Eq. (1), one obtains, for the nth iteration,

$$\left(A_n - \frac{I}{\Delta t}\right)\Delta Q_n = -R(Q_n), \tag{3}$$

where A is the Jacobian matrix of $R(Q)$ given by

$$A = \frac{\partial R(Q)}{\partial Q}. \tag{4}$$

The term $\frac{I}{\Delta t}$ is a simplified notation in that this term is zero for the boundary conditions, and the time step can vary spatially. As $\Delta t \to \infty$, Newton's method is recovered:

$$A_n \Delta Q_n = -R(Q_n). \tag{5}$$

At each iteration, the linear system given by either Eq. (3) or Eq. (5) must be solved. With a fixed time step, the implicit Euler method converges linearly, while Newton's method converges quadratically if certain conditions are satisfied.

In an inexact-Newton method, Eq. (5) is not solved exactly. Such a method can be written as

$$\|R(Q_n) + A_n \Delta Q_n\|_2 \leqslant \eta_n \|R(Q_n)\|_2, \tag{6}$$

where the parameter $\eta_n \in [0, 1)$ controls the degree of convergence of the linear system at each iteration. It is convenient to use the L_2 norm, since it may be provided by the linear iterative solver at no extra cost. Superlinear and even quadratic convergence can be obtained by a properly chosen sequence of values of η_n [30]. However, such a sequence is typically not optimal in terms of minimizing computing time. It is important to avoid oversolving the linear system, and a constant value of η_n equal to 0.1 is often the most efficient choice.

Since Newton's method is not globally convergent, a globalization strategy is needed to bring the initial iterate into the radius of convergence of Newton's method. Knoll and Keyes [1] discuss several strategies, including line search and trust region methods, pseudo-transient continuation, and continuation methods. Since the nonlinear algebraic system of equations of interest here is the steady solution of a system of ordinary differential equations, we consider only pseudo-transient continuation. This requires the choice of a time step sequence in Eq. (3), where Δt is replaced by Δt_n. Since we are not concerned with a physically relevant transient, Δt_n can vary spatially and can also vary among equations. For example, the time step in the turbulence model equation need not be equal to that in the mean-flow equations. The time step strategy is further discussed in Section 4.

3.2. Solving the linear system – the inner iterations

The linear systems arising from Eqs. (3) and (5) are typically large, sparse, nonsymmetric, and ill-conditioned. For such problems, the appropriate iterative method is dependent on several factors, including the properties of the matrix and the degree of convergence needed. Here we consider only the generalized minimal residual method (GMRES) [31] preconditioned using incomplete lower–upper preconditioning with some fill, based on a level of fill strategy (ILU(p)) [32]. GMRES forms an orthonormal basis from the Krylov subspace using Arnoldi's method and finds the iterate that minimizes the residual using this basis. In order to limit the memory use associated with the Krylov vectors, it is common to restart GMRES after a set number of iterations. This can be highly detrimental to convergence and hence is not recommended unless absolutely necessary. For the loose tolerance used here ($\eta_n = 0.1$), the size of the Krylov subspace is rarely excessive. For the examples shown later, we have limited the number of GMRES iterations and hence the subspace size to 30. If this limit is reached, rather than restarting GMRES, we terminate and move to the next outer iteration. This limit was reached only for the computation of the three-element configuration. For the single-element airfoils, the average number of inner iterations per outer iteration is roughly 13 with a maximum of 16. For the two-element configuration, the average is 22 and the maximum is 28.

GMRES requires only matrix–vector products. Hence the Jacobian matrix need be neither explicitly formed nor stored. Instead, the product of the Jacobian and an arbitrary vector v can be approximated by a first-order forward difference expression:

$$Av \approx \frac{R(Q + \epsilon v) - R(Q)}{\epsilon}, \tag{7}$$

where ϵ is a small scalar parameter. It is immediately evident that the calculation can be prone to round-off error in the addition and the subtraction if ϵ is too small. Conversely, if ϵ is too large, the error in the finite-difference approximation can become excessive. As a result, choosing a value of ϵ is a tricky balance between round-off and truncation error. If the variables or equations are not well scaled, then finding a suitable value of ϵ can be particularly challenging, if not impossible. Occasionally it can be advantageous to utilize a second-order finite-difference expression, but this should generally be avoided due to the extra cost of the second residual function evaluation. The subject of scaling and the choice of ϵ is further discussed in Section 4.

It is often incorrectly assumed that the Jacobian-free matrix–vector products are inherently advantageous in terms of computing time. If a large number of inner iterations is needed, then it can actually be faster to form the Jacobian matrix and apply it repeatedly rather than performing repeated evaluations of the residual vector. However, the Jacobian-free approach is usually favored as a result of memory considerations. For the applications presented here, the number of inner iterations is typically well below the threshold above which the Jacobian-free approach becomes less efficient.

The ordering of the boundary condition equations at a given boundary node is arbitrary. Since it is typically beneficial to avoid small pivots, the boundary equations should be reordered to maximize the diagonal elements. This operation needs to be performed at the first iteration only.

3.3. Preconditioning

The rate of convergence of a Krylov method such as GMRES is strongly dependent on the conditioning of the matrix. The Jacobian matrices resulting from the discretized Navier–Stokes equations are typically highly ill-conditioned, such that GMRES will converge very poorly unless the system is preconditioned. For example, right preconditioning can be written as

$$(AM^{-1})(Mx) = b. \tag{8}$$

Thus, the convergence of GMRES depends on the conditioning of the matrix AM^{-1} rather than A, and M should be chosen to cluster the eigenvalues of AM^{-1}. Therefore, we would like M^{-1} to be as close to A^{-1} as possible. This can be accomplished by forming an approximation to the inverse of the Jacobian matrix or by applying one or more iterations of an iterative method, such as GMRES (leading to the nested GMRES method [33]) or a relaxation method. Although these two approaches are closely related [34], the latter can be implemented such that it is fully matrix-free, which greatly reduces memory use, but necessitates the use of flexible GMRES [35], which somewhat increases memory use. Although this approach can lead to good performance in some contexts [35], we concentrate here on a specific approximate inverse, the incomplete lower–upper (ILU) factorization. For further discussion of parallel, multigrid, and physics-based preconditioners, see Knoll and Keyes [1]. We note in particular that success has been achieved using multigrid preconditioning, and this may be added to our algorithm in future.

In an ILU factorization, an approximation to the LU factorization of a matrix is obtained by dropping elements according to some rule as the factorization proceeds. In the simplest case, known as ILU(0), only elements that have a corresponding entry in the original matrix are kept. Since such a preconditioner is often inadequate, strategies have been developed to introduce additional fill into the incomplete factorization, including a level-of-fill strategy, ILU(p), and a threshold strategy, ILUT(P,τ) [32]. In the level-of-fill approach, each element that is introduced in the factorization is assigned a level. If it has a corresponding entry in the original matrix, then it is assigned a level of zero. Otherwise, the level assigned is the sum of the levels of the two elements that contributed to the formation of the new element, plus one. If the new element's level of fill exceeds p, then it is dropped. This approach is based solely on the graph of the matrix and thus can be precomputed. In the threshold strategy, ILUT(P,τ), elements are dropped if they are smaller than τ, and at most the P largest elements are kept in each row. Although the ILUT(P,τ) strategy allows more precise control, it is much more expensive to form, and both Pueyo and Zingg [13] and Wong and Zingg [19] have demonstrated that the ILU(p) strategy is much more efficient in the current context.

The Jacobian matrix is naturally organized into blocks, which are 5×5 in the present case. Some blocks contain a mix of zero and nonzero entries. This can prove problematic for a scalar ILU algorithm. One solution, called BFILU [36,13], is to assign a level of fill of zero to the zero entries within such a block, such that nonzero entries are permitted in these locations in the ILU factorization. The method used here, BILU, treats each block as a single entry. Hence division operators become block matrix inversions.

Given that the Jacobian-free approach avoids the need to form and store the Jacobian matrix, it is natural to consider basing the preconditioner on a reduced-storage approximate Jacobian. The matrix–vector products needed by the GMRES algorithm with right preconditioning can then be written as

$$AM_1^{-1}v \approx \frac{R(Q + \epsilon M_1^{-1}v) - R(Q)}{\epsilon}, \tag{9}$$

where M_1^{-1} is a preconditioner based on an approximate Jacobian. With an upwind spatial discretization, the approximate Jacobian is normally constructed from a first-order upwind discretization, which involves nearest neighbours only. With the present spatial discretization, next to nearest neighbours are introduced solely as a result of the fourth-difference artificial dissipation. Therefore, a reduced-storage Jacobian can be constructed by neglecting fourth-difference dissipation and adding additional second-difference dissipation to compensate. Pueyo and Zingg [13] introduced the following formula:

$$\epsilon_2^M = \epsilon_2^R + \sigma\epsilon_4^R, \tag{10}$$

where ϵ_2^M is the coefficient of second-difference dissipation in the approximate Jacobian used to form the ILU preconditioner, ϵ_2^R is the coefficient of second-difference dissipation in the residual, ϵ_4^R is the coefficient of fourth-difference dissipation in the residual, and σ is a parameter. A value of $\sigma = 5$ is typically optimal with scalar dissipation, while $\sigma = 10$ is preferred with matrix dissipation. Nonlocal terms, such as the circulation correction, are also excluded from the approximate Jacobian. Numerical experiments demonstrate that the absence of these nonlocal terms from the approximate Jacobian used to form the preconditioner does not significantly affect convergence.

Although the original motivation for using an approximate Jacobian matrix to form the ILU preconditioner was memory savings, Pueyo and Zingg [13] showed that this also leads to a much more effective preconditioner. This has also been observed when a first-order upwind discretization is used in the formation of the preconditioner [1]. It is important to recognize that we are trying to minimize the difference $AM^{-1} - I$ rather than the difference $A - M$. The former difference can be very large if small pivots are encountered during the incomplete factorization, or the triangular solves are unstable [37]. The approximate Jacobian matrix has larger diagonal entries and consequently avoids these difficulties, which can greatly reduce the effectiveness of the preconditioner.

Returning to Eq. (9), we see that in ILU-preconditioned Jacobian-free GMRES a matrix–vector product is formed in the following manner. First $M_1^{-1}v$ is computed by a forward sweep followed by a back sweep using the ILU factorization. Then the product $AM_1^{-1}v$ is computed based on Eq. (9). Hence the primary costs of the inexact solution of the linear problem at each outer iteration include the initial formation of the ILU factorization plus two sweeps and a residual evaluation per GMRES iteration.

The ordering of the unknowns can significantly affect the performance of ILU preconditioning [38]. Among the orderings investigated by Pueyo and Zingg [13], they found the reverse Cuthill–McKee (RCM) ordering [39] to be the most effective. Hence it is the only ordering considered here. However, this issue remains open, and further orderings should be examined in this context. Pueyo and Zingg found that the initial ordering of the unknowns before applying the RCM reordering is also important. To understand why, we briefly review the RCM algorithm. All of the nodes are divided into a number of level sets $\mathrm{ls}(s)$ $s = 0, \ldots M$. The degree of a node is the number of neighbours it has in the graph of the matrix. The algorithm can be described as follows:

(1) Set $m = 0$. Choose a root node of minimum degree, which becomes the sole member of $\mathrm{ls}(0)$. In our case, all nodes of minimum degree are located on either the airfoil surface or the far-field boundary.
(2) For each node in $\mathrm{ls}(m)$, add all unordered neighbours to $\mathrm{ls}(m+1)$ in order of increasing degree.
(3) Set $m = m + 1$, and if unordered nodes remain, return to 2.
(4) Place the nodes in the following order: the entry in $\mathrm{ls}(0)$, followed by the entries (in order) in $\mathrm{ls}(1)$, followed by the entries in $\mathrm{ls}(2)$, etc.
(5) Reverse the order.

This algorithm operates by locally minimizing the average bandwidth of a wavefront, where a wavefront is the set of nodes in a level set. As presented, the algorithm is not fully defined. First, there may be several nodes of minimum degree, so there is some flexibility in the choice of the root node. Second, when neighbours of a node are added to the next level set, some may be of equal degree, and again there is an ambiguity. Typically, the initial ordering of the nodes is used to fully define the algorithm, and this explains its influence.

In the absence of reliable theory to guide the reordering, we have performed a vast number of experiments with different root nodes and strategies for choosing between two or more nodes of equal degree, examining both the estimated condition number of M^{-1} and the computing time for convergence. Based on the data generated, the following observations can be made:

- The convergence time is highly dependent on the root node choice, especially for turbulent flows; many choices can lead to nonconvergence.
- Choosing a root node on the far-field boundary is preferred over the airfoil surface.
- Choosing a root node on the downstream boundary is advantageous.

As a result, we always choose a root node on the downstream boundary (recall that the ordering is reversed, so the root node becomes the last node in the final ordering). Furthermore, consistent with this idea, when a choice must be made between two or more nodes of equal degree, the node that is downwind is chosen first. This consistently yields a significantly superior preconditioner.

4. Algorithm parameters

Although a JFNK algorithm is conceptually straightforward, there are a number of parameters that must be properly specified and details to which attention must be paid in order to achieve an efficient and robust solver. Our strategy is to find a set of parameters that is appropriate for a wide range of flow problems, rather than tuning the parameters for specific problems. We accept parameter variation, as long as it can be specified *a priori*. For example, it is prudent to develop different parameters for laminar and turbulent flows. This is due to the strong effect of the turbulence model, which is significantly less stable than the mean-flow equations. The turbulence model has a number of components that are highly nonlinear, especially in the trip terms, which are a challenge for Newton's method. Moreover, there is often a large difference in the magnitude of the residuals and Jacobian matrix entries between the turbulence model and mean-flow equations, which presents a challenge for the linear solver. The measures introduced to address the difficulties arising from the turbulence model represent a key contribution of this paper.

The optimal parameters have been selected based on comprehensive testing of the algorithm on a range of flow problems, including subsonic and transonic, inviscid and turbulent flows over single- and multi-element airfoils. The grids used vary from 12,000 nodes for the inviscid single-element problems to 72,000 nodes for the turbulent multi-element problems, with maximum cell aspect ratios ranging from 1900 to over 2 million. Multi-block grids are used for the multi-element configurations.

4.1. Equation and variable scaling

Row and column scaling can be applied to the linear system in the following manner:

$$S_r A S_c S_c^{-1} x = S_r b, \tag{11}$$

where S_r and S_c are diagonal matrices. S_r scales the residual vector, and S_c scales the solution vector. The reasons to use these scalings are primarily related to the fact that the linear system is solved inexactly. For example, as a result of the inherent

scaling of the Navier–Stokes equations by the inverse of the Jacobian of the coordinate transformation, the residual at nodes far from the body, where J^{-1} is large, is typically severals orders of magnitude larger than the residual at nodes near the body, where J^{-1} is small. To satisfy the requirement that the linear residual decrease by one order of magnitude, only the residual at nodes far from the body needs to be reduced. The residual at nodes near the body can increase, with potentially disastrous consequences for convergence of the nonlinear iterations. Similarly, the residual of the turbulence model equation can be orders of magnitude larger than that of the mean-flow equations. Again, the required reduction in the residual of the linear problem can be achieved by reducing only the residual associated with the turbulence model, while the mean-flow residual can increase, leading to nonconvergence of the nonlinear iterations.

The entries in the Jacobian matrix are also strongly affected by poor scaling of variables and equations. Fig. 1 shows a typical unscaled diagonal block in the near wake of an airfoil, where $J^{-1} = 10^{-5}$. The off-diagonal entries for the turbulence model (row 5, columns 1–4) are many orders of magnitude higher than the off-diagonal entries of the mean-flow equations, especially those resulting from the derivative with respect to the turbulence variable (rows 2–4, column 5). This disparity results in part from the scaling of the mean-flow variables and equations by J^{-1}. Note that the diagonal elements are close in magnitude.

A third area where a poor scaling of variables and equations can be detrimental is in the evolution of the time step during pseudo-transient continuation. Most such continuation methods determine the time step based on the norm of the residual. If the residual norm is dominated by a single component, such as the turbulence model, then the time step evolution will be appropriate for that component only and can be too large or too small for the under-represented component, leading to sub-optimal convergence, or divergence.

In the present solver, scaling problems have the following four primary sources:

- The mean-flow equations and variables have a node-by-node scaling by the inverse of the Jacobian of the coordinate transformation.
- The turbulence model equation and variables have no inherent geometric scaling.
- The turbulence model variable ranges up to roughly 1000, while the mean-flow variables typically do not exceed two.
- Local differences may arise, for example, near transition trip points.

In order to address the problems caused by poor scaling of variables and equations, we will consider two categories of scaling: inherent scaling and auto-scaling. The former is used to address *ab initio* scaling problems, such as the inherent scaling differences between our turbulence model and mean-flow variables and equations resulting from the presence or absence of J^{-1} scaling. This is equivalent to rewriting the equations in a different form. Auto-scaling is performed each iteration to address discrepancies between the residual norms of the various equations. We consider only scalings where the equation scaling is the same as the variable scaling, so that $S_r = S_c^{-1}$. This ensures that the eigenvalues and diagonal elements of the Jacobian matrix are unchanged.

After experimenting with several different scaling strategies, we have found the following approach to be effective. The mean-flow equations and variables are multiplied by J; this eliminates the disparity caused by the geometric scaling. The turbulence model variable and equation are multiplied by 10^{-3} to normalize with respect to the maximum value of the turbulence variable. This is analogous to nondimensionalizing with respect to eddy viscosity rather than viscosity. Auto-scaling is then performed to bring the maximum difference between the mean-flow and turbulence model residual norms to within an order of magnitude. The reference time step evaluation for the pseudo-transient continuation (Section 4.2) is calculated based on the residual norm after the inherent scaling is applied, but before the auto-scaling is applied. This combination of inherent scaling and auto-scaling reduces the disparities in the elements of the Jacobian blocks, ensures that the linear residuals of the mean-flow and turbulence equations must both be reduced during the inexact linear solve, and ensures that the time step evolution is dependent on both sets of equations. Overall, this scaling strategy leads to some improvement in the speed of the algorithm, but more importantly, greatly increases its robustness.

4.2. Globalization: pseudo-transient continuation

The pseudo-transient continuation strategy used to globalize the inexact-Newton method is based on the implicit Euler method introduced in Section 3.1. Once the time step in Eq. (3) reaches roughly 10^4, the method is effectively an inexact-

$$\begin{bmatrix} -3\cdot 10^{2} & 0 & 0 & 0 & 0 \\ 1\cdot 10^{1} & -4\cdot 10^{2} & 9\cdot 10^{-1} & 0 & 8\cdot 10^{-10} \\ 9\cdot 10^{-1} & 9\cdot 10^{-1} & -4\cdot 10^{2} & 0 & 2\cdot 10^{-11} \\ 2\cdot 10^{2} & 7\cdot 10^{0} & 3\cdot 10^{-1} & -4\cdot 10^{2} & 1\cdot 10^{-11} \\ 6\cdot 10^{7} & -1\cdot 10^{8} & -4\cdot 10^{9} & 5\cdot 10^{4} & -1\cdot 10^{2} \end{bmatrix}$$

Fig. 1. Typical unscaled diagonal block.

Newton method. The goal of the pseudo-transient continuation is to reach this stage as efficiently and reliably as possible. Since time accuracy is not needed, each node and each equation can have an individual time step that changes at each iteration. To simplify this situation, we use the following form for the time step:

$$\Delta t = \Delta t_{ref} \cdot \Delta t_{loc}. \tag{12}$$

The reference time step, Δt_{ref}, evolves as the iterations progress; the same value is used for all nodes. The local time step Δt_{loc} varies at each node, and potentially for each equation. The turbulence model requires an extra degree of stabilization to prevent negative values of the turbulence quantity from occurring. A special local time step formulation is used to accomplish this; this is presented in Section 4.3.

The norm of the residual provides a measure of the degree of convergence of the algorithm. Consequently, Δt_{ref} should increase as the residual norm decreases, suggesting the following formula

$$\Delta t_{ref} = \max(\alpha \cdot \|R\|_2^{-\beta}, \Delta t_{min}). \tag{13}$$

The parameter Δt_{min} is introduced to keep the reference time step from becoming too small. Without this parameter, a spike in the residual convergence history can lead to a very small time step, and a large number of iterations is then required to recover. Choosing $\beta = 1$ provides a rapid increase in the time step while maintaining stability. The parameters α and Δt_{min} are dependent on the nature of the flow. Conservative choices are as follows: $\alpha = 100$, $\Delta t_{min} = 100$ for subsonic flows, $\Delta t_{min} = 10$ for transonic flows. With this choice of α, the reference time step reaches 10^4 when the residual norm is 10^{-3}. For many flow problems, faster convergence can be obtained with higher values of Δt_{min} and α, but we use the above values to provide a robust algorithm that does not require parameter tuning. The parameters are chosen to minimize the computing time, not the number of nonlinear iterations, needed to achieve convergence. During the period when the time step is relatively small, the linear problem is easier to solve due to the increased magnitude of the diagonal elements, so the number of inner (GMRES) iterations per outer iteration is reduced.

There are some subtleties to be considered in using the above Δt_{ref} formulation. First, recall that the residual has been scaled such that inherent scaling differences are removed. This is important, as it ensures that all components contribute to the determination of the time step. Second, since our initial condition is a uniform flow, the residual in the interior of the domain is zero at the first iteration, and the sole contribution to the residual is from the boundary conditions. If the residual from the boundary conditions is small, then this will lead to a large time step for the first few iterations until the residual in the interior is sufficiently high to drive Eq. (13). This is undesirable because an initially large time step can be destabilizing, and furthermore, we do not want the boundary conditions to drive the time step, since they are not directly affected by the time step. In our solver, the initial boundary condition residuals are large enough to force $\Delta t_{ref} = \Delta t_{min}$ for the initial time steps, so this problem does not appear. However, it is an important issue to be aware of, in case measures need to be taken to avoid having a time step that is too large initially. Problems can also arise if Δt_{ref} increases too rapidly from one iteration to the next. We have thus introduced a safeguard that limits the maximum increase in Δt_{ref} to one order of magnitude. While this rarely takes effect, it occasionally prevents the solver from diverging.

Note that Eq. (13) is only effective when there is a clear connection between the residual and the degree of nonlinearity. Due to the high nonlinearity in the turbulence model, especially the trip terms to fix the transition point, this is not always the case. We have found that adjusting the parameter α is unsuccessful in addressing this situation. If α is too large, divergence will occur when the reference time step first ramps up. If α is too small, a long plateau occurs in the convergence history. Therefore, the following measure has been introduced to address this situation. If one or more nodes have been flagged to use a restricted time step based in the turbulence model stabilization described in Section 4.3, then we choose $\Delta t_{ref} = \Delta t_{min}$. This measure has little effect on many flow problems, but when the trip terms are particularly problematic, it can significantly speed up convergence.

Finally, we consider the local time step, Δt_{loc}. Following Pulliam [16], we use the following formula to provide a spatially varying time step:

$$\Delta t_{loc} = \frac{1}{1 + \sqrt{J}}. \tag{14}$$

This formula decreases the time step for small cells, consistent with maintaining a roughly constant Courant number, while recognizing that the grid spacings vary much more than the wave speeds. This assumes convection-dominated flow, and, although the turbulence model is less convection-dominated than the mean-flow, we have found this formula to be effective when applied to the turbulence model equation as well.

Grid sequencing can also be used to advantage during the pseudo-transient continuation phase. Beginning with a coarse grid, a series of grids is used to provide a good initial guess on the finest grid [16]. This has the advantage of initiating the solution on the fine grid much closer to the region of convergence of Newton's method, thus reducing the time spent in the continuation phase. A larger Δt_{min} and a more aggressive time step sequence can be used on the fine grid, while more conservative values are used on the coarse grids, where the iterations are quicker. We use a three-grid sequence, with each coarse grid being formed by removing every second grid line from its parent grid. Scalar dissipation is used on the coarse grids. The transition to the next grid level is initiated when the residual norm drops below 10^{-4}. The use of grid sequencing is not essential but speeds up convergence and improves robustness.

4.3. Turbulence model stabilization

The turbulence model introduces a number of challenges. First, the Spalart–Allmaras model is highly nonlinear, especially the production and trip terms, so efficient globalization is essential. Second, the model becomes unstable for negative $\tilde{\nu}$, so it is critical to ensure that $\tilde{\nu}$ remain positive.

A number of measures can help to avoid difficulties in the early iterations. The turbulence model is essentially meaningless unless the mean-flow is somewhat physically realistic. Therefore, we initially perform a few iterations of the mean-flow equations without the turbulence model equation in order to establish some shear near the body. In addition, the turbulence model is seeded with a low level of eddy viscosity ($\tilde{\nu} = 10$) in regions downstream of the trip points. The shear established near the body is sufficient to maintain the eddy viscosity where appropriate, while it rapidly convects out of regions that lack enough shear to sustain a significant level of eddy viscosity.

In order to ensure positive values of the turbulence quantity, Spalart and Allmaras [24] use a first-order upwind discretization for the convective terms in the turbulence model and modify the turbulence model Jacobian matrix such that it is an *M*-matrix. While this approach is effective, it is difficult to implement in a Jacobian-free manner and is not compatible with Newton-like convergence. Therefore, we retain the first-order discretization of the convective terms, but rather than modifying the Jacobian matrix, we use a local time step for the turbulence model equation in order to maintain positivity of $\tilde{\nu}$.

If we neglect off-diagonal terms, applying the implicit Euler method to the turbulence model equation results in

$$(\Delta t_{\tilde{\nu}}^{-1} - J_D)\Delta\tilde{\nu} = R_{\tilde{\nu}}, \tag{15}$$

where $\Delta t_{\tilde{\nu}}$ is the local turbulence model time step, J_D is the diagonal element of the Jacobian matrix for the turbulence model equation, and $R_{\tilde{\nu}}$ is the residual of the turbulence model equation. Note that $J_D < 0$. We wish to limit the update as follows:

$$|\Delta\tilde{\nu}| \leqslant |r| \cdot \max(\tilde{\nu}, 1), \tag{16}$$

where $|r|$ is a specified ratio that can range from 0.4 to 0.8. Choosing $|r| = 1$ would keep the updated $\tilde{\nu}$ positive in the uncoupled case, but in practice a smaller value is preferable. The maximum function ensures that values of $\tilde{\nu}$ very close to zero can be increased in a reasonable time, i.e. it prevents the time step from becoming too small. While this allows small negative values, these are clipped to zero after each update; such clipping is not needed during the later stages of convergence, so the steady solution is unaffected. Rearranging Eqs. (15) and (16), we can find an approximate maximum allowable time step:

$$\Delta t_{\tilde{\nu}} = \left[\frac{R_{\tilde{\nu}}}{r \cdot \max(\tilde{\nu}, 1)} + J_D\right]^{-1}, \tag{17}$$

where $\Delta\tilde{\nu}$, and therefore r, must have the same sign as $R_{\tilde{\nu}}$. This moves the update in the direction of the residual as required.

Note that the reference time step is held to Δt_{min} if the time step calculated from Eq. (17) is less than that calculated from Eq. (12) anywhere in the grid. This normally does not occur during the late stages of convergence, so that Newton-like convergence can be achieved. The use of a restricted time step for the turbulence model in this manner has proven to be a critical element in achieving a robust algorithm.

In other solvers, for example solvers using unstructured grids, the need for clipping may persist, both degrading convergence and affecting the steady solution. Under these circumstances, a more sophisticated procedure involving a modification to the source terms in the turbulence model is needed when $\tilde{\nu} < 0$ [23].

We conclude this section with an additional measure added when the transition trip terms are used in the turbulence model. These terms are destabilizing and highly nonlinear, but it is important to include them in order to ensure that transition occurs where expected. Following the suggestion of Spalart and Allmaras [24], we use a limited number of nodes within a reasonable distance of the trip for the calculation of the trip function. These additional coupling terms are not included in the approximate Jacobian used to form the preconditioner. Changes in the turbulence variable near the trip can be large enough to destabilize the solution. In order to damp this effect, we reduce the time step at nodes where the trip terms are highly sensitive to the velocity components. Nodes where the sum of the absolute values of the partial derivatives of the trip terms with respect to the velocity components exceed 10^5 have the time step reduced according to the following:

$$\Delta t_{ref}^{trip} = \begin{cases} \Delta t_{ref} & \Delta t_{ref} < 10, \\ 10 & 10 < \Delta t_{ref} < 1000, \\ \frac{\Delta t_{ref}}{100} & \Delta t_{ref} > 1000. \end{cases} \tag{18}$$

This measure is active only during the early stages of convergence during which the location of transition is established.

4.4. Preconditioner

Preconditioning is achieved using an ILU(p) factorization of an approximation to the Jacobian matrix. Parameters introduced include σ, which affects the approximate Jacobian and was discussed in Section 3.3, and the level of fill, p. Increasing p reduces the number of inner iterations by improving the effectiveness of the preconditioner. However, the cost of forming and applying the preconditioner is increased, as are the memory requirements, so there is a trade-off. In computations of two-dimensional flows, levels of fill between 2 and 4 are typically optimal. Lower values are preferred in three-dimensions

[19,20]. In the present work, a value of p equal to 4 is used in all cases. The effectiveness of the ILU(p) factorization is also influenced by the reordering used, and thus the choice of the root node in the RCM ordering is very important, as described in Section 3.3.

4.5. Linear system solution

Two subtle difficulties can arise in solving the linear system. First, the reduction in the largest component of the residual can be sufficient to meet the linear convergence tolerance even though some components of the residual have not decreased and may even have increased. This can lead to divergence of the nonlinear iterations. The scalings discussed previously are designed to alleviate this difficulty. In addition, we perform a minimum of five linear iterations at each nonlinear iteration. The number of linear iterations rarely drops below five, so this is rarely enforced. When it is enforced, it is usually needed, as it indicates that the residual reduction is probably misleading in that not all components have been uniformly reduced.

The second problem that arises in the linear system solution is associated with the Jacobian-free matrix–vector product, Eq. (7). As described in Section 3.2, the parameter ϵ must be carefully chosen such that both round-off and truncation error are sufficiently small. It is common to choose ϵ based on the following formula:

$$\epsilon = \frac{\sqrt{\delta}}{\|v\|_2}, \tag{19}$$

where most authors suggest using δ equal to the value of machine zero. If the entries in v vary greatly in magnitude, then $\|v\|_2$ will be a good representative value for the larger entries, but will be too large for the smaller entries. This leads to a value of ϵ that is too small for the smaller entries, thus producing round-off error in those components of $Q + \epsilon v$. If the variation in the magnitudes of the entries in v is too great, then it can be impossible to find a value of ϵ that meets the competing demands of round-off and truncation error. A similar argument applies to the difference of the two residuals in Eq. (7). If the components of the residual vector vary widely in magnitude, then there may be no choice of ϵ that is suitable. In both cases, large round-off errors can result from the small entries.

Clearly, the scalings described previously help to address this problem, which we call Jacobian-free breakdown, but they are not sufficient to eliminate it completely. This issue is particularly pernicious because it can manifest itself in the nonlinear iterations. The linear solver may converge, but to the solution of a different linear system. Alternatively, an increase in the number of inner iterations may be seen, resulting from a mismatch between the erroneous linear problem being solved and the preconditioner. One means of reducing the likelihood of Jacobian-free breakdown caused by round-off error is to use a higher-order finite-difference approximation in Eq. (7), which reduces the truncation error for a given value of ϵ, thereby permitting the use of larger values of ϵ, which in turn reduces the round-off error. This is, however, an expensive cure. Since we have found round-off error to be a more significant problem than truncation error, we instead address this issue by using a larger value of ϵ than that obtained with δ equal to machine zero. With $\delta = 10^{-10}$, Jacobian-free breakdowns are essentially eliminated. Alternatively, the norm of v can be divided by the square root of the number of nodes (i.e. a root-mean-square norm), and δ can be left at machine zero.

5. Algorithm performance

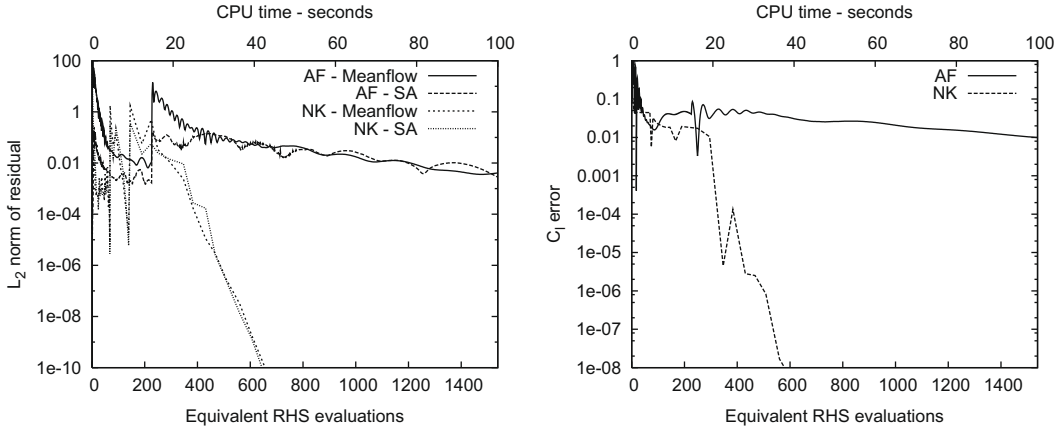
Before presenting some sample results to demonstrate the performance of the JFNK algorithm presented, we will review the algorithm parameters used to obtain these results. As discussed previously, the large number of parameters in Newton–Krylov algorithms is often seen as a negative characteristic. Therefore, we stress that the parameters have not been varied for the results presented here. The following are the parameters used to obtain all of the results presented in this section:

- The linear system residual is reduced by a factor of 10, i.e. $\eta_n = 0.1$ for all n in Eq. (6), with a minimum of five iterations.
- An ILU(p) preconditioner is used with $p = 4$, as described in Section 3.3. It is based on the approximate Jacobian, with $\sigma = 10.0$ in Eq. (10), since all results shown use matrix dissipation. The preconditioner is calculated before every outer iteration.
- Before applying the preconditioner, the system is reordered using the reverse Cuthill–McKee method, with the root node chosen on the downstream boundary, as discussed in Section 3.3.
- The equations are scaled as described in Section 4.1, including both inherent scaling and auto-scaling.
- The parameters used in the reference time step equation (Eq. (13)) are:
  - $\alpha = 100$.
  - $\Delta t_{min} = 100$.
  - $\beta = 1$.
- In Eq. (17), $|r| = 0.5$.
- Matrix–vector products are calculated with the Jacobian-free approach. The parameter used in Eq. (19) is $\delta = 10^{-10}$.
- Grid sequencing is used, as described in Section 4.2. Three grids are used, with $\alpha = 10$ on the coarsest grid and scalar dissipation on both coarse grids. The switch from coarse grid to the next finer grid is made when the residual reaches 10^{-4}.

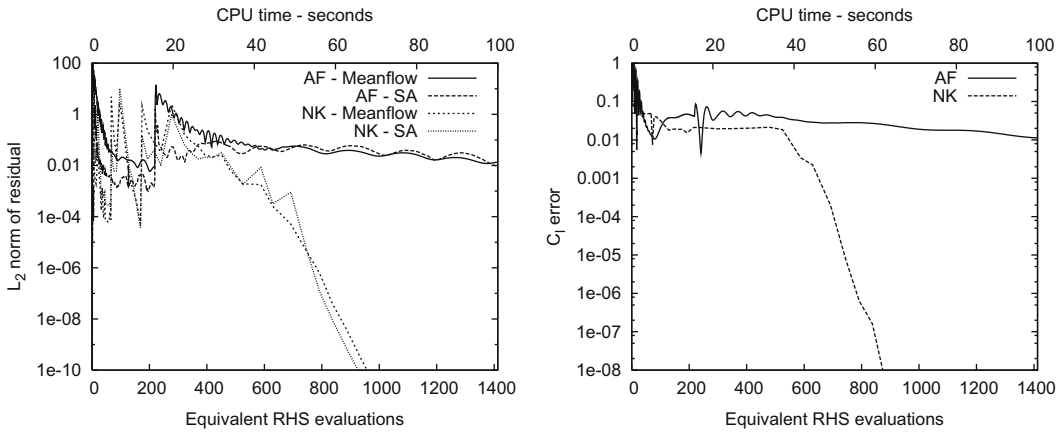
In order to provide a reference that may be useful to a number of readers, the convergence results of the Newton–Krylov (NK) algorithm are compared with those of the well-known approximate factorization (AF) algorithm [16]. The latter algorithm was run as efficiently as possible, with standard parameters and grid sequencing. The spatial discretizations and hence the residual evaluations for the two solvers are precisely the same. Therefore, the converged steady solutions are identical, and exactly the same computing time is required for a residual evaluation.

Convergence histories are presented in terms of both the residual norm and the convergence of the lift coefficient, labelled "C_l error", which is the difference between the current lift coefficient and the final converged lift coefficient. The lift coefficient is the lift force per unit depth nondimensionalized by dividing by the dynamic pressure times the airfoil chord. The lift coefficient error is very easy to interpret, since it shows the number of decimal places to which the lift coefficient has converged. Moreover, in some cases with the approximate factorization algorithm, the residual fails to converge at a very small number of nodes, but the solution converges well overall. In such cases, the lift coefficient convergence provides a better representation of the convergence history, as the residual norm plot is misleading. The residuals of the mean-flow and turbulence model equations are shown separately. Normally the residual for the approximately factored algorithm does not include any scaling. To provide a one-to-one comparison, we scale the residual vector before the norm is evaluated.

Two measures of computing time are reported. The first is the actual computing time (CPU time) on an Intel 2800 Pentium 4 desktop computer with 1.5 gigabytes of memory. The second is the total computing time divided by the computing time required for a single evaluation of the residual, including the mean-flow equations, numerical dissipation, turbulence model and all boundary conditions, which we term "equivalent right-hand side (RHS) evaluations". It is important to understand that this measure is not a count of the actual number of residual evaluations; it is a normalization of the computing time in an effort to reduce the dependence on the processor. Both of these measures of computational effort have their shortcomings. The CPU time is processor dependent and therefore hinders comparisons across platforms. It includes the efficiency of both the spatial discretization and the iterative solver. The equivalent RHS evaluations measure enables comparisons across platforms and algorithms and to an extent evaluates the efficiency of the iterative solver alone. However, it favors an expensive spatial discretization because such a discretization increases the cost of a residual evaluation, which reduces



(a) without trip terms



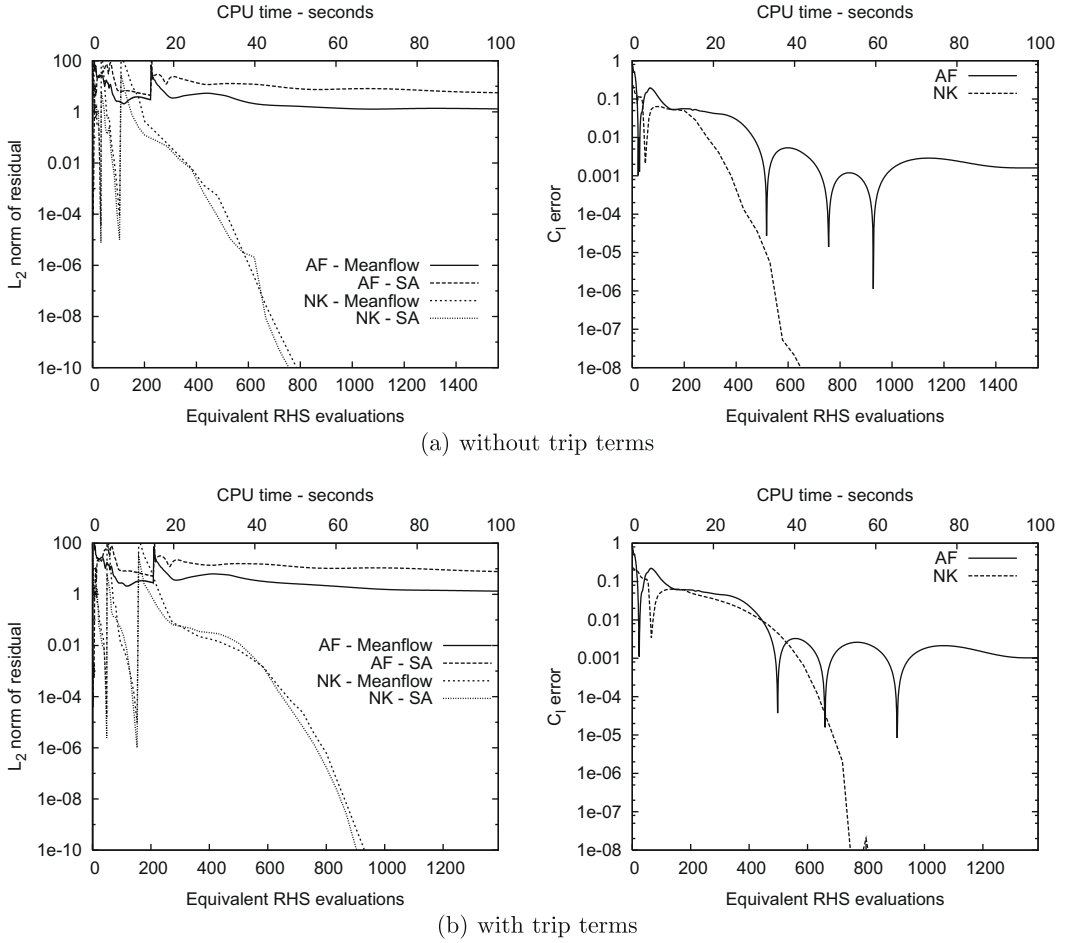
(b) with trip terms

Fig. 2. NACA 0012 airfoil at an angle of incidence of 6°, a Mach number of 0.15, and a Reynolds number of 9 million.

the effective cost of the iterative method. The residual evaluation in the present work is relatively inexpensive, primarily due to the use of a simple flux function. When performance measured in terms of equivalent RHS evaluations is compared with solvers with more expensive spatial discretizations and hence residual evaluations, such as higher-order schemes on unstructured grids or schemes with expensive flux functions and limiters, it is important to recognize this shortcoming of the use of this measure. In the present case, both solvers compared require the same computing time for a residual evaluation for a given flow problem, which varies roughly from 3×10^{-6} to 4×10^{-6} s/grid node. The trip terms in the turbulence model add about 10% to the cost of a residual evaluation.

The thesis of Chisholm [23] includes convergence results for subsonic and transonic inviscid flows over airfoils. For meshes with roughly 12,000 nodes, the residual norm is reduced to 10^{-10} in roughly 7 s, and the lift coefficient converges to four decimal places in 3–5 s. Here we will present results only for turbulent flows, both with and without the laminar-turbulent trip terms in the turbulence model. Although we recommend the use of the trip terms in order to ensure that transition occurs where expected, many researchers do not include them, and convergence is somewhat faster when they are excluded. All of the results presented use the matrix dissipation model; results obtained using scalar dissipation are shown in Chisholm [23].

Fig. 2 shows convergence histories for the flow over the NACA 0012 airfoil at an angle of incidence of 6°, a Mach number of 0.15, and a Reynolds number of 9 million. The grid has 17,385 nodes, with an off-wall spacing of 10^{-6} chords and a maximum cell aspect ratio of 13,672. Unless otherwise indicated, grids have been generated by an elliptic grid generator that allows significant user control. Grid lines emanating from the trailing edge have been flared to keep cell aspect ratios moderate. With the trip terms present, the JFNK algorithm reduces the residual ten orders of magnitude in roughly 60 s, or well under 1000 equivalent RHS evaluations. The lift coefficient converges to four figures in approximately 50 s, or 700 equivalent RHS evaluations. After 100 s, or 1400 equivalent RHS evaluations, the approximate factorization algorithm has reduced the residual by two orders of magnitude and converged the lift coefficient to two figures. Note that the approximate factorization algorithm converges to machine zero, but only a portion of the convergence history is shown. Without the trip terms, the JFNK algorithm converges 30–50% faster, while the effect on the approximate factorization algorithm is smaller. One aspect



(a) without trip terms

(b) with trip terms

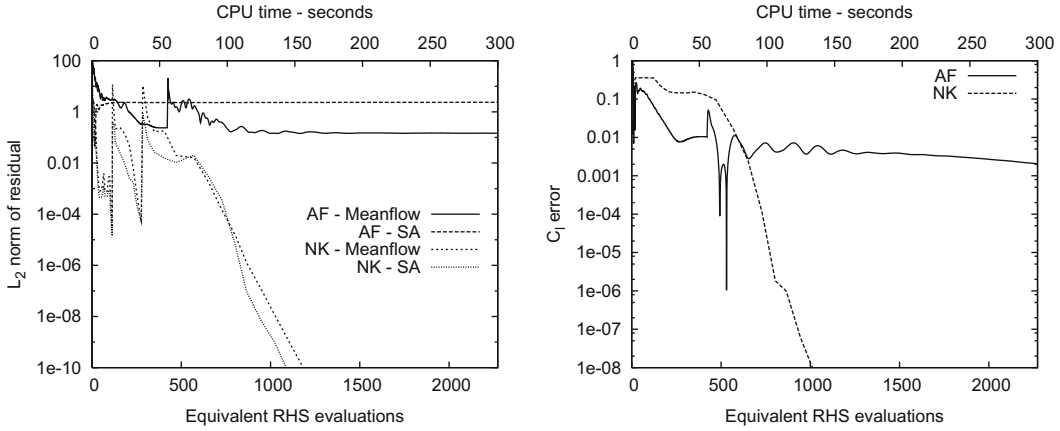
Fig. 3. RAE 2822 airfoil at an angle of incidence of 2.31°, a Mach number of 0.729, and a Reynolds number of 6.5 million.

of this particular flow is the Mach number, which is relatively low for a compressible flow solver without low Mach number preconditioning. The increased stiffness associated with the low Mach number appears to be having a greater adverse effect on the approximate factorization solver than on the JFNK solver.

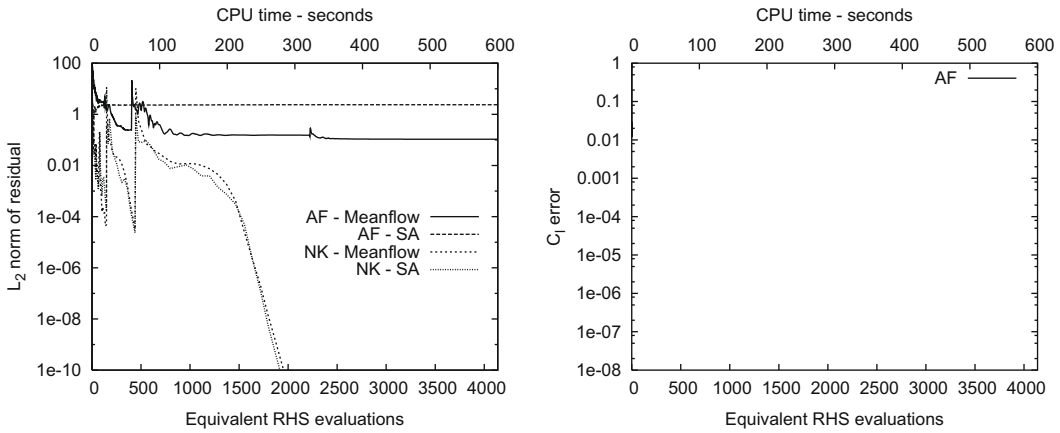
Fig. 3 shows the performance of the JFNK algorithm for a transonic flow over the RAE 2822 airfoil at an angle of incidence of 2.31°, a Mach number of 0.729, and a Reynolds number of 6.5 million. The grid has 17,385 nodes, with an off-wall spacing of 2×10^{-6} chords and a maximum cell aspect ratio of 8483. The performance of the JFNK algorithm is roughly the same as for the subsonic flow above, while the approximate factorization algorithm is more effective. Note that the spikes in the approximate factorization lift coefficient convergence history are not significant, as they simply indicate that the lift coefficient has passed through the converged value. This example demonstrates that the globalization strategy presented is capable of handling the strong nonlinearities associated with shock waves in transonic flows.

Fig. 4 shows convergence histories for a two-element configuration (main airfoil and flap deflected 20°) studied experimentally by van den Berg [40]. The angle of incidence is 6°, the Mach number is 0.3, and the Reynolds number is 2.51 million. The multi-block grid has 44,059 nodes, an off-wall spacing of 10^{-6} chords, and a maximum cell aspect ratio of 510,577. The JFNK solver initially converges quite slowly, but eventually converges 10 orders of magnitude in under 5 min or 2000 equivalent RHS evaluations, with the trip terms present. The lift coefficient converges to four figures in about 200 s, or 1500 equivalent RHS evaluations. Together with Fig. 3, this figure shows that the JFNK algorithm is particularly advantageous when deep convergence is needed.

The results presented thus far provide examples of the efficiency and reliability of the present JFNK algorithm in the solution of complex turbulent aerodynamic flows. In particular, the reliability of the algorithm stems primarily from the globalization strategies developed. The phase of the convergence history during which globalization is important can consume a large portion of the overall computing time. This makes the JFNK algorithm particularly well-suited to applications where the globalization phase can be reduced or avoided. For example, in aerodynamic shape optimization, the shape changes from one iteration to the next are often small. Hence the flow solver can be warm started from the previous solution, thereby shortening the globalization phase. Similarly, when a JFNK solver is used to solve the nonlinear problem arising at



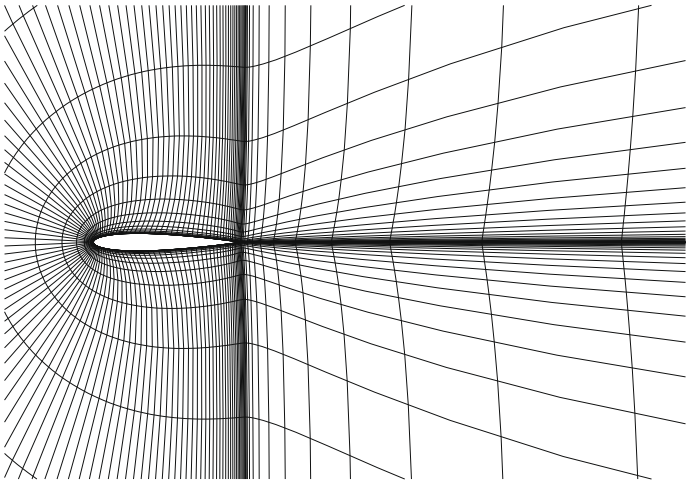
(a) without trip terms



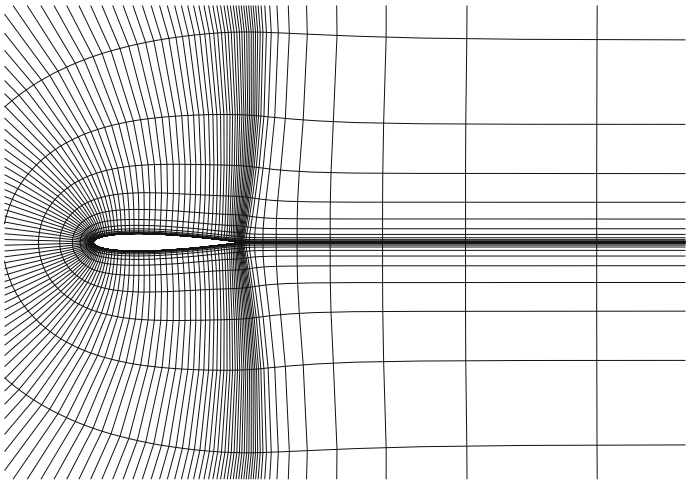
each time step of an implicit time-marching method in a time-accurate simulation of an unsteady flow, the initial guess at each time step can lie within the radius of convergence of Newton's method, and the globalization phase can be avoided or reduced.

Next, results are presented for some particularly difficult cases which further demonstrate the effectiveness of the JFNK algorithm. First we repeat the flow problem shown in Fig. 2 with a grid generated by a hyperbolic grid generator. As shown in Fig. 5, the grid lines emanating from the trailing edge cannot be flared as with the elliptic grid generator; hence the cell aspect ratios are greatly increased, with a maximum of over 2 million. Convergence histories are displayed in Fig. 6. The performance of the JFNK algorithm is almost unaffected relative to that shown in Fig. 2. The performance of the approximate factorization algorithm is also unaffected initially, but its convergence slows significantly later on in the convergence history. It is likely that an algorithm based on explicit multi-stage time stepping would converge very slowly on a grid that includes cells with such high aspect ratios.

For the results presented in Fig. 4, the blunt trailing edges of the two elements were closed by rotating the two surfaces, as is common practice. Fig. 7 shows the convergence of the lift coefficient with the blunt trailing edges retained. The grid has a long thin block behind each trailing edge, leading to a large number of cells with high aspect ratios; the maximum cell aspect ratio is over a half million. The JFNK algorithm converges somewhat more slowly, especially with the trip terms present, but it is able to converge to a steady solution, while the approximate factorization algorithm does not. Residual histories are not shown, as the turbulence model residual stalls early on at a limited number of points with the approximate factorization algorithm, so the residual is not a good measure of convergence. As the two solvers have identical spatial discretizations, a fully coupled turbulence model appears to be an important aspect of achieving convergence for such flows.



(a) Grid generated by elliptic grid generator



(b) Grid generated by hyperbolic grid generator

Fig. 5. Elliptic and hyperbolic grids for results shown in Figs. 2 and 6.

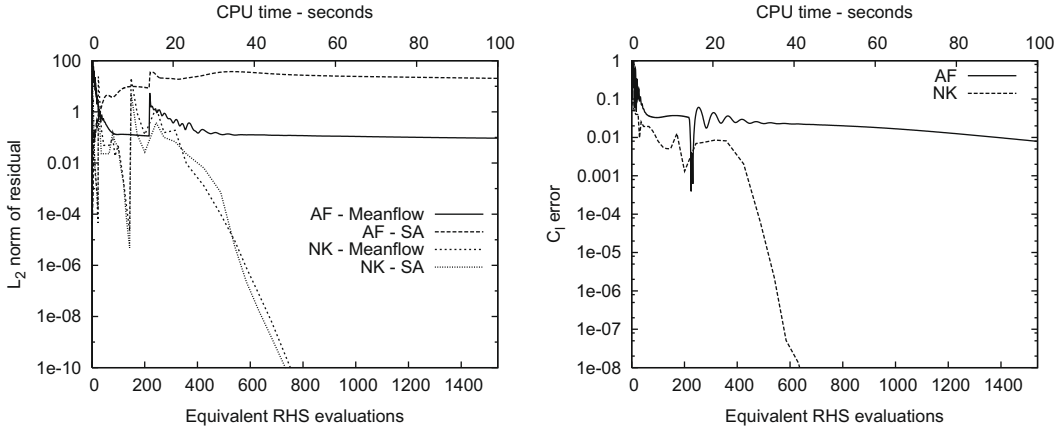
Finally, we show convergence results for a particularly challenging flow problem, the flow over a three-element configuration, an airfoil with slat and flap investigated experimentally by Moir [41]. The configuration is at an angle of incidence of 20.18°, the Mach number is 0.197, and the Reynolds number is 3.52 million. This angle of incidence is close to maximum lift, and there are large regions of separation in the coves of the slat and main element. The grid has 71,868 nodes, with an off-wall spacing of 10^{-6} chords and a maximum cell aspect ratio of 18,434. Fig. 8 shows that the JFNK algorithm takes some time to reach the point where it can converge rapidly, especially with the trip terms present, indicating that this flow is a severe test of the globalization strategies. It converges quite rapidly without the trip terms, suggesting that some work should be done to develop trip terms that are not so problematic.

6. Diagnosing problems in a JFNK algorithm

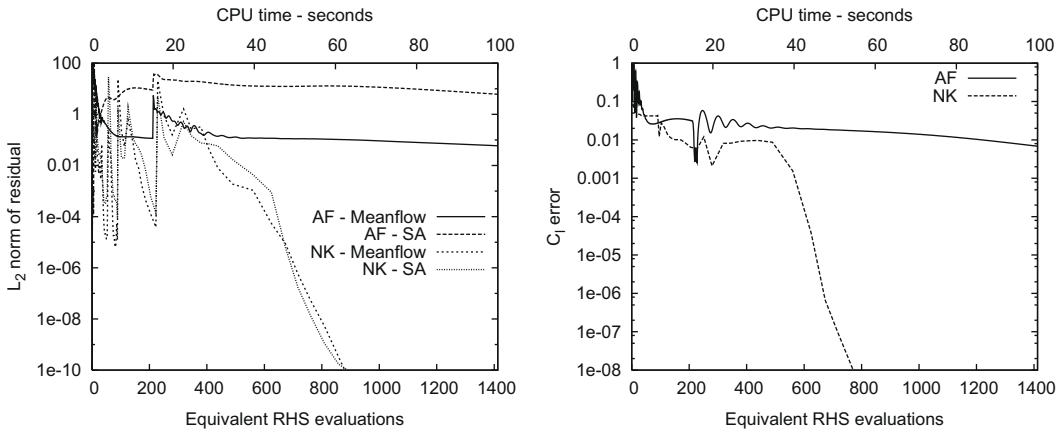
This section is included to aid those readers who are developing JFNK solvers that are similar to ours but involve some significant differences. For example, the physics, the spatial discretization, or the turbulence model might be different. Therefore, although many of the ideas and techniques introduced in this paper are relevant, they may need to be modified in order to apply to the specific details of the solver under development. We will assume during this discussion that all elements of the residual evaluation, including the spatial discretization and the boundary conditions, have been correctly coded.

The obvious symptom of a problem is typically a failure to decrease the nonlinear residual, a large increase in the nonlinear residual, or the appearance of an unphysical quantity, such as a negative pressure, after a nonlinear update. The first step in diagnosing the problem is to identify whether it lies with the linear solver or the nonlinear solver. Difficulties with the linear solver include preconditioner failure, Jacobian-free breakdown, and scaling problems. Failure of the nonlinear portion of the solver is usually associated with globalization.

If the linear solution is unable to meet the required convergence tolerance, then the problem lies with the linear solver. However, successful convergence of the linear solution does not prove that the difficulty lies with the nonlinear component. If Jacobian-free breakdown or scaling problems are occurring, the linear solver may appear to converge well, but the non-

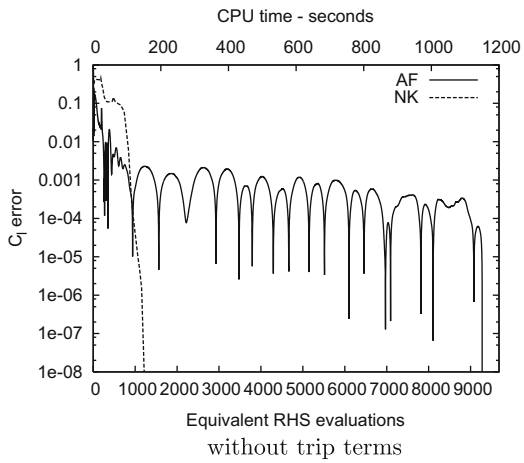


(a) without trip terms

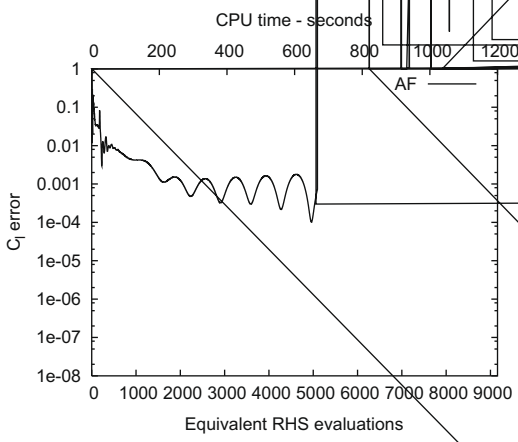


(b) with trip terms

Fig. 6. Hyperbolic grid results.



without trip terms



linear iterations diverge. Under these circumstances, it is important to recognize that the linear solver is the cause of the problem. In order to diagnose scaling problems, the first step is to calculate the linear residual reduction for each equation separately. The residual norm for each equation should be reduced by the desired factor. If not, then there is likely a problem with the scaling of the equations. If this test is passed, then one must examine the residual reduction on a node by node

basis. If the residual at some nodes is not being reduced, this indicates a scaling problem that is spatially dependent (such as the J^{-1} scaling we discussed previously).

In order to diagnose Jacobian-free breakdown, one can form the Jacobian matrix explicitly, either by hand or through a series of finite-difference or complex-step [43–45] approximations. The linear residual computed using the Jacobian-free formula can be compared with that computed using the Jacobian matrix. If they disagree, then Jacobian-free breakdown has occurred and consideration should be given to increasing δ in Eq. (19). Forming the Jacobian matrix explicitly can be time consuming, and it may be worthwhile to examine the impact of varying δ first. Large values of δ can be combined with a higher-order difference formula in computing the matrix–vector product. If this solves the problem, then this identifies Jacobian-free breakdown as the cause of the convergence problem.

Returning to a failure of the linear system to converge, preconditioner breakdown caused by small pivots or unstable LU solves should be considered. Two statistics can be helpful in identifying this. The first is simply the size of the reciprocal of the smallest pivot. If it approaches the reciprocal of machine zero, then this is the likely problem. The second is a condition number estimate given by $\|(LU)^{-1}e\|_\infty$, where $e = (1,\ldots,1)^T$ [42], which gives an indication of the stability of the triangular solves. Increasing σ in Eq. (10) can help to alleviate either of these difficulties. The preconditioner can also be inadequate if the approximate Jacobian matrix upon which it is based is too different from the true Jacobian matrix. This is often indicated if increasing fill is not helpful. In this case, some of the approximations made in the approximate Jacobian must be reevaluated. If preconditioner breakdown and an inadequate approximate Jacobian can be ruled out, then the level of fill may need to be increased. Finally, the reordering can have an important effect on the effectiveness of the preconditioner, and a poor ordering must be considered as a possible cause of preconditioning problems manifested by poor convergence of the linear iterations.

If the linear system converges well, and both scaling problems and Jacobian-free breakdown can be ruled out, then an improved globalization strategy is required. In order to develop such a strategy, the following steps can be helpful. First, solve the problem with a tight linear tolerance and a very small reference time step in order to establish the maximum stable initial reference time step. Then determine when a full Newton step can be taken by experimenting with initiating the Newton step at various stages in the convergence history. Next, establish how the time step can be effectively ramped from the required initial time step to the Newton stage. Finally, repeat while loosening the linear tolerance.

7. Conclusions

An efficient and reliable JFNK algorithm for turbulent aerodynamic flows has been presented. In the process, several important aspects of JFNK algorithms have been elucidated, and measures for addressing them have been described. The performance of the algorithm is demonstrated for a wide ranging suite of flow problems to be highly competitive with popular algorithms for computing turbulent aerodynamic flows. Hence the JFNK algorithm merits serious consideration as the algorithm of choice for this application.

References

[1] D. Knoll, D. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, J. Comput. Phys. 193 (2004) 357–397.
[2] W. Gropp, D. Keyes, L. McInnes, M. Tidriri, Globalized Newton–Krylov–Schwarz algorithms and software for parallel implicit CFD, Int. J. High Perform. Comput. Appl. 14 (2000) 102–136.
[3] D. Jespersen, T.H. Pulliam, P.G. Buning, Recent enhancements to OVERFLOW, AIAA Paper 97-0644, 1997.
[4] S.L. Krist, R.T. Biedron, C.L. Rumsey, CFL3D user's manual (version 5.0), NASA TM-1998-208444, 1998.
[5] V.N. Vatsa, B.W. Wedan, Development of a multigrid code for 3D Navier–Stokes equations and its application to a grid refinement study, Comput. Fluid 18 (1990) 391–403.
[6] M.J. Aftosmis, M.J. Berger, G. Adomavicius, A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries, AIAA Paper 2000-0808, 2000.
[7] W.K. Anderson, D.L. Bonhaus, An implicit upwind algorithm for computing turbulent flows on unstructured grids, Comput. Fluid 23 (1994) 1–22.
[8] E.M. Lee-Rausch, V.N. Vatsa, C.L. Rumsey, Computational analysis of dual radius circulation control airfoils, AIAA Paper 2006-3012, June 2006.
[9] H. Luo, H.D. Baum, R. Lohner, A fast, matrix-free implicit method for compressible flows on unstructured grids, J. Comput. Phys. 146 (1998) 664–690.
[10] T. Smith, R. Hooper, C. Ober, A. Lorber, Intelligent nonlinear solvers for computational fluid dynamics, AIAA Paper 2006-1483, 2006.
[11] S.A. Northrup, C.P.T. Groth, Solution of laminar combusting flows using a parallel implicit adaptive mesh refinement algorithm, in: Proceedings of the Fourth International Conference on CFD, ICCFD4, Springer-Verlag, 2008.
[12] M. Nemec, D.W. Zingg, A Newton–Krylov algorithm for aerodynamic design using the Navier–Stokes equations, AIAA J. 40 (2002) 1146–1154.
[13] A. Pueyo, D.W. Zingg, Efficient Newton–Krylov solver for aerodynamic computations, AIAA J. 36 (1998) 1991–1997.
[14] A. Nejat, C. Ollivier-Gooch, A high-order accurate unstructured finite volume Newton–Krylov algorithm for inviscid compressible flows, J. Comput. Phys. 227 (2007) 2582–2609.
[15] S. Isono, D.W. Zingg, A Runge–Kutta Newton–Krylov algorithm for fourth-order implicit time marching applied to unsteady flows, AIAA Paper 2004-0433, 2004.
[16] T.H. Pulliam, Efficient solution methods for the Navier–Stokes equations, Lecture Notes for the von Karman Inst. for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Brussels, Belgium, 1986.
[17] M. Blanco, D.W. Zingg, A fast Newton–Krylov solver for unstructured grids, AIAA J. 36 (1998) 607–612.
[18] M. Blanco, D.W. Zingg, A Newton–Krylov algorithm with a loosely coupled turbulence model for aerodynamic flows, AIAA J. 45 (2007) 980–987.
[19] P. Wong, D.W. Zingg, Three-dimensional aerodynamic computations on unstructured grids using a Newton–Krylov approach, Comput. Fluid 37 (2008) 107–120.
[20] J.C. Nichols, D.W. Zingg, A three-dimensional multi-block Newton–Krylov flow solver for the Euler equations, AIAA Paper 2005-5230, 2005.
[21] J.E. Hicken, D.W. Zingg, A parallel Newton–Krylov solver for the Euler equations discretized using simultaneous approximation terms, AIAA J. 46 (2008) 2773–2786.

[22] M.P. Rumpfkeil, D.W. Zingg, The optimal control of unsteady flows with a discrete adjoint method, Optim. Eng. (2008). doi:10.1007/s11081-008-9035-5.
[23] T.T. Chisholm, A fully coupled Newton–Krylov solver with a one-equation turbulence model, Ph.D. Thesis, University of Toronto Institute for Aerospace Studies, 2007.
[24] P. Spalart, S. Allmaras, A one-equation turbulence model for aerodynamic flows, AIAA Paper 92-0439, 1992.
[25] G.A. Ashford, An unstructured grid generation and adaptive solution technique for high Reynolds number compressible flows, Ph.D. Thesis, University of Michigan, 1996.
[26] R.C. Swanson, E. Turkel, On central-difference and upwind schemes, J. Comput. Phys. 101 (1992) 292–306.
[27] D.W. Zingg, S. De Rango, M. Nemec, T.H. Pulliam, Comparison of several spatial discretizations for the Navier–Stokes equations, J. Comput. Phys. 160 (2000) 683–704.
[28] P. Godin, D.W. Zingg, T.E. Nelson, High-lift aerodynamic computations with one and two-equation turbulence models, AIAA J. 35 (1997) 237–243.
[29] T.E. Nelson, P. Godin, S. De Rango, D.W. Zingg, Flow computations for a three-element airfoil system, Cdn. Aero. Space J. 45 (1999) 132–139.
[30] R.S. Dembo, S.C. Eisenstat, T. Steihaug, Inexact Newton methods, SIAM J. Numer. Anal. 19 (1982) 400–408.
[31] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–869.
[32] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing Company, Boston, 1996.
[33] H.A. van der Vorst, C. Vuik, GMRESR: a family of nested GMRES methods, Numer. Linear Algebra Appl. 1 (1993) 1–7.
[34] H. Lomax, T.H. Pulliam, D.W. Zingg, Fundamentals of Computational Fluid Dynamics, Springer, Germany, 2001.
[35] Y. Saad, A flexible inner–outer preconditioned GMRES algorithm, J. Sci. Stat. Comput. 14 (1993) 461–469.
[36] P.D. Orkwis, Comparison of Newton's and quasi-Newton's method solvers for the Navier–Stokes equations, AIAA J. 31 (1993) 832–836.
[37] H.C. Elman, A stability analysis of incomplete LU factorizations, Math. Comput. 47 (1986) 191–217.
[38] M. Benzi, D.B. Szyld, A.V. Duin, Orderings for incomplete factorization preconditioning of nonsymmetric problems, SIAM J. Sci. Comput. 20 (1999) 1652–1670.
[39] W.H. Liu, A. Sherman, Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices, SIAM J. Numer. Anal. 13 (1976) 198–213.
[40] B. van den Berg, Boundary layer measurements on a two-dimensional wing with flap, Technical Report TR 79009U, NLR, The Netherlands, 1979.
[41] J.R.M. Moir, Measurements on a two-dimensional aerofoil with high-lift devices, AGARD AR 303 (1994).
[42] E. Chow, Y. Saad, Experimental study of ILU preconditioners for indefinite matrices, Technical Report UMSI 97/95, University of Minnesota Supercomputing Institute, June 1997.
[43] W. Squire, G. Trapp, Using complex variables to estimate derivatives of real functions, SIAM Rev. 1 (1998) 110–112.
[44] W.K. Anderson, J.C. Newman, D.L. Whitfield, E.J. Nielsen, Sensitivity analysis for the Navier–Stokes equations on unstructured meshes using complex variables, AIAA Paper 99-3294, 1999.
[45] J.R.R.A. Martins, P. Sturdza, J.J. Alonso, The complex-step derivative approximation, ACM Trans. Math. Softw. 3 (2003) 245–262.