INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2000; **33**: 223–248

A matrix-free preconditioned Newton/GMRES method for unsteady Navier–Stokes solutions

Ning Qin*,[1], David K. Ludlow and Scott T. Shaw

Cranfield University College of Aeronautics, Bedfordshire, MK43 0AL, U.K.

SUMMARY

The unsteady compressible Reynolds-averaged Navier–Stokes equations are discretized using the Osher approximate Riemann solver with fully implicit time stepping. The resulting non-linear system at each time step is solved iteratively using a Newton/GMRES method. In the solution process, the Jacobian matrix–vector products are replaced by directional derivatives so that the evaluation and storage of the Jacobian matrix is removed from the procedure. An effective matrix-free preconditioner is proposed to fully avoid matrix storage. Convergence rates, computational costs and computer memory requirements of the present method are compared with those of a matrix Newton/GMRES method, a four stage Runge–Kutta explicit method, and an approximate factorization sub-iteration method. Effects of convergence tolerances for the GMRES linear solver on the convergence and the efficiency of the Newton iteration for the non-linear system at each time step are analysed for both matrix-free and matrix methods. Differences in the performance of the matrix-free method for laminar and turbulent flows are highlighted and analysed. Unsteady turbulent Navier–Stokes solutions of pitching and combined translation–pitching aerofoil oscillations are presented for unsteady shock-induced separation problems associated with the rotor blade flows of forward flying helicopters. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: GMRES; matrix-free; Newton; precondition; transonic; unsteady

1. INTRODUCTION

For most practical applications the efficiency of computational fluid dynamics (CFD) codes is a key issue for shortening design and analysis processes. The overall efficiency of a code depends upon three major aspects: (i) the convergence rate of the iterative scheme, (ii) the CPU time for each iteration, and (iii) the memory required to run the code. Memory efficiency is important as it is usually this factor that limits the size of the problem one can solve on a given computer and therefore restricts the resolution of the discretization of the flow problem.

* Correspondence to: Cranfield University College of Aeronautics, Bedfordshire, MK43 0AL, U.K.
[1] E-mail: n.qin@cranfield.ac.uk

Received July 1998
Revised May 1999

Explicit methods are memory efficient but are limited by stability conditions, which make convergence disappointingly slow, especially for viscous applications where highly stretched grids are necessary. Hence, implicit methods have been developed to avoid this limitation, resulting in a faster convergence rate. Although implicit methods need more CPU time per iteration due to the requirement of solving large sparse linear systems, in most cases this is amply offset by the greater convergence rate.

Some of the most attractive implicit methods are based upon Newton's method for solving the non-linear system of equations arising from discretization of the governing flow equations. Given a good initial condition, these methods should theoretically provide quadratic or superlinear convergence, which means that the solution of the non-linear system is obtained after a very small number of iterations. However, this inevitably requires more computation per iteration compared with approximate implicit schemes, such as the popular approximate factorization scheme, where the resulting linear system of equations is typically better conditioned or simpler to solve. The success of Newton methods will depend heavily on the efficiency with which the linear system is solved and on the availability of a good initial condition.

Exact Newton methods require the analytical calculation of the Jacobian matrix, and the exact solution of the resulting linear system. In practice it is often necessary or convenient to use Newton-like methods, such as the sparse finite difference Newton method, where the components of the Jacobian matrix are obtained by finite differences [1,2] and the sparse quasi-Newton method, where the Broyden update to the approximation of the Jacobian is used [1]. The sparse linear systems are generally solved using iterative linear solvers to certain tolerances, which can affect strongly the overall efficiency of the computation. Orkwis [3] has compared the performance of Newton methods for compressible laminar steady Navier–Stokes equations with different convergence tolerances with the solution of the linear system. Eisenstat and Walker [4] studied the effect of the linear solution tolerance on the local convergence property of the Newton methods and suggested some useful ways to determine the solution tolerance. More recently, Forsyth and Jiang [5] compared the Newton method with the defect-correction method, i.e. the simplified implicit method, for steady state Navier–Stokes solutions of compressible aerofoil flows and found that the former is not only more robust but also more efficient.

The main drawback of a fully implicit or Newton approach is the large memory required for storing the Jacobian and preconditioning matrices. This generally limits the use of such methods to small-scale problems unless a large memory machine is available. Recent developments in parallel computer architectures have made it possible to tackle the memory problem by using a large number of distributed memory parallel processors. One difficulty is to formulate a fully implicit parallel CFD code with an efficient parallelized preconditioner. For example, the popular ILU preconditioner becomes less effective after domain decomposition as full coupling among different domains for the preconditoner cannot be generally achieved. To address this problem, Qin *et al.* [6] and Xu *et al.* [7] have developed Newton-like methods to solve steady state problems using a combination of block diagonal and damping preconditioning for a restarted GMRES linear solver. The simplicity of the preconditioner led to a convenient and efficient parallelization of the methodology to relax the large memory required for the Jacobian storage. More recently, developments have been made in parallel preconditioning using Schwarz-ILU domain-decomposition preconditioners [8,9], in which the ILU preconditioning is applied on overlapped sub-domains.

Another approach to reduce the memory requirement is to use matrix-free methods combined with Krylov sub-space based linear solvers to solve the large sparse linear system. Some of the earliest work developing matrix-free methods for the numerical solution of differential equations was performed by Gear and Saad [10], Chan and Jackson [11] and Brown and Hindmarsh [12]. Further, some local convergence theory for matrix-free Newton-like methods has been established by Brown [13]. Johan [14] has used matrix-free methods in a finite element context to solve the compressible Navier–Stokes equations. In their study of incompressible flows, McHugh and Knoll [15,16] compared the convergence behaviour of a matrix-free Newton/CGS method with the corresponding matrix method. However, in their matrix-free method they still evaluate the Jacobian matrix for generating the preconditioning matrix and store the matrix in the form of an incomplete lower–upper (ILU) decomposition. In a subsequent paper, Knoll *et al.* [17] studied a chemically reacting compressible flow and showed that by using the same ILU preconditioning matrix over several Newton iterations, the matrix-free method may take less CPU time than their matrix counterpart, where the Jacobian is evaluated using finite difference for overall convergence. Recently, Choquet and Erhel [18] have studied a matrix-free Newton/GMRES method and have applied it to discretizations of the steady compressible Euler and Navier–Stokes equations in two distinct ways. In their first approach, the steady equations are preconditioned by the addition of a time derivative term and Newton's method is applied at each time step. Their second approach is to solve the steady equations directly by a Newton method. Their test cases are such that no preconditioning seems to be necessary.

If high-resolution methods are used for the spatial discretization of the compressible Navier–Stokes equations, the Jacobian matrix is usually ill-conditioned. Therefore, the matrix needs preconditioning before an iterative linear solver can be successfully used. Obviously, there are a large number of different ways a linear system can be 'preconditioned'. However, the effectiveness of the preconditioning depends on whether the redistribution of the eigenvalues improves the convergence of the linear solver used. Some of the most successful preconditioners used for this class of problems are the family of ILU preconditioning matrices [17,19]. Use of ILU preconditioning requires storage of a matrix that is at least the size of the Jacobian matrix of the system. In addition to the explicit matrix preconditioning, the system can also be preconditioned 'implicitly' by any modification of the original system to a mathematically equivalent system, such as the multigrid methods and the point or line Gauss–Seidel methods. Preconditioning can also be invoked at the original governing equation stage before the discretization. It is primarily associated with physical related singularities of the system, such as at the incompressible limit ($M=0$) for a compressible solver. However, this will certainly also have an effect on the condition of the resulting linear system. The scope of the current study is limited to explicit matrix preconditioning. No attempt has been made to precondition the compressible Navier–Stokes equations for the incompressible limit.

If a matrix-free Newton/Krylov method still needs to store a preconditioning matrix, the whole solution procedure is obviously not fully matrix-free. The development of a truly matrix-free procedure requires efficient matrix-free preconditioners other than the ILU preconditioner.

Badcock *et al*. [20,21] have recently developed an efficient preconditioning procedure based on an approximate factorization (AF) for an unfactored implicit scheme. Although the method is not a Newton-like method, an analytical Jacobian was obtained with the help of a symbolic package (REDUCE) and is stored and used in the linearized unfactored implicit solution. Furthermore, the AF preconditioner, based on a directional factorization of the Jacobian, is also stored in an array that is the same size as the Jacobian matrix.

The present paper addresses primarily two issues. Firstly, we devise and investigate a fully matrix-free, low-memory, Newton-like method regarding its convergence properties and computational efficiency in comparison with both its matrix counterpart and a conventional AF implicit method. Secondly, we study its applications to unsteady turbulent transonic aerofoil flows in comparison with similar laminar flow problems. To our knowledge, no success has been reported in using Newton-like methods for unsteady turbulent transonic flows. To generate the initial solution for the unsteady problem, the application of the method to a steady state problem is also addressed. The paper is arranged in the following layout. The physical and mathematical model is briefly described in Section 2. The numerical methodology, including spatial and temporal discretization and solution procedures, is described in detail in Section 3. In the solution procedure, the resulting large sparse linear system is solved by the GMRES(k) method, within which the Jacobian matrix–vector products are replaced by finite difference calculations of the directional derivatives. A matrix-free preconditioner is devised based upon the AF preconditioner so that neither the Jacobian matrix nor the preconditioning matrix is stored. Differences between the application of this method to the steady problem and that to the unsteady problem are also discussed in Section 3. In Section 4 numerical results are presented and discussed using both steady and unsteady solution procedures for laminar and turbulent transonic flows. Convergence rates, CPU time and memory requirements are compared with a corresponding explicit fourth-order Runge–Kutta solver, an AF sub-iteration implicit method and a matrix Newton/GMRES solver. Effects of convergence tolerances for the linear solver on the convergence of the matrix-free and matrix Newton iterations and numerical efficiency of the overall procedures are also discussed. Difficulties with Newton-like methods for turbulent problems are also highlighted and its implication on convergence tolerances is studied for computational efficiency. Finally, unsteady turbulent Navier–Stokes solutions of combined translation–pitch aerofoil oscillation are presented for transonic shock induced separation problems arising from the flow around the rotor blades of forward flying helicopters.

2. PHYSICAL MODEL

The physical problem under consideration is that of compressible viscous air flow involving shock waves, shear layers (including boundary layers) and their interactions. The mathematical model used is the two-dimensional compressible thin-layer Navier–Stokes equations.

The law of conservation of mass, momentum and energy over an area S bounded by a contour line l can be expressed in integral form as

$$\frac{\partial}{\partial t}\int_S \boldsymbol{q}\,\mathrm{d}S + \int_l (\boldsymbol{H}\cdot\boldsymbol{n})\,\mathrm{d}l = \boldsymbol{0} \tag{1}$$

where $\boldsymbol{q}$ is the vector of conserved variables $(\rho, \rho u, \rho v, \rho E)^T$, in which ρ is the density of the fluid, u and v are the components of velocity in a Cartesian co-ordinate system and E is the energy of the fluid particle element, $E = e + \frac{1}{2}\rho(u^2 + v^2)$. In Equation (1), $\boldsymbol{n}$ is the outward pointing unit vector normal to the line l.

The flux tensor $\boldsymbol{H}$ is made up from contributions that may be conveniently grouped together as convection and diffusion terms, denoted by the superscripts i and v respectively. In a Cartesian co-ordinate system, the convective fluxes may be written as

$$\boldsymbol{H}_x^{\mathrm{i}} = \begin{bmatrix} \rho\bar{U} \\ \rho u\bar{U} + p \\ \rho v\bar{U} \\ \bar{U}(\rho E + p) + u_T p \end{bmatrix} \quad \text{and} \quad \boldsymbol{H}_y^{\mathrm{i}} = \begin{bmatrix} \rho\bar{V} \\ \rho u\bar{V} \\ \rho v\bar{V} + p \\ \bar{V}(\rho E + p) + v_T p \end{bmatrix}$$

The contravariant velocities $\bar{U}$ and $\bar{V}$ are

$$\bar{U} = u - u_T \quad \text{and} \quad \bar{V} = v - v_T$$

where u_T and v_T represent the grid velocities, $\mathrm{d}x/\mathrm{d}t$ and $\mathrm{d}y/\mathrm{d}t$, with which the integration boundaries move. The corresponding components of the diffusive flux vector are

$$\boldsymbol{H}_x^{\mathrm{v}} = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} - q_x \end{bmatrix} \quad \text{and} \quad \boldsymbol{H}_y^{\mathrm{v}} = \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} - q_y \end{bmatrix}$$

in which τ is the shear stress tensor and q is the heat flux vector. The thin-layer approximation is used when evaluating τ, i.e. derivatives of flow variables with respect to the generalized co-ordinate in the wall-tangential direction are neglected.

In order to obtain a closed mathematical system, auxiliary relations describing the interdependence of the thermodynamic variables are introduced. For a perfect gas, the gas-law relates the pressure p, density and temperature T by $p = \rho RT$, where R is the universal gas constant. Further, the internal energy e is defined by $e = (\gamma - 1)^{-1}p\rho^{-1}$, where γ is the ratio of specific heats.

The laminar viscosity in the shear stress is determined as a function of T by Sutherland's formula. In turbulent cases, the Reynolds-averaged Navier–Stokes equations are used with the Baldwin–Lomax algebraic turbulence model [22].

3. NUMERICAL METHODOLOGY

3.1. *Spatial and implicit temporal discretization*

A cell-centred finite volume scheme built upon the flux difference splitting scheme of Osher and Solomon [23] is used for the spatial discretization of the convective flux. The convective numerical flux at the cell interface is evaluated using an approximate Riemann solver, which can be written as

$$\tilde{e}_{i+1/2,j} = \frac{1}{2}[\bar{e}(\boldsymbol{q}^{\mathrm{L}}) + \bar{e}(\boldsymbol{q}^{\mathrm{R}})] - \frac{1}{2}\int_{\boldsymbol{q}^{\mathrm{L}}}^{\boldsymbol{q}^{\mathrm{R}}} \left|\frac{\partial \bar{e}}{\partial \boldsymbol{q}}\right| \mathrm{d}\boldsymbol{q} \tag{2}$$

where $\bar{e}$ is the transformed inviscid flux vector in the ξ-direction and the superscripts 'L' and 'R' stand for the left and the right of the cell interfaces. The integration in the last term on the right-hand side in the state space is carried out using a *natural* ordering of the sub-paths parallel to the eigenvectors of the flux Jacobian $\partial\bar{e}/\partial\boldsymbol{q}$. To enhance the spatial resolution of the method, a MUSCL interpolation [24] of the primitive variables $\boldsymbol{w} = (\rho, u, v, p)^T$ is included, i.e.

$$\boldsymbol{w}^{\mathrm{L}}_{i+1/2,j} = \left\{1 + \frac{1}{4}s[(1-\kappa s)\Delta_- + (1+\kappa s)\Delta_+]\right\}\boldsymbol{w}_{i,j}$$

$$\boldsymbol{w}^{\mathrm{R}}_{i+1/2,j} = \left\{1 - \frac{1}{4}s[(1+\kappa s)\Delta_- + (1-\kappa s)\Delta_+]\right\}\boldsymbol{w}_{i+1,j}$$

together with a flux limiter

$$s = \frac{2\Delta_+\boldsymbol{w}\Delta_-\boldsymbol{w} + \varepsilon}{(\Delta_+\boldsymbol{w})^2 + (\Delta_-\boldsymbol{w})^2 + \varepsilon}$$

to avoid spurious oscillations near discontinuities. Here Δ_+ and Δ_- are respectively the forward and backward difference operators, ε is a small parameter that avoids division by zero in regions of null gradient and κ is a constant determining the spatial order of the scheme. For the present calculations $\kappa = \frac{1}{3}$, which produces a third-order spatial accuracy for the convective terms away from flow field discontinuities. The thin-layer diffusive fluxes are discretized using Gauss' theorem. Time discretization is achieved through a backward Euler time stepping. The resulting non-linear system for solution at time level $(n+1)$ is

$$\tilde{\boldsymbol{R}}(\boldsymbol{Q}^{n+1}) = \frac{\boldsymbol{Q}^{n+1} - \boldsymbol{Q}^n}{\Delta t} + \boldsymbol{R}(\boldsymbol{Q}^{n+1}) = \boldsymbol{0} \tag{3}$$

where $\boldsymbol{Q}$ is the state vector composed of the conservative variable vectors $\boldsymbol{q}_{i,j}$ and $\boldsymbol{R}(\boldsymbol{Q})$ is the vector constructed from $(i_{n-1}) \times (j_{n-1})$ vectors $\boldsymbol{r}_{i,j}$ defined at each grid cell by

$$\boldsymbol{r}_{i,j} = \tilde{e}_{i+1/2,j} - \tilde{e}_{i-1/2,j} + \tilde{\boldsymbol{f}}_{i,j+1/2} - \tilde{\boldsymbol{f}}_{i,j-1/2} + \boldsymbol{f}^{\mathrm{viscous}}_{i,j+1/2} - \boldsymbol{f}^{\mathrm{viscous}}_{i,j-1/2} \tag{4}$$

where $\tilde{\boldsymbol{e}}_{i\pm 1/2,j}$ is the Osher flux vector in the ξ-direction at the interface between the cells (i,j) and $(i\pm 1,j)$, and $\tilde{\boldsymbol{f}}_{i,j\pm 1/2}$ and $\boldsymbol{f}^{\text{viscous}}_{i,j\pm 1/2}$ are respectively the Osher and thin-layer viscous flux vectors in the η-direction at the interface between the cells (i,j) and $(i,j\pm 1)$. Note that the discretized scheme used here is continuously differentiable for laminar flows justifying the application of the Newton-like methods in a strict sense. Here, the term 'continuously differentiable' relates to the discretization operator $\tilde{\boldsymbol{R}}$ in Equation (3) rather than the flow field variables $\boldsymbol{Q}$, which can describe a flow field containing discontinuous features such as shocks. For the algebraic Baldwin–Lomax turbulence model used in the current study, a number of the features make the operator no longer continuously differentiable, including the functions for absolute, minimum and maximum values used in the model for switching between different formulations for the wake function calculation and the search for the peak of the vorticity moment for a turbulence length scale definition. Nevertheless, some of the two-equation models are actually continuously differentiable.

The non-linear system (3) is solved using a Newton method to produce the following iterative scheme at each time-step

$$\boldsymbol{J}^l[(\boldsymbol{Q}^{n+1})^{l+1}-(\boldsymbol{Q}^{n+1})^l] = -\tilde{\boldsymbol{R}}[(\boldsymbol{Q}^{n+1})^l] \tag{5}$$

where $\boldsymbol{J}$ is the Jacobian matrix, $\partial\tilde{\boldsymbol{R}}/\partial\boldsymbol{Q}^{n+1}$, of the non-linear system. The initial solution $(\boldsymbol{Q}^{n+1})^0$ is taken to be the converged solution at the previous time step, $\boldsymbol{Q}^n$. If the time step is reasonably small so that the solution at one time step is a good initial solution to the next time step, Newton's method should produce a quadratic convergence.

3.2. Linear solver

Some of the most effective iterative linear solvers for non-symmetric large sparse systems are based on Krylov sub-space methods. Given a linear equation

$$\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \tag{6}$$

a solution is sought in the form

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \langle \boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \dots, \boldsymbol{A}^{k-1}\boldsymbol{r}_0 \rangle \tag{7}$$

where $\langle \boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \dots, \boldsymbol{A}^{k-1}\boldsymbol{r}_0 \rangle$ is a k-dimensional (Krylov) sub-space, $\boldsymbol{x}_0$ is an initial guess and $\boldsymbol{r}_0 = \boldsymbol{r}(\boldsymbol{x}_0)$. One such Krylov sub-space method proposed by Saad and Schultz [25], the so-called Generalized Minimal RESidual (GMRES) algorithm, chooses a solution such that the Euclidean norm of the residual $\boldsymbol{r}(\boldsymbol{x}_k)$ is a global minimum. If one chooses k to be the size of the original vector space, then GMRES obtains the exact solution of the linear system (6). However, this requires a large amount of computer memory and CPU time, hence Saad and Schultz suggested a restarted form of GMRES, called GMRES(k), which uses a relatively small sub-space size k, evaluates $\boldsymbol{x}_k$ and then uses the new approximate solution as a new $\boldsymbol{x}_0$. This process is repeated until the converged solution is obtained.

In the present approach, the large sparse linear system at each Newton iteration (5) is solved using the GMRES(k) method. The Krylov sub-space size k is usually kept small because the amount of computer memory required is a linear function of k. Unlike some other iterative linear equation solvers, GMRES(k) in its original form always decreases the Euclidean norm of the residual every iteration, which provides a monotonic convergence and therefore robustness of the method. Indeed, our numerical testing in the matrix-free environment has shown that the GMRES method is much more robust than the CGS method. A similar observation was made by McHugh and Knoll [15], the robustness of GMRES being found especially useful when used in a matrix-free implementation.

In the present computations the initial guess $\boldsymbol{x}_0$ for GMRES(k) is taken to be the vector $\boldsymbol{Cb}$, where $\boldsymbol{C}$ is the AF preconditioning matrix (an approximation to $\boldsymbol{A}^{-1}$), and the iteration is considered converged if

$$\|\boldsymbol{b}-\boldsymbol{Ax}\| < \varepsilon_{\mathrm{TOL}}\|\boldsymbol{b}\| \tag{8}$$

for a given tolerance $\varepsilon_{\mathrm{TOL}} > 0$. Here and throughout '$\| \quad \|$' denotes the Euclidean norm of a vector.

3.3. *Matrix-free Newton/GMRES method*

In Krylov sub-space based methods, such as the GMRES method, the Jacobian matrix $\boldsymbol{J}$ is only required in the evaluation of the Jacobian matrix–vector products. This matrix–vector product can be approximated by the one sided directional derivative formula [10–12]

$$\boldsymbol{Jx} \approx \frac{\tilde{\boldsymbol{R}}(\boldsymbol{Q}+\varepsilon\boldsymbol{x}) - \tilde{\boldsymbol{R}}(\boldsymbol{Q})}{\varepsilon} \tag{9}$$

where ε is a carefully chosen small number.

Various authors have used different expressions for ε in their matrix-free formulations. For example, Brown and Saad [26] extended earlier work by Dennis and Schnabel [27] to derive the following formula for ε:

$$\varepsilon = \frac{\sigma^{1/2}}{\|\boldsymbol{x}\|^2}\max\{|\boldsymbol{Q}^T\boldsymbol{x}|, (\text{typ } \boldsymbol{Q})^T|\boldsymbol{x}|\}\ \text{sign}\{\boldsymbol{Q}^T\boldsymbol{x}\} \tag{10}$$

where typ $\boldsymbol{Q} = (\text{typ } Q_1, \text{typ } Q_2, \ldots, \text{typ } Q_n)^T$ is the column vector containing typical values of the components of $\boldsymbol{Q}$, $|\boldsymbol{x}| = (|x_1|, |x_2|, \ldots, |x_n|)$, and σ is a constant that has a similar order of magnitude as machine-zero. Choquet and Erhel [18] used a simplified form of Equation (10), where the first argument of the maximum function is assumed to be larger than the second. Johan [14], following Gill *et al.* [28] used

$$\varepsilon = 2\sqrt{\frac{\sigma}{\|\boldsymbol{DJx}\|}} \tag{11}$$

where the denominator represents the second-order derivatives of $\boldsymbol{R}(\boldsymbol{Q})$ evaluated at $\boldsymbol{x}$, and σ is as before. The main disadvantage with this formula is the determination of the second-order derivatives, which requires an extra function evaluation of $\boldsymbol{R}$ every time the formula is used. Johan, therefore, used it sparingly, by only updating ε after a fixed number of iterations. More recently, McHugh and Knoll [15] used a very simple form of ε, i.e.

$$\varepsilon = \sigma^{1/2}\left(1 + \frac{1}{n}\sum_{i=1}^{n} Q_i\right) \tag{12}$$

In the present double-precision computations, an effective choice for ε was found to be

$$\varepsilon = \sigma^{1/2}/\|\boldsymbol{x}\| \tag{13}$$

when $\|\boldsymbol{x}\| \neq 0$, and if $\|\boldsymbol{x}\| = 0$, the result of the matrix–vector product is set identically to zero. Here, σ is taken to be 10^{-14}. In the authors' experience, this formula performs just as well as that of Equation (10) but is more computationally efficient and has no ambiguously defined values such as the vector typ $\boldsymbol{Q}$.

The directional derivative in (9) may be replaced by a more accurate central difference formula, however, this would require an additional evaluation of $\boldsymbol{R}(\boldsymbol{Q})$, making the calculation slower. In the present computations, performed in double precision arithmetic, the accuracy of (9) was sufficient to produce a convergence rate very similar to the matrix method. However, the use of single precision arithmetic may require a central difference formula.

By using Equation (9), the evaluation of the Jacobian matrix is avoided. More importantly, it results in a huge saving of computer memory as there is no need to store the Jacobian matrix. However, the saving in memory is at the expense of more CPU time as each matrix–vector product requires an evaluation of the non-linear function.

3.4. Matrix-free preconditioner

The success of Krylov sub-space based methods depends upon the eigenvalue distribution of the matrix and therefore on an efficient preconditioner. A preconditioner has the effect of redistributing the eigenvalues of the matrix so that the convergence of the Krylov sub-space method is improved. Approximate eigenvalue analysis by Qin *et al.* [29] illustrated the effect of a block-diagonal preconditioning matrix with damping on the eigenvalues of a Jacobian matrix.

In the present matrix-free computations, the AF preconditioner for the GMRES method is implemented in a matrix-free way. If $\boldsymbol{R}(\boldsymbol{Q})$ is split in the generalized curvilinear co-ordinate directions ξ and η denoted by $\boldsymbol{R}_\xi$ and $\boldsymbol{R}_\eta$ respectively, the AF preconditioning matrix is then either $\boldsymbol{C}_\xi^{-1}\Delta t\boldsymbol{C}_\eta^{-1}$ or $\boldsymbol{C}_\eta^{-1}\Delta t\boldsymbol{C}_\xi^{-1}$, where

$$\boldsymbol{C}_\xi = \left[\boldsymbol{I} + \Delta t\,\frac{\partial \boldsymbol{R}_\xi}{\partial \boldsymbol{Q}}\right], \qquad \boldsymbol{C}_\eta = \left[\boldsymbol{I} + \Delta t\,\frac{\partial \boldsymbol{R}_\eta}{\partial \boldsymbol{Q}}\right] \tag{14a,b}$$

After a suitable ordering of columns, both of these matrices can be written as block-diagonal matrices, where each sub-matrix in the diagonal is itself a block-pentadiagonal matrix. The inverse of $\boldsymbol{C}_\xi$ is therefore found by inverting each sub-matrix successively using a suitable block-pentadiagonal matrix inversion routine; the present code uses lower–upper (LU) decomposition. Consequently, the components of each sub-matrix only need to be evaluated once immediately before the block-pentadiagonal matrix inversion routine and so all the blocks can use the same memory, equivalent to $5(i_{n-1})$ sub-blocks of 16 scalars. This is negligible compared with the storage required for $\boldsymbol{R}(\boldsymbol{Q})$ for a reasonable size grid. The matrix $\boldsymbol{C}_\eta$ is inverted in an analogous way.

For comparison purposes, this AF preconditioning has also been used on its own to give an iterative solution of the non-linear system (3) as an approximation to (5)

$$[\boldsymbol{C}_\xi(1/\Delta t)\boldsymbol{C}_\eta]^l[(\boldsymbol{Q}^{n+1})^{l+1}-(\boldsymbol{Q}^{n+1})^l]=-\tilde{\boldsymbol{R}}[(\boldsymbol{Q}^{n+1})^l] \tag{15}$$

This directional related factorization is similar to those used in the popular ADI methods, such as those proposed by Beam and Warming [30] and Briley and McDonald [31]. However, the factors used here are from a Jacobian matrix of the right-hand side, while the ADI methods use the Jacobians of the flux vectors through the linearization process.

3.5. Application to steady state problems

For steady state solution, instead of (3) we want to solve

$$\boldsymbol{R}(\boldsymbol{Q})=\boldsymbol{0} \tag{16}$$

However, as is common, a time-dependent approach (3) is used for its physical robustness. Nevertheless, as time accuracy is no longer a concern, a large local time step (Courant–Friedrich–Lewy (CFL) number) can be used to improve convergence to the steady state and no sub-iteration at each pseudo-time step is required. Starting $\boldsymbol{Q}$ from freestream values, the initial time steps must be small for the scheme to converge. However, the convergence rate can be improved by increasing the CFL number as the calculation proceeds by

$$\mathrm{CFL}=c_1-c_2\log_{10}\|\boldsymbol{R}\| \tag{17}$$

where c_1 and c_2 are constants and the residual $\|\boldsymbol{R}\|$ is normalized by its value at the first iteration. If the CFL number reaches infinity then the scheme becomes equivalent to the Newton iterative method for (16). However, for the present computations this limit can never be reached because the AF preconditioning deteriorates with increasing time step due to factorization error; hence, there is an upper limit on the CFL number for the approach to be effective. The result is a (globally) convergent damped Newton method for (16) without the need to generate an acceptable initial solution for the Newton iteration. Forsyth and Jiang [5] have devized a similar mechanism to determine the time step for steady state solutions based on the solution variation rather than the residuals.

4. NUMERICAL RESULTS

4.1. *Combined pitching–in-plane oscillations*

Investigation of the matrix-free Newton/GMRES method has been carried out for steady and unsteady transonic flow around an NACA0012 aerofoil. The unsteady problem corresponds to the flow that an aerofoil cross-section of a helicopter rotor blade experiences in forward flight. The helicopter forward speed is limited by dynamic shock-induced separation on the advancing side (high Mach number (M), low angle of attack (α)) and by dynamic stall on the retreating side (low M, high α). A two-dimensional approximation is justified because of the high aspect ratio of typical helicopter rotor blades.

The unsteady motion of the aerofoil in the present computation is described by

$$\begin{aligned} M_\infty &= M_0(1+\mu \sin \psi) \\ \alpha &= \alpha_0 - \Delta\alpha \sin \psi \end{aligned} \tag{18}$$

in which the mean angle of incidence α_0 is 5°, the variation in angle of incidence $\Delta\alpha$ about the quarter chord point is 3°, the mean Mach number M_0 is 0.536 and the advance ratio μ is 0.61. Correspondingly, the variations of the angle of incidence and Mach number are $3° \leq \alpha \leq 8°$ and $0.209 \leq M_\infty \leq 0.863$ respectively. The azimuth angle is defined by $\psi = (\omega c/U_\infty)t$, where $\omega = 0.185$ is the non-dimensional rotational speed, c is the aerofoil chord length and U_∞ the freestream velocity magnitude. The Reynolds numbers based on chord length (Re_c) are 1.75×10^6 for turbulent flow tests and 1.75×10^5 for laminar flow tests respectively. The freestream temperature is 288.17 K with an isothermal wall temperature condition. The problem is designed to study the unsteady dynamic shock movement and the shock induced separation on the advancing side of a helicopter rotor. The highest incidence on the retreating side of the rotor is not high enough to cause retreating blade dynamic stall.

Two algebraically generated body-fitted C-type grids have been used in the present study: (i) a fine mesh containing 258×96 nodes (200 of which lay upon the surface), and (ii) a coarse mesh with 153×48 nodes (100 grid points on the aerofoil surface). Both grids were generated using an algebraic mesh generation procedure that ensures that grid lines emanating from the aerofoil surface are locally normal. Grid clustering was performed near the aerofoil surface to resolve the details of the boundary layer. In the computational mesh, the maximum distance between the first cell centre and the aerofoil surface was around $0.000001c$ for the fine grid and $0.00001c$ for the coarse grid. The computational domain extends to a distance $20c$ from the aerofoil.

4.1.1. The steady state starting solution. A steady solution at mean conditions, i.e. $M_\infty = M_0$, $\alpha_\infty = \alpha_0$, is generated for starting the unsteady calculations. A Krylov sub-space of dimension 5 is used in the restarted GMRES method, i.e. GMRES(5), for both coarse and fine grid calculations. This is very small considering the size of the matrix: $(28\,576)^2$ for the coarse grid and $(97\,660)^2$ for the fine grid, and is sufficient due to the effective preconditioning and also to the damping effect of the unsteady term, both of which deteriorate in their effectiveness, as the time step becomes large. The parameters defining the CFL number in (17) are $c_1 = 10$ and

$c_2 = 22$. Figure 1 shows for the coarse grid tests the convergence with respect to CPU time of the matrix-free calculation compared with the matrix calculation, the AF solution and the explicit calculation for the turbulent steady state problem. One unit of CPU time is defined throughout this paper as the CPU time required for the evaluation of one right-hand side vector $\boldsymbol{R}(\boldsymbol{Q})$. Therefore, the number of the CPU units required by a method represents an equivalence of corresponding number of required function evaluations of $\boldsymbol{R}(\boldsymbol{Q})$. It is not surprising to see that all three implicit methods outperform the explicit method by a huge margin, although the latter has incorporated a four-stage Runge–Kutta method with local time stepping for better convergence. Similar convergence rates can be seen for the early stage for the three implicit methods but clear differences can be identified for the later stage of the convergence. An option of zero GMRES(k) iteration, i.e. $n = 0$, has been incorporated in the Newton-like method solvers for solution initialization, which explain the similarity in the early stage of convergence. As far as the convergence rate against iteration number is concerned, the matrix-free method has maintained the convergence of the matrix method, indicating a successful implementation of the matrix-free method regarding the choice of ε in (9). However, the matrix-free method costs more per iteration as compared with the matrix method as will be explained later and the overall computing time ends up longer than that of the matrix method. On the other hand, the AF implicit method is a lot cheaper per iteration than the two Newton-like methods but due to its much slower convergence the overall computational time is significantly longer.

The convergence of the matrix-free solver and the AF solver on the fine grid are shown in Figure 2. It is seen that the convergence of the matrix-free solver on the fine grid is consistent

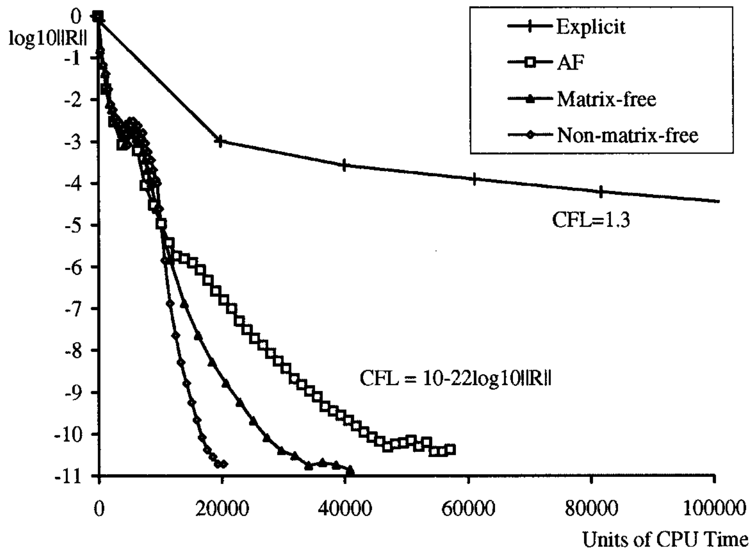

Figure 1. Convergence for steady state solution.

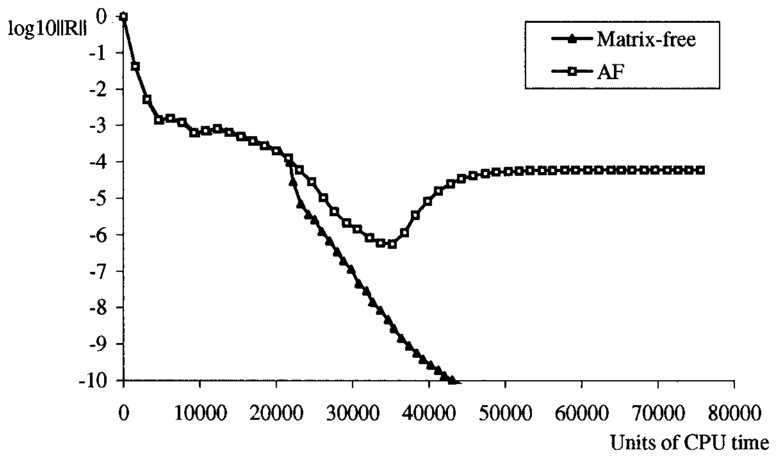

Figure 2. Comparison of convergence of matrix-free method with AF method for steady state solution on a fine grid.

with that achieved on the coarse grid. However, the convergence for the AF solver deteriorated on the finer grid and stagnated at about four orders of reduction in the residual, indicating a stronger dependency of the convergence on the problem size for AF scheme.

Table I gives a detailed breakdown of the CPU times required by each part of the calculation for both the matrix-free and matrix methods. Since the matrix-free method maintains the convergence rate of the matrix method, the CPU counts per iteration give a good indication of the computing efficiency of the two Newton-like methods. In the context of Table I, the AF implicit method and the explicit method only require 5.4 and 4 CPU units per iteration respectively; the poorer overall performances of these methods result from slower convergence. The efficiency of implicit methods compared with the explicit Runge–Kutta method is strongly dependent upon the maximum time-step allowable in each method. While a theoretical approximate CFL condition exists for the explicit method, an optimal time step is much more difficult to find for the implicit method with an approximate solution to the linear system.

Table I. CPU count analysis for matrix-free and matrix implicit solvers[a].

Operation	Matrix-free method	Matrix method
Right-hand side evaluation	1	1
Evaluation of Jacobian and preconditioning matrix	—	4.4
Jacobian–vector product	1	0.4
Preconditioner–vector product	4.4	0.76
Total CPU units for one Newton iteration of n GMRES(k) sub-iterations	$1+5.4n(k+1)$	$5.4+1.16n(k+1)$

[a] CPU time for evaluating $\boldsymbol{R}(\boldsymbol{Q})$ is 1 unit of time.

Different ways in obtaining the Jacobian matrix play an important role in the balance between the matrix and matrix-free methods. In the present matrix calculations, the analytical Jacobian matrix was derived by use of chain rules. Use of the analytical Jacobian can be seen to be much more computationally efficient than using a Jacobian derived by finite differences [1,2], although the latter is much easier to code. Even with exploitation of the matrix's sparsity pattern, finite differences would require 100 evaluations of the right-hand side vector for the current stencil.

It can be seen from the last row of Table I, that for all n and k, the matrix-free approach employed here will always be slower than the matrix approach, and the ratio of CPU times is least when $n = k = 1$, which are of course not practical.

For the present solver, one evaluation of the right-hand side costs much more than an ordinary matrix–vector product. It may be possible that a Navier–Stokes solver exists for which one evaluation of the residual takes less CPU time than an ordinary matrix–vector product. Then using a matrix-free product with a stored preconditioning matrix will take less CPU time than the corresponding implicit method with an ordinary matrix–vector product.

Alternatively, for steady state solutions, Knoll *et al.* [17] have found a matrix-free method quicker than a matrix method, by exploiting the relatively high set-up cost per iteration (i.e. per time step or damped-Newton iteration for steady state cases) required by the latter method. In their matrix-free approach, a preconditioner is evaluated, stored and used for the next $(m-1)$ iterations.

From the data in Table I, the matrix-free method could be quicker than the present matrix method if we stored the preconditioning matrix and froze it for m steps under the following condition:

$$m[5.4 + 1.16n(k+1)] > 4.4 + m[1 + 1.76n(k+1)] \tag{19}$$

which is only satisfied for the eight cases

$$(k, n, \text{Min } m) \in \{(1, 1, 2), (2, 1, 2), (3, 1, 3), (4, 1, 4), (5, 1, 6), (6, 1, 7), (1, 2, 3), (2, 2, 6)\} \tag{20}$$

In their computations, Knoll *et al.* [17] were able to choose much larger values of n and k keeping m small, because the components of the Jacobian matrix are evaluated by finite differences, which makes the set-up CPU time per iteration much longer than in the present matrix code.

Table I also shows that most of the extra CPU time required for the matrix-free method originates from the multiplication of an arbitrary vector by the preconditioner. This is not surprising, since the inverse of a block-pentadiagonal matrix has to be calculated in addition to the evaluation of the AF elements.

A significant advantage of the present matrix-free method is that both the Jacobian and the preconditioning matrices are not stored, which accounts for a huge saving in the required memory. For the two-dimensional cases considered here, these two matrices account for about 25 Mb for a 153×48 grid and 82 Mb for a 258×96 grid. A detailed breakdown of relative memory requirements for the matrix-free, matrix, AF and explicit solvers is given in Table II;

Table II. Memory usage analysis for steady implicit and explicit solvers (unsteady implicit codes in brackets)[a].

Piece of code requiring storage	Units of memory
Grid and its derivatives, cell areas, Δt, flow variables, viscosities, $\boldsymbol{R}(\boldsymbol{Q})$	5 (6.5)
GMRES(k)	$k+2$
Matrix-free product	1
Jacobian and preconditioning matrices	80
Other implicit overheads	2
Explicit solver	5 (6.5)
AF solver	5 (6.5)
Matrix-free implicit solver	$k+10$ ($k+11.5$)
Matrix implicit solver	$k+89$ ($k+90.5$)

[a] Storing $\boldsymbol{R}(\boldsymbol{Q})$ is 1 unit of memory.

the memory requirement for the corresponding unsteady algorithm is similar. From a storage point of view, a matrix-free preconditioner is as important as a matrix-free non-linear solver, since a matrix preconditioner will add 40 units to the memory required by the matrix-free solver.

In the matrix implicit solver, although the present discretization of the thin-layer Navier–Stokes equations yields a nine point stencil, the Jacobian and the preconditioning matrix are each stored in 4×10 units of memory since the block-diagonal term is stored in two parts, one for each co-ordinate direction. This is necessary to efficiently calculate the AF preconditioner.

4.1.2. Convergence for the unsteady cases. For the unsteady cases, the calculations described in the present paper use 1200 time steps per revolution with a GMRES(10) linear solver at each Newton iteration unless otherwise specified. The corresponding maximum CFL number is typically 17000. Both laminar and turbulence cases were tested.

For laminar cases, the implicit operator is fully consistent with the right-hand side non-linear system and, therefore, quadratic convergence is expected if the convergence tolerance is set properly. Figure 3 shows the rates of convergence with respect to iteration number for the first five time steps of both the unsteady matrix-free and matrix laminar calculations. The convergence tolerance for the linear system is taken to be $\varepsilon_{\mathrm{TOL}} = 0.001$. Except for the first time step, the matrix-free version has shown to reproduce the near quadratic convergence of the matrix version. However, when the residual approaches machine-zero, the rounding error in both approaches becomes significant, making convergence level out.

For the first time step, convergence curves for the two approaches separate after the third iteration due to the effect of the finite difference approximation in the matrix-free implementation as shown in Figure 3. Here, the initial solution is the steady state solution, i.e. the term $\Delta\boldsymbol{Q}/\Delta t$, is zero in the residual vector. Hence, the norm of the residual is smaller than in the rest of the calculation, resulting in the finite difference error becoming significant after only three iterations. The curve also flattens earlier.

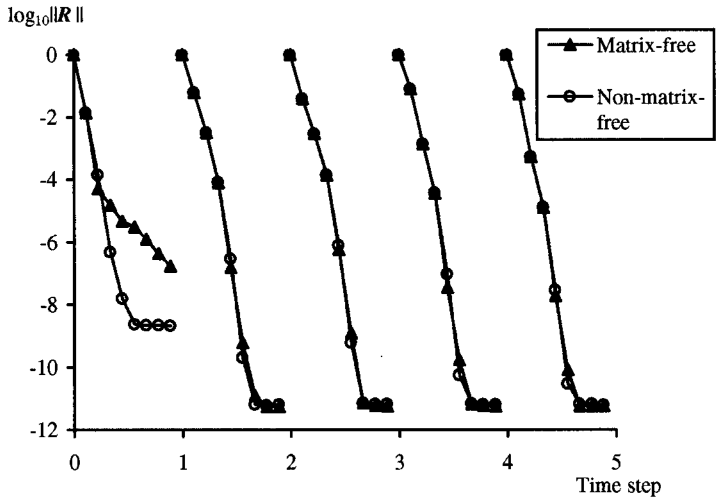

Figure 3. First five Newton iterations for unsteady solution.

Figure 4 shows the convergence of the Newton iterations at the eleventh time step with different convergence tolerances $\varepsilon_{\mathrm{TOL}}$, for both of the implicit methods. Quadratic convergence is clearly obtained for the first three iterations in both calculations when $\varepsilon_{\mathrm{TOL}} = 0.0001$. It is

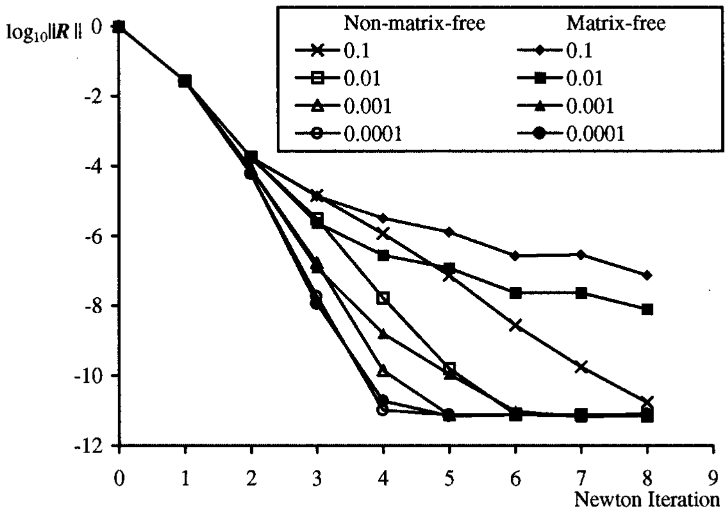

Figure 4. Newton iterations at eleventh time step with different tolerances for convergence of linear system.

reasonable that the fourth iteration is not quadratic (i.e. $\approx 10^{-16}$) since the rounding errors due to machine-zero prevent convergence from further reduction of the residuals under $\approx 10^{-11}$. With increasing $\varepsilon_{\mathrm{TOL}}$, the convergence leaves the quadratic curve at the third iteration and becomes superlinear, then linear at decreasing orders. Both matrix-free and matrix methods produce identical convergence until they reach the fourth iteration where the curves clearly separate indicating the matrix-free finite difference approximation becomes significant.

For all the cases considered in Figure 4, quadratic convergence occurs for the first two Newton-steps. Therefore, in computations with smaller $\varepsilon_{\mathrm{TOL}}$, there is excess convergence of the linear system for the first few steps, wasting valuable CPU time. Therefore, to obtain the most efficient quadratic convergence at each time step, $\varepsilon_{\mathrm{TOL}}$ should decrease with each Newton step to a minimum value at the order where the matrix-free finite difference approximation error becomes significant. From theory presented by Brown [13], a reasonable convergence tolerance should be proportional to $\|\boldsymbol{b}\|$.

A comparison of the CPU times taken for the Newton method with different $\varepsilon_{\mathrm{TOL}}$ in one typical time step (the eleventh) is shown in Figure 5. From the figure it is clear that if machine zero accuracy is required then quadratic convergence appears to be the most efficient in CPU time for the iterative solver used (GMRES(10)). Further by taking

$$\varepsilon_{\mathrm{TOL}} = \mathrm{Max}\{c_3\|\boldsymbol{b}\|, c_4\} \tag{21}$$

where $c_3 = 1$, $c_4 = 0.0001$, quadratic convergence is maintained with even less CPU time required for the first two iterations than that for the curve corresponding to $\varepsilon_{\mathrm{TOL}} = 0.0001$. However, for practical applications, machine-zero accuracy may not be required for

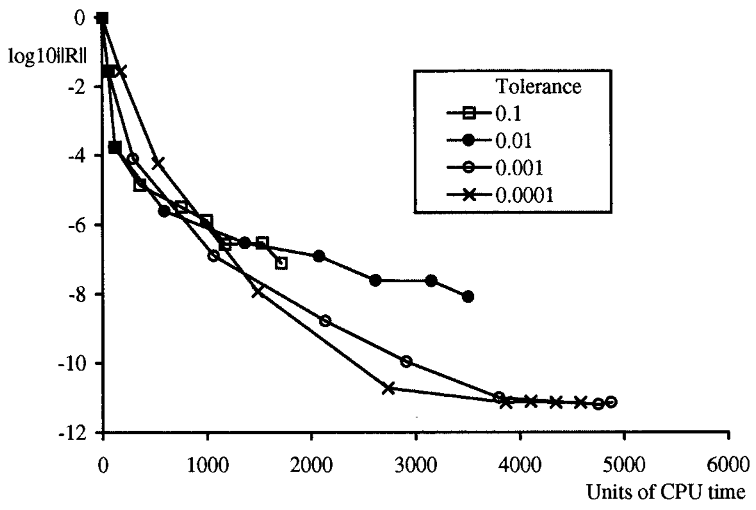

Figure 5. Matrix-free Newton iterations at eleventh time step with different tolerances for convergence of linear system.

engineering answers and $\varepsilon_{\mathrm{TOL}} = 0.1$ may suffice for a quadratic convergence in three iterations with a fourth-order reduction in the residuals using less than 100 CPU units for the matrix-free method (Figure 5).

To enable quadratic convergence, several iterations of GMRES(10) are required to solve most of the linear system of equations. Therefore, the sum-part containing the term *nk* in Table I dominates the overall CPU time, which makes the CPU time for the matrix-free method much longer than that of the matrix method.

For turbulent applications, the situation changes concerning quadratic convergence. In the present code, the Baldwin–Lomax algebraic turbulence model based on the mixing length hypothesis is incorporated. This actually causes difficulty in a strict application of Newton-like methods. One problem is due to the global operation of finding the maximum moment of vorticity in the boundary layer. The stencil is no longer compact and the Jacobian matrix becomes a full matrix instead of a sparse one. Additionally, the model is not continuously differentiable so that it needs to be smoothed before a Newton-like method can be applied. One remedy is to ignore the derivatives of the turbulent viscosity in the left-hand side of (5).

Note that using this approach, a non-iterative implicit method, see e.g. Reference [21], obtained by linearizing $\boldsymbol{R}(\boldsymbol{Q}^{n+1})$ in (3) does not fully employ the turbulence model in a strict sense for unsteady flows, since some terms are omitted in the left-hand side. However, in the Newton-like method employed here, despite the absence of these terms the turbulence model is fully accounted for in the final solution through the sub-iterations.

Figure 6 presents the convergence at one time step for the unsteady Navier–Stokes solution for both laminar and turbulent flow calculations. The laminar flow exhibits the quadratic-like convergence behaviour as described above. However, for the turbulent calculation, the convergence is reduced to a linear rate due to the inconsistency between the Jacobian used (no turbulence derivatives) and the non-linear system (including turbulence variations) solved.

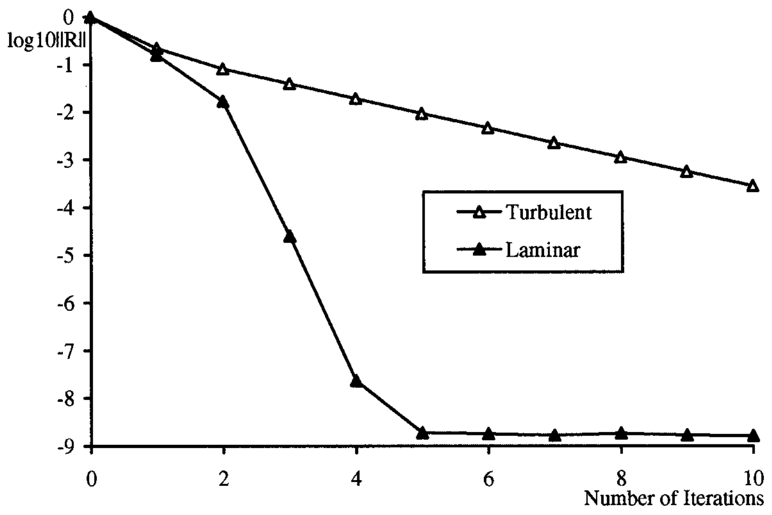

Figure 6. Newton iterations for unsteady solution at one time step.

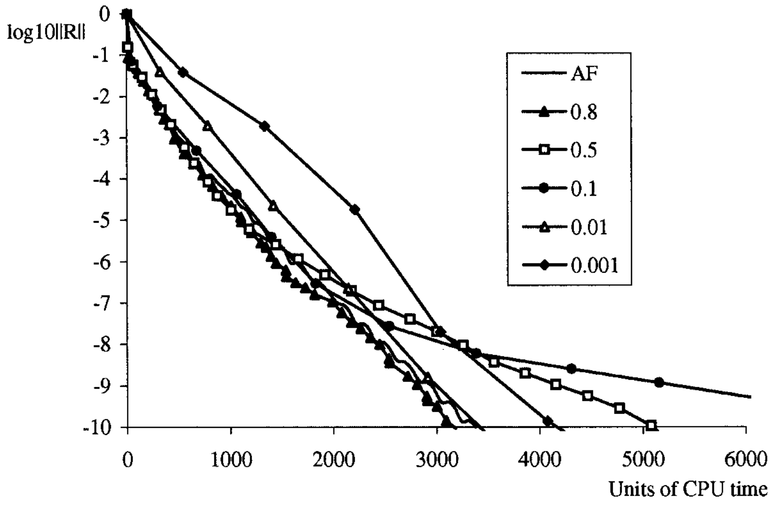

Figure 7. Comparison of convergence of matrix-free methods with different GMRES tolerances and AF method for laminar unsteady flow.

Figure 7 compares the AF method and the matrix-free method using GMRES(5) with different tolerances $\varepsilon_{\mathrm{TOL}}$ for a typical time step in a laminar unsteady calculation. It can be seen that the AF method performs quite well in this situation, competing with the fastest matrix-free method. The highest value of $\varepsilon_{\mathrm{TOL}}$ shown gives the best performance with only one GMRES(5) sub-iteration being required. Figures 5 and 7 together seems to indicate that, for efficiency one should either use a high tolerance to limit the GMRES sub-iteration or to use a properly chosen small one to achieve a quadratic convergence to limit the Newton iterations. The worst efficiency occurs in the middle range of tolerances for which neither GMRES sub-iteration nor Newton iteration number is small.

A similar comparison for the turbulent unsteady case is shown in Figure 8. Here the AF solver failed to converge below three orders and the matrix-free solver considerably outperforms the AF solver. Figure 8 also shows that for this case there is no point using small tolerances since quadratic convergence is not achievable; a relatively high value of $\varepsilon_{\mathrm{TOL}}$ therefore limits the inner GMRES sub-iteration number producing best overall CPU time efficiency.

4.1.3. Discussion of the unsteady solution. For the present unsteady case described by (18), it was found that, starting from a converged steady state solution, a periodic unsteady solution was established after one cycle of the aerofoil motion. The unsteady results are shown in Figure 9 as instantaneous Mach number contours at selected azimuth angles ψ. Here, $\psi = 90°$ corresponds to the highest Mach number and the lowest incidence on the advancing side, and $\psi = 270°$ corresponds to the lowest Mach number and highest incidence on the retreating side.

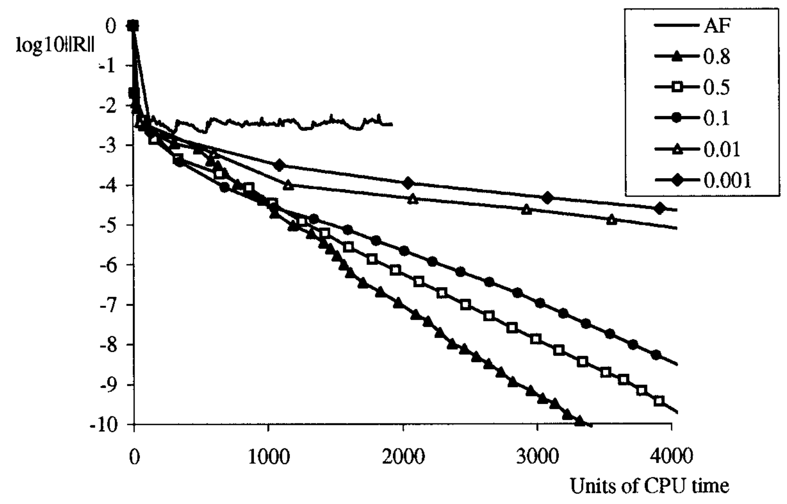

Figure 8. Comparison of convergence of matrix-free methods with different GMRES tolerances and AF method for turbulent unsteady flow.

In the present case, the dominant dynamic feature is the shock movement and shock induced separation on the advancing side. Therefore, only results on the advancing side are shown in Figure 9.

At $\psi = 40°$, illustrated by Figure 9(a), a weak shock wave begins to form on the upper side of the aerofoil. This shock wave becomes stronger and moves downstream as the Mach number increases and angle of incidence decreases (see Figure 9(b)). By the time the azimuth angle reaches 90° at the maximum Mach number and minimum angle of incidence, illustrated by Figure 9(c), the shock wave on the upper side of the aerofoil surface induces boundary layer separation and then moves back upstream (see Figure 9(d)). Simultaneously, on the lower side of the aerofoil a weak shock forms and moves downstream as the Mach number decreases and incidence angle increases. Prior to the azimuth angle reaching $\psi = 150°$ (Figure 9(e)), the shock on the upper side of the aerofoil has disappeared close to the leading edge and a separation vortex starts moving downstream, the lower shock has become stronger and moves back upstream. Near the end of the advancing side (around $\psi = 170°$, Figure 9(f)) both shocks disappear and the flow once more becomes fully attached to the aerofoil.

The unsteady dynamic effects can be clearly seen by comparing the azimuth angles at which both the Mach number and incidence angle are the same. For example, comparison can be made between Figure 9(b) and (d) at the azimuth angles of $\psi = 60°$ and $\psi = 120°$ respectively. The former exhibits a shock on the upper side of the aerofoil, while the latter shows a shock at a more upstream position and a separated flow region on the upper side in addition to a shock wave on the lower side of the aerofoil. These significantly different figures indicate strong dynamic effects on the advancing side of the rotor blade.

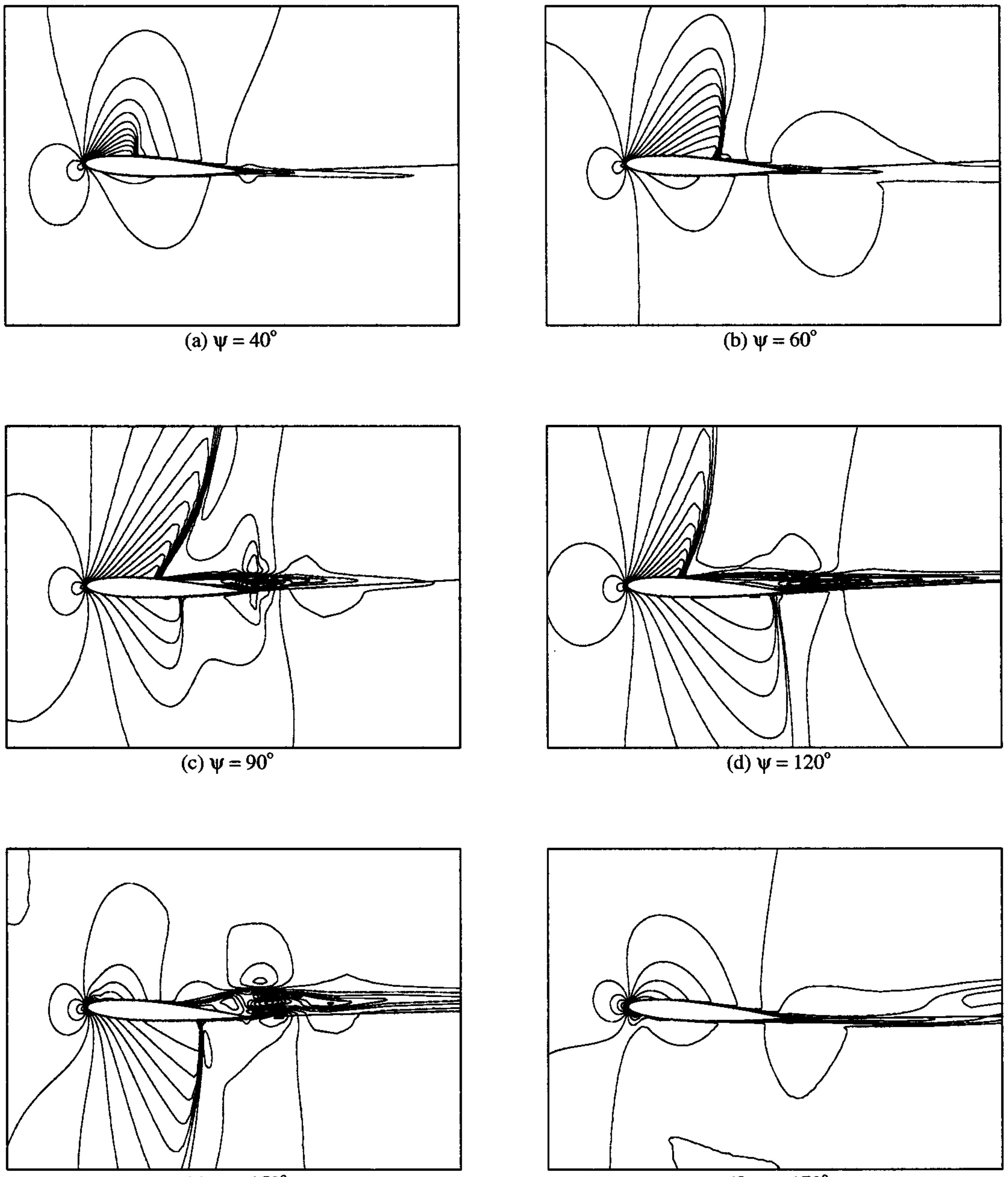

Figure 9. Mach number contours from unsteady turbulent solution for combined translation–pitch oscillation.

4.2. Flow solver validation

Validation of the above unsteady calculation directly with experimental data is extremely difficult due to the non-existence of corresponding two-dimensional experimental work. Consequently, two validation cases for the NACA0012 aerofoil are considered; the first for in-plane motion only, the second for pitching motion only. Both validation calculations are performed on the same fine grid as described above and use the same methodology and turbulence model.

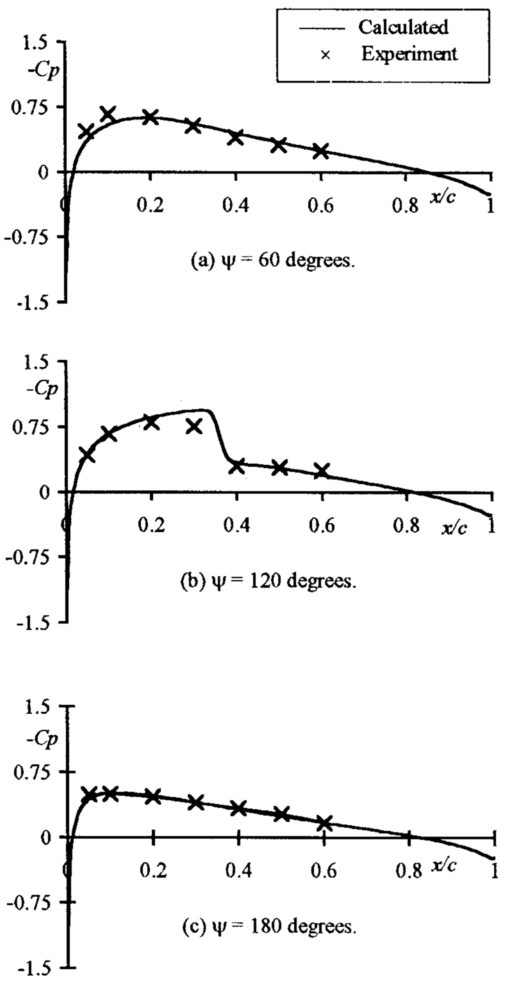

Figure 10. Comparison with experimental data for in-plane oscillation.

In Figure 10, the results of unsteady Navier–Stokes calculations performed around an aerofoil undergoing pure in-plane oscillations, i.e. a periodic incoming flow Mach number ($M_0 = 0.536$, $\mu = 0.61$, $\omega = 0.185$ in Equation (18)) at zero incidence ($\alpha = 0$, $\Delta\alpha = 0$) are compared with corresponding three-dimensional rotor measurements made at ONERA as reported by Lerat and Sides [32]. Reasonably good agreement between the unsteady computation and the experimental data is observed in the surface pressure comparison.

Landon [33] has presented experimental measurements for a number of standard configurations that have been established for use in the development and validation of numerical methods for prediction of aeroelastic responses and their associated unsteady aerodynamics. Case CT1 concerns the flow ($M_0 = 0.60$, $\mu = 0$, $Re_c = 4.8 \times 10^6$) around a rigid NACA0012 aerofoil oscillating in pitch about its quarter chord point with $\alpha_0 = 2.89°$, $\Delta\alpha = 2.41°$, and $\omega = 0.1616$. For this test case a periodic flow exhibiting Tidjeman Type B [34] shock motion was computed. Starting from a steady state flow at the mean flow conditions, three complete cycles of the unsteady motion were computed, the resulting normal force and pitching moment coefficients are plotted in Figures 11 and 12 along with corresponding experimental measurements. The figures suggest that a well-defined periodic flow is established within the first cycle of the aerofoil motion. For this case, 720 time steps were used to resolve each period of the motion; this corresponds to a maximum CFL number in the solution domain of around 50000. Ten Newton iterations were performed at each time step. The linear system arising at each time step was solved using GMRES(10), a tolerance of 0.05 was employed, which could typically be achieved within one or two steps of the method. Comparison between the computed and measured data is considered satisfactory and furthermore, the present results compare favourably with those obtained by other authors [21,35].

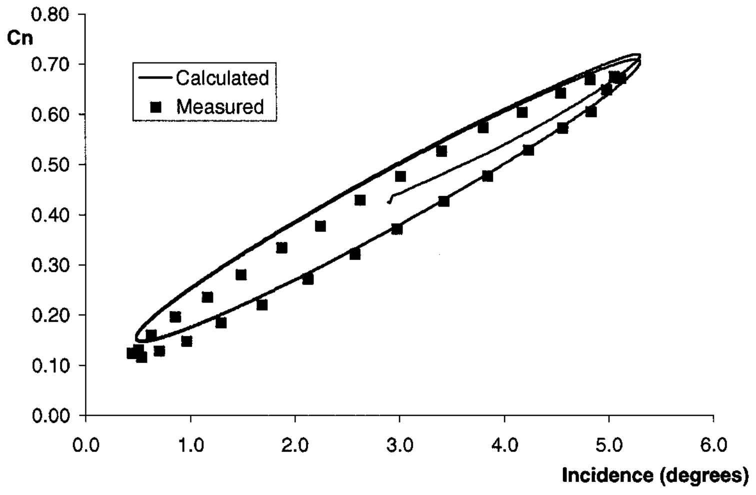

Figure 11. Normal force comparison for AGARD CT1 case.

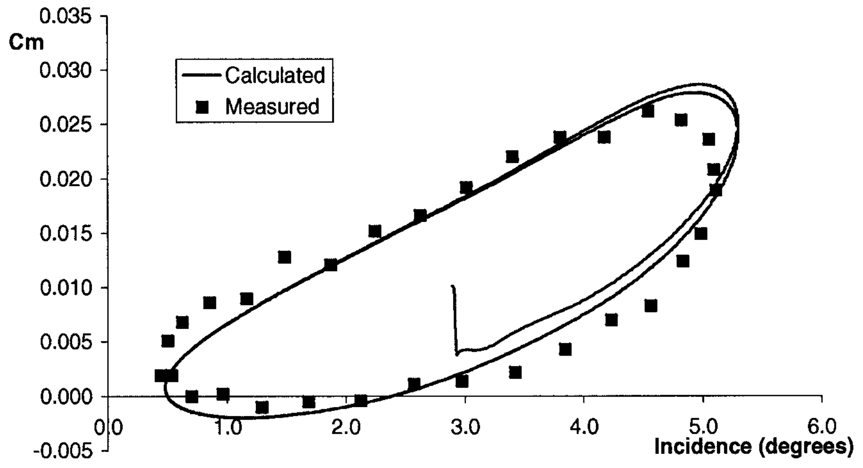

Figure 12. Pitching moment comparison for AGARD CT1 case.

5. CONCLUSION

A low-memory, fully matrix-free preconditioned Newton/GMRES method has been presented and studied for unsteady and steady Navier–Stokes solutions. It has a similar convergence rate to the corresponding matrix version but uses significantly less computer memory since neither the Jacobian matrix nor the preconditioning matrix needs to be stored. The expense for this memory saving is some extra CPU time required for each iteration.

It has been found that the tolerance on the GMRES iterations plays an important role in the overall performance of the matrix-free method depending on the consistency of the implicit operator and the right-hand side. If the consistency can be guaranteed, such as in the laminar cases presented, the performance is the worst for a medium GMRES tolerance. However, a better performance can be achieved by either reducing the tolerance to give quadratic or near quadratic convergence or increasing it to limit the GMRES iteration to a very small number. For the turbulent cases, consistency can not be achieved for the current algebraic model and therefore a strategy of using relatively large GMRES tolerances is proposed to achieve a good efficiency. In both the steady and the unsteady turbulent test cases, the present matrix-free method outperforms the conventional AF implicit method in overall performance. Its convergence is consistent for different problem sizes tested and exhibits less sensitivity on the problem size compared with the AF method.

An exploratory unsteady turbulent Navier–Stokes solution of the combined translation–pitch oscillations has been carried out for helicopter rotor blade flow interactions. The dynamic features such as the shock movement and the shock-induced separation are clearly revealed in the flow field evolution, providing useful information for the understanding of the flow problem. Validation has been limited to simpler cases for a pitching aerofoil and an in-plane oscillation case.

Although the current study has been limited to two-dimensional unsteady problems, the matrix-free approximate factorization preconditioner can be extended to three-dimensional applications. A direct application of the dimensional splitting may be problematic due to the stability problem with three-dimensional splitting and the cubic order factorization errors, both of which may possibly make the preconditioner a lot less effective. In this regard, the two-factor schemes, such as that proposed in Reference [36], seem to be good candidates.

ACKNOWLEDGMENTS

The research reported in this paper was funded by grants from the Leverhulme Trust (F509C) and the Engineering and Physical Science Research Council (GR/K31664). The authors also acknowledge useful discussions with Bryan Richards and Ken Badcock of Glasgow University and John Perry, Rob Harrison and Tassos Kokkalis of GKN Westland Helicopters Ltd.

REFERENCES

1. Qin N, Richards BE. Sparse quasi-Newton methods for high resolution schemes. *Notes on Numerical Fluid Mechanics* 1988; **20**: 310–317.
2. Whitfield DL, Taylor LK. Discretized Newton-relaxation solution of high resolution flux-difference split schemes. AIAA Paper 91-1539, 1991.
3. Orkwis PD. Comparison of Newton's and quasi-Newton's method solvers for the Navier–Stokes equations. *AIAA Journal* 1993; **31**: 832–836.
4. Eisenstat SC, Walker HF. Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing* 1996; **17**(1): 16–32.
5. Forsyth PA, Jiang H. Non-linear iteration methods for high speed laminar compressible Navier–Stokes equations. *Computers and Fluids* 1997; **26**(3): 249–268.
6. Qin N, Xu X, Richards BE. Newton-like methods for Navier–Stokes solution. In *Proceedings of the First European Computational Fluid Dynamics Conference*, vol. 1, Hirsch Ch, Periaux J, Kordulla W (eds), 1992; 117–124.
7. Xu X, Qin N, Richards BE. GMRES—a new parallelizable iterative solver for large sparse on-symmetric linear systems arising from CFD. *International Journal for Numerical Methods in Fluids* 1992; **15**: 613–623.
8. Cai XC, Gropp WD, Keyes DE, Tidriri MD. Newton–Krylov–Schwarz methods in CFD. In *Numerical Methods for Navier–Stokes Equations*, Hebeker F, Rannacher R, Wittum G (eds). Vieweg: Brunswick, 1994; 17–30.
9. McHugh PR, Knoll DA, Keyes DE. Schwarz-preconditoned Newton–Krylov algorithm for low speed combustion problems. AIAA Paper 96-0911, 1996.
10. Gear CW, Saad Y. Iterative solution of linear equations in ODE codes. *SIAM Journal on Scientific and Statistical Computing* 1983; **4**: 583–601.
11. Chan TF, Jackson KR. Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms. *SIAM Journal on Scientific and Statistical Computing* 1984; **5**: 533–542.
12. Brown PN, Hindmarsh AC. Matrix-free methods for stiff systems of ODEs. *SIAM Journal on Numerical Analysis* 1986; **23**: 610–638.
13. Brown PN. A local convergence theory for combined inexact-Newton/finite difference projection methods. *SIAM Journal on Numerical Analysis* 1987; **24**: 407–434.
14. Johan J. Data parallel finite element techniques for large scale computational fluid dynamics. PhD thesis, Department of Mechanical Engineering, Stanford University, U.S.A, 1992 (Chapter 2).
15. McHugh PR, Knoll DA. Comparison of standard and matrix-free implementations of several Newton–Krylov solvers. *AIAA Journal* 1994; **32**: 2394–2400.
16. McHugh PR, Knoll DA. Inexact Newton's method solutions to the incompressible Navier–Stokes and energy equations using standard and matrix-free implementations. AIAA Paper 93-3332, 1993.
17. Knoll DA, McHugh PR, Keyes DE. Newton–Krylov methods for low Mach number combustion. *AIAA Journal* 1996; **34**: 961–967.
18. Choquet R, Erhel J. Newton–GMRES algorithm applied to compressible flows. *International Journal for Numerical Methods in Fluids* 1996; **23**: 177–190.

19. Nielsen EJ, Anderson WK, Walters RW, Keyes DE. Application of Newton–Krylov methodology to a three-dimensional unstructured Euler code. AIAA Paper 95-1733, 1995.
20. Badcock KJ. An efficient unfactored implicit method for unsteady airfoil flows. Glasgow University Technical Report, G.U. Aero 9313, 1993.
21. Badcock KJ, Gaitonde AL. An unfactored implicit moving mesh method for the two-dimensional unsteady N–S equations. *International Journal for Numerical Methods in Fluids* 1996; **23**: 607–631.
22. Baldwin B, Lomax H. Thin layer approximation and algebraic model for separated turbulent flows. AIAA Paper 78-257, 1978.
23. Osher S, Solomon F. Upwind difference schemes for hyperbolic systems of conservative laws. *Mathematics of Computation* 1992; **38**: 339–374.
24. Van Leer B. Towards the ultimate conservative difference scheme V: a second-order sequel to Godunov's method. *Journal of Computational Physics* 1979; **32**: 101–136.
25. Saad Y, Schultz MH. GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1986; **7**: 856–869.
26. Brown PN, Saad Y. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing* 1990; **11**: 450–481.
27. Dennis JE, Schnabel RB. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall: Englewood Cliffs, NJ, 1983.
28. Gill PE, Murray W, Wright MH. *Practical Optimization*. Academic Press: New York, 1981.
29. Qin N, Xu X, Richards BE. Newton-like methods for fast high resolution simulation of hypersonic viscous flows. *Computing Systems in Engineering* 1992; **3**: 429–435.
30. Beam R, Warming RF. An implicit factored scheme for the compressible Navier–Stokes equations. *AIAA Journal* 1978; **16**(4): 393–402.
31. Briley WR, McDonald H. Solution of the multidimensional Navier–Stokes equations by a generalized implicit method. *Journal of Computational Physics* 1977; **24**(4): 372–397.
32. Lerat A, Sides J. Numerical simulation of unsteady transonic flows using the Euler equations in integral form. ONERA Report, TP 1979-10, 1979.
33. Landon RH. NACA0012 oscillatory and transient pitching. Paper 3, AGARD R-702, 1982.
34. Tidjeman H. Investigation of the transonic flow around oscillating aerofoils. NLR Report TR 77090U, 1977.
35. Gaitonde AL, Jones DP, Fiddes SP. A 2D Navier–Stokes method for unsteady compressible flow calculations on moving meshes. *Aeronautical Journal* 1998; **102**(1012): 89–97.
36. Yoon S, Kwak D. Implicit Navier–Stokes solver for three-dimensional compressible flows. *AIAA Journal* 1992; **30**(11): 2653–2659.