INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 19, 439–455 (1994)

FULLY COUPLED FINITE VOLUME SOLUTIONS OF THE INCOMPRESSIBLE NAVIER–STOKES AND ENERGY EQUATIONS USING AN INEXACT NEWTON METHOD

PAUL R. McHUGH AND DANA A. KNOLL

Computational Fluid Dynamics Unit, Idaho National Engineering Laboratory, EG&G Idaho, Inc., Idaho Falls, ID 83415-3895, U.S.A.

SUMMARY

An inexact Newton method is used to solve the steady, incompressible Navier–Stokes and energy equations. Finite volume differencing is employed on a staggered grid using the power law scheme of Patankar. Natural convection in an enclosed cavity is studied as the model problem. Two conjugate-gradient-like algorithms based upon the Lanczos biorthogonalization procedure are used to solve the linear systems arising on each Newton iteration. The first conjugate-gradient-like algorithm is the transpose-free quasi-minimal residual algorithm (TFQMR) and the second is the conjugate gradients squared algorithm (CGS). Incomplete lower–upper (ILU) factorization of the Jacobian matrix is used as a right preconditioner. The performance of the Newton-TFQMR algorithm is studied with regard to different choices for the TFQMR convergence criteria and the amount of fill-in allowed in the ILU factorization. Performance data are compared with results using the Newton-CGS algorithm and previous results using LINPACK banded Gaussian elimination (direct-Newton). The inexact Newton algorithms were found to be CPU competetive with the direct-Newton algorithm for the model problem considered. Among the inexact Newton algorithms, Newton-CGS outperformed Newton-TFQMR with regard to CPU time but was less robust because of the sometimes erratic CGS convergence behaviour.

KEY WORDS Incompressible Navier–Stokes Newton's method Conjugate gradient

1. INTRODUCTION

The use of robust, fully implicit algorithms to solve the Navier–Stokes equations has grown in popularity in both finite element and finite volume applications owing to the rapid advances in computer speed and available memory.[1–9] Direct matrix solution methods are commonly used, employing either a banded solver or the frontal method.[10] Research by the finite volume community has demonstrated the robustness of fully implicit methods compared with the more common segregated solution procedures such as SIMPLE.[11] This improved robustness is due to the implicit treatment of the velocity–pressure–temperature coupling.

The main drawback of direct Newton methods is the large memory required to factor the Jacobian matrix. This drawback has been countered with advances in sparse matrix iterative solution algorithms. Specifically, the development of efficient conjugate-gradient-like algorithms for the solution of non-symmetric, non-positive definite linear systems[12,13] has enabled the implementation of 'in-core', multidimensional, fully implicit Newton method solutions for the Navier–Stokes and energy equations. Since the use of an iterative solver does not require the exact solution of the linear system, the resulting algorithm has been labelled 'inexact' Newton method.[14,15] This feature is advantageous in the sense that the tolerance of the linear equation

CCC 0271–2091/94/170439–17

Received 13 May 1993
Revised 29 December 1993

solve can be relaxed when far from the true solution and tightened as the true true solution is approached.

'Out-of-core' matrix solvers such as the frontal method can also be used to handle large Jacobian matrices that exceed available 'in-core' memory. Only a limited number of matrix entries (contributing to the active 'frontal matrix') must be stored in core memory, yet partial or full pivoting is possible in the frontal matrix.[3,10] Einset and Jensen found that despite the advantages of the frontal method, there is a break-even point in front width above which iterative solutions become more efficient.[3] Their preconditioned iterative method outperformed the frontal method in their tests when the frontal width exceeded approximately 500. These results as well as those of other researchers[16] have encouraged us to focus on the performance of 'in-core' conjugate-gradient-like iterative algorithms.

True conjugate gradient methods compute approximations to $\mathbf{x}$ in the affine space $\mathbf{x}_0 + \mathbf{K}_m$, where $\mathbf{K}_m$ is the Krylov subspace of dimension m.[17] They are characterized by an optimality condition and economical or short vector recurrences.[18] Note that for symmetric matrices short vector recurrence relationships arise naturally, resulting in constant work and storage requirements on each iteration. For non-symmetric matrices, however, short recursions do not exist[19] and so the work and storage requirements increase with the iteration number; making the use of true conjugate gradient methods impractical for large problems.

In the case of non-symmetric matrices some problems allow successful application of true conjugate gradient algorithms to the normal equations (i.e. $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$).[12] Disadvantages in this approach, however, are that the condition number of the new system is made much worse and matrix–vector multiplications with $\mathbf{A}^T$ are required. Working with $\mathbf{A}^T$ is undesirable for several reasons:[20] first, the transpose is not always readily available; second, the efficiency of matrix–vector multiplications with the transpose may be reduced on vector/parallel computers; and third, working with the transpose eliminates the option of matrix-free implementations of Newton's method.[21–23] For these reasons we chose to concentrate on the performance of conjugate-gradient-like algorithms.

Conjugate-gradient-like methods are derived by either relaxing the optimality condition or sacrificing economical vector recursions.[24] The optimality condition may be relaxed by allowing periodic algorithm restarts and artificially truncating the recursion (i.e. the new direction vector is orthogonal to only the previous s direction vectors). Economical vector recursions can also be obtained at the expense of optimality by using the non-symmetric Lanczos biorthogonalization procedure (i.e. using three-term recursions to build a pair of biorthogonal bases).[24]

This investigation considers algorithms derived using the non-symmetric Lanczos procedure. Compared with Arnoldi-based methods (i.e. GMRES[25]), these Lanczos-based methods typically require less work and storage per iteration. The first of the Lanczos-based methods developed was the biconjugate gradient (BCG) algorithm, which requires matrix–vector multiplications with the matrix transpose.[26,27] This shortcoming was overcome with the development of CGS, a variant of BCG that avoids the use of the matrix transpose. CGS doubles the rate of convergence of BCG, but unfortunately it also doubles the rate of divergence. Since CGS lacks a minimization property, it sometimes exhibit very erratic convergence behaviour. In order to obtain more smoothly convergent CGS-like solutions, Freund applied the quasi-minimal residual idea to the CGS algorithm (TFQMR).[29] Note that both CGS and TFQMR may encounter algorithm breakdown, although in our experience these occurrences appear to be infrequent in practice. The look-ahead Lanczos procedure has been used in other algorithms to avoid these breakdowns, but they once again require working with the matrix transpose.[12,30,31]

In this study two inexact Newton algorithms are used to solve the incompressible Navier–Stokes and energy equations. The first is formed using the TFQMR algorithm (Newton-

TFQMR) and the second uses the CGS algorithm (Newton-CGS). The important features of these algorithms are discussed further in Section 2, while performance results obtained in solving the well-known problem of natural convection in an enclosed cavity[32] are presented in Section 3. The latter section demonstrates the significant memory advantages possible with the use of iterative solvers in contrast with the use of LINPACK banded Gaussian elimination (direct-Newton). Two parameter studies are also included in Section 3. The first investigates varying the inner iteration convergence criteria and the second investigates increasing the level of fill-in used in the ILU preconditioner. Based on the results of these parameter studies, the two inexact Newton algorithms are benchmarked against the direct-Newton algorithm. Additionally, the relative merits of both the Newton-CGS and Newton-TFQMR algorithms are studied and compared. Conclusions and important observations are summarized in Section 4.

2. NUMERICAL SOLUTION ALGORITHM

This section describes the important features of the inexact Newton algorithms. These features include the use of a numerical Jacobian evaluation to simplify implementation and the use of mesh sequencing to extend the radius of convergence of the algorithm. In addition, important points regarding the use of conjugate-gradient-like algorithms within an inexact Newton iteration are discussed.

2.1. Newton's method

Newton's method is a robust technique for solving systems of non-linear equations of the form

$$\mathbf{F}(\mathbf{x}) = [f_1(\mathbf{x}) f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})]^{\mathrm{T}} = 0, \tag{1}$$

where the state variable $\mathbf{x}$ can be expressed as

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}. \tag{2}$$

Application of Newton's method requires the solution of the linear system

$$\mathbf{J}^n \delta \mathbf{x}^n = -\mathbf{F}(\mathbf{x}^n), \tag{3}$$

where the elements of the Jacobian $\mathbf{J}$ are defined by

$$J_{ij} = \partial f_i / \partial x_j \tag{4}$$

and the new solution approximation is obtained from

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \mathrm{d}\delta \mathbf{x}^n. \tag{5}$$

The constant d $(0 < d < 1)$ in equation (5) is sometimes used to damp the Newton updates. The damping strategy is designed to prevent the calculation of non-physical variable values (i.e. negative temperature) and to scale large variable updates when the solution is far from the true solution. Damping was not necessary to obtain the solutions presented Section 3. This iteration is continued until the norm of $\delta\mathbf{x}$ and/or the norm of $\mathbf{F}(\mathbf{x})$ are below some suitable tolerance level. This convergence criterion is discussed further in Section 2.5.

We note that for problems where forming and factoring the Jacobian matrix account for a significant fraction of the CPU time, successful use of a modified Newton iteration can lead to substantial CPU time savings.[1] Implementation of a modified Newton iteration with a direct linear equation solver is straightforward and efficient, but CPU savings can also be realized with iterative solvers by freezing the Jacobian and preconditioning matrices for several Newton steps within an inexact Newton iteration. In this study, however, the focus is on the performance of the inexact Newton iteration. An investigation of the benefits of using a modified Newton iteration with an iterative linear equation solver will be deferred to a later study. Thus the performance of our inexact Newton algorithm will be benchmarked against a full Newton iteration using LINPACK banded Gaussian elimination.

2.2. Numerical Jacobian

The elements of the Jacobian in equation (4) are evaluated numerically using finite difference approximations,

$$J_{ij} = \frac{f_i(x_1, x_2, \ldots, x_j + \Delta x_j, \ldots, x_n) - f_i(x_1, x_2, \ldots, x_n)}{\Delta x_j}, \tag{6}$$

where

$$\Delta x_j = a x_j + b \tag{7}$$

and a and b are small perturbation constants.[1,8,33]

We use an algorithm that maintains much of the flexibility of a standard numerical Jacobian but requires only a small fraction of the total CPU time per iteration.[1,8]

2.3. Mesh sequencing

Mesh sequencing is used to obtain an initial guess on the final grid that lies within the radius of convergence of Newton's method. Mesh sequencing is analogous to the first upward cycle of a full multigrid (FMG) algorithm.[34] We use third-order Lagrangian interpolation to move through a series of uniform grids that are generated from the previous grid by doubling the grid dimension in both directions. Thus the interpolated solution from the previous grid is used as the initial guess on the new grid. The improved efficiency resulting from the use of mesh sequencing will be demonstrated in Section 3.

2.4. Iterative linear equation solver

The desire to extend our original research to fine, two-dimensional grids and to three-dimensional grids motivated the switch from a banded Gaussian elimination solver to a preconditioned conjugate-gradient-like algorithm for the solution of equation (3). The use of banded Gaussian elimination in solving these types of problems can be extremely costly, in terms of both CPU time and memory requirements, owing to large matrix bandwidths. To avoid these problems, the CGS and TFQMR algorithms have been implemented to take advantage of the sparse, banded structure of the Jacobian matrix.

In order to accelerate the convergence of the TFQMR algorithm, we currently use right preconditioning. The right-preconditioned TFQMR algorithm[29] is presented below for completeness, where $\mathbf{P}_r$ represents the inverse of the preconditioning matrix and the algorithm is designed to solve the system $\mathbf{Ax} = \mathbf{b}$. A listing of the CGS algorithm is omitted here, but note that TFQMR is an extension of the CGS algorithm. For additional information regarding the development of these algorithms see References 28 and 29. In the listing below, keep in mind that within the inexact Newton algorithm, $\mathbf{A} = \mathbf{J}$, $\mathbf{x} = \delta\mathbf{x}$ and $\mathbf{b} = -\mathbf{F}(\mathbf{x}^n)$. All other variables are defined within the TFQMR listing.

```
I. Initialize
  1. Choose $\mathbf{x}_0$.
  2. Set
     $\mathbf{r}_0^{\mathrm{CGS}} = \tilde{\mathbf{r}}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0, \quad \mathbf{q}_0 = \mathbf{p}_{-1} = \mathbf{d}_0 = \mathbf{0}, \quad v_0 = \eta_0 = n = 0;$
     $\rho_{-1} = 1, \quad \tau_0 = \|\mathbf{r}_0^{\mathrm{CGS}}\| \quad (\|\cdot\| \text{ denotes Euclidean norm}).$
II. For $n = 0, 1, 2, \ldots,$ do:
  1. Set
     $\rho_n = \tilde{\mathbf{r}}_0^{\mathrm{T}}\mathbf{r}_n^{\mathrm{CGS}}, \quad \beta_n = \rho_n/\rho_{n-1}, \quad \mathbf{u}_n = \mathbf{r}_n^{\mathrm{CGS}} + \beta_n\mathbf{q}_n;$
     $\mathbf{p}_n = \mathbf{u}_n + \beta_n(\mathbf{q}_n + \beta_n\mathbf{p}_{n-1}), \quad \mathbf{v}_n = \mathbf{A}(\mathbf{P}_r)\mathbf{p}_n, \quad \sigma_n = \tilde{\mathbf{r}}_0^{\mathrm{T}}\mathbf{v}_n, \quad \alpha_n = \rho_n/\sigma_n;$
     $\mathbf{q}_{n+1} = \mathbf{u}_n - \alpha_n\mathbf{v}_n, \quad \mathbf{v}_n = \alpha_n(\mathbf{P}_r)(\mathbf{u}_n + \mathbf{q}_{n+1}), \quad \mathbf{r}_{n+1}^{\mathrm{CGS}} = \mathbf{r}_n^{\mathrm{CGS}} - \mathbf{A}\mathbf{v}_n.$
  2. For $m = 2n+1, 2n+2$ do:
     $\omega_{m+1} = \begin{cases} \|\mathbf{r}_{n+1}^{\mathrm{CGS}}\| & \text{if } (m+1) \text{ is odd} \\ \sqrt{(\|\mathbf{r}_n^{\mathrm{CGS}}\| \cdot \|\mathbf{r}_{n+1}^{\mathrm{CGS}}\|)} & \text{if } (m+1) \text{ is even} \end{cases}, \quad v_m = \frac{\omega_{m+1}}{\tau_{m-1}};$
     $c_m = \frac{1}{\sqrt{(1 + v_m^2)}}, \quad \tau_m = \tau_{m-1}v_m c_m, \quad \eta_m = c_m^2\alpha_n;$
     $\mathbf{y}_m = \begin{cases} \mathbf{u}_n & \text{if } m \text{ is odd} \\ \mathbf{q}_n & \text{if } m \text{ is even} \end{cases}, \quad \mathbf{d}_m = \mathbf{y}_m + \frac{v_{m-1}^2\eta_{m-1}}{\alpha_n}\mathbf{d}_{m-1};$
     $\mathbf{x}_m = \mathbf{x}_{m-1} + \eta_m(\mathbf{P}_r)\mathbf{d}_m;$
  continue until $\mathbf{x}_m$ has converged.
```

In this study $\mathbf{P}_r$ is the inverse of an incomplete lower–upper (ILU(k)) factorization of the Jacobian matrix.[35–38] A modified 'level of fill-in' idea is used to determine non-zero locations in the LU factors. This is accomplished by initializing the level of all original non-zero elements in the Jacobian matrix to zero. Then fill-in terms arising from the elimination of a k-level term are set to level $k + 1$.[37] In our implementation, however, we take advantage of the banded structure of our Jacobian matrix and store only non-zero diagonals. This means that if a fill-in term is not located in one of the stored diagonals, an additional matrix diagonal must be added to the diagonal set in order to include this fill-in term. We recognize that more compact storage schemes would eliminate this additional fill-in.[39–41] However, for the simple test problem in this study the memory advantages of these schemes did not warrant sacrificing the convenience of the diagonal storage scheme.

Since $\mathbf{P}_r = (\mathbf{LU})^{-1}$, where LU represents the incomplete factorization of $\mathbf{J}$, one can see that products of the form $(\mathbf{P}_r)\mathbf{v}$ are calculated using simple forward/backward solves. Additionally, since $\mathbf{P}_r$ is an approximate inverse of $\mathbf{J}$, it can be used to obtain an initial guess for the solution of the linear system (i.e. by setting $\delta\mathbf{x}^0 = -\mathbf{P}_r\mathbf{F}(\mathbf{x}^0)$).

One of the difficulties in solving the primitive variable form of the incompressible Navier–Stokes equations is that pressure does not explicitly appear in the continuity equation. Thus, if the continuity equation is solved for pressure, a zero will appear on the main diagonal in all the rows in the Jacobian matrix representing the continuity equation. This difficulty is overcome by a direct solver with efficient pivoting, but pivoting may not be practical when using a sparse matrix iterative solver. Alternatives to pivoting include adding non-zeros to the diagonal using some sort of penalty function[2,16] and realigning the equations and variables to avoid zeros on the main diagonal.[6] In the case of an iterative solver using ILU preconditioning, fill-in resulting from the incomplete factorization will generate non-zero terms in most of these zero diagonal rows. However, Chin *et al.*[9] pointed out that for a natural ordering (i.e. '*uvpT*') there will be no fill-in on the continuity equation row if the finite volume lies adjacent to a corner boundary such that the bottom and left face coincide with the boundary. They further note that if there is only one such cell, the difficulty can be removed by arbitrarily fixing the pressure in that cell. However, problems arise when more than one such cell exists in the computational grid.

Chin *et al.* chose to investigate clever alternative ordering strategies to solve this problem.[9] An alternative technique, which avoids the additional complexity caused by variable reordering, is the use of Kershaw's method for treating unstable pivots in incomplete LU factorizations.[42] This method allows near-zero pivots to be adjusted in such a way that the incomplete factorization algorithm is kept stable and the error associated with the pivot adjustment is minimized.[42] One advantage in this approach is that very small pivots, which may cause algorithm instability, are adjusted along with the hard zero pivots. We have observed the benefit of this feature elsewhere in solving the equations governing the incompressible flow over a backward-facing step.[8]

In the case of the natural convection problem, however, only one such 'problem' cell exists (lower left corner). Thus the difficulty is overcome by simply fixing the pressure to a constant value in that cell, which is justified for this model problem and the incompressible flow assumption.[43]

2.5. *Inexact Newton method*

One advantage in coupling an iterative linear equation solver with Newton's method is that the linear system can be solved less accurately during the initial Newton iterations when far from the true solution and more accurately as the true solution is approached. This is in contrast with the use of a direct solver, which requires the same amount of work whether one is close to the true solution or not.

We adopt an inner iteration convergence criterion similar to that proposed by Averick and Ortega[14] and Dembo *et al.*[15] Specifically, the inner TFQMR iteration is assumed converged when

$$R_n^{\mathrm{i}} = \frac{\| \mathbf{J}^n \delta \mathbf{x}^n + \mathbf{F}(\mathbf{x}^n) \|_2}{\| \mathbf{F}(\mathbf{x}^n) \|_2} < \gamma_n, \tag{8}$$

where the superscript on R_n^{i} refers to the inner iteration and the subscript indicates the dependence on the Newton iteration. The selection of the best value of γ_n is highly empirical. In Section 3.3.2 we investigate two options for setting γ_n. The first is to set γ_n to a constant value and the second is to let γ_n vary on each Newton iteration.[23]

This inner iteration convergence criterion is used with a limit on the maximum number of inner iterations. In our numerical experiments we have set this upper limit equal to 200. We will demonstrate that in situations where this upper limit is encountered frequently, the selected iterative algorithm should not display erratic convergence behaviour such as that exhibited by BCG[26,27] and CGS.[28]

The convergence criterion for the outer Newton iteration is based upon a relative update defined by

$$R_n^o = \underset{\text{all } i}{\mathrm{Max}}\left[\frac{|\delta x_i^n|}{\mathrm{Max}\{|x_i^n|, 1\}}\right], \tag{9}$$

where the superscript on R_n^o refers to the outer Newton iteration and the subscript indicates the dependence on the Newton iteration. Convergence is then assumed when

$$R_n^o < 1 \times 10^{-6}. \tag{10}$$

This means that six digits of accuracy are required when the magnitude of the state variable is greater than one, and six decimal places of accuracy are required when the magnitude of the state variable is less than one.

3. RESULTS

3.1. Natural convection problem description

The geometry for the natural convection model problem is displayed in Figure 1. Using the Boussinesq approximation,[44] the governing equations for the natural convection problem in dimensionless and conservative form can be expressed as

continuity

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{11}$$

momentum

$$\frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} = -\frac{\partial p}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \tag{12}$$

$$\frac{\partial uv}{\partial x} + \frac{\partial v^2}{\partial y} = -\frac{\partial p}{\partial y} + \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 y}{\partial y^2} + GrT, \tag{13}$$

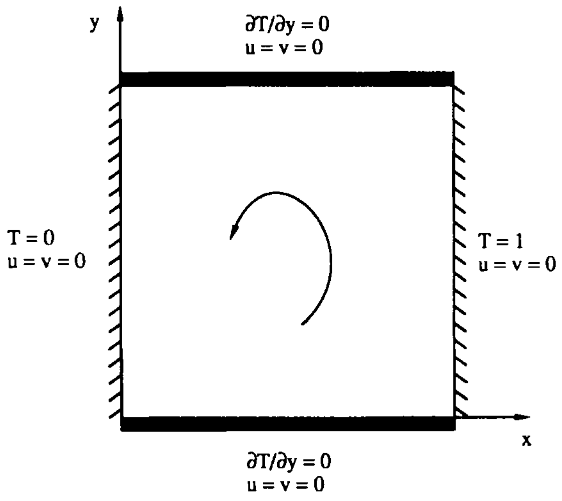

Figure 1. Geometry for natural convection model problem

Table I. Comparison of algorithm memory requirements using direct versus iterative linear equation solvers (in megawords)

Grid	Direct solver	Iterative solver			
		ILU(0)	IU(1)	ILU(2)	ILU(3)
15 × 15	0·174	0·0342	0·0414	0·0558	0·077
30 × 30	1·343	0·1368	0·166	0·2232	0·31
60 × 60	10·56	0·547	0·662	0·8928	1·24
120 × 120	83·69	2·189	2·65	3·57	4·95

energy

$$\frac{\partial uT}{\partial x} + \frac{\partial vT}{\partial y} = \frac{1}{Pr}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right), \tag{14}$$

where Gr is the Grashof number, Pr is the Prandtl number and the Rayleigh number Ra is given by $Ra = Gr\,Pr$. The gravity vector is assumed pointing in the negative y-direction. Boundary conditions for this problem are specified in Figure 1.

3.2. *Memory requirements*

Our inexact Newton method uses conjugate-gradient-like algorithms to solve equation (3). Since these algorithms are well suited for sparse matrix applications, significant reductions in computer memory requirements are possible. Table I demonstrates this memory advantage for the natural convection problem, comparing the preconditioned TFQMR algorithm with LINPACK banded Gaussian elimination. The direct solve data represent the memory required to store the Jacobian matrix, while the iterative solve data represent the memory required to store the Jacobian matrix and the ILU preconditioner. Four different levels of fill-in are considered for the ILU preconditioner. Table I shows that the potential memory advantage increases with grid refinement. For the 60 × 60 grid the memory required for the direct solve is roughly an order of magnitude larger than that required for the iterative solve. For the coarsest grid listed, the direct solve memory requirement is a factor of two larger than the iterative solve with ILU(3) preconditioning and a factor of five larger with ILU(0) preconditioning. Note that all computations were run on a CRAY X-MP/216 with 16 megawords of memory. Thus the 60 × 60 grid was the finest allowable grid for calculations with the direct solver.

3.3. *Algorithm performance*

In this section the performance of the inexact Newton method algorithm is benchmarked against a direct Newton iteration using LINPACK banded Gaussian elimination. The performance of the inexact Newton method using ILU-preconditioned TFQMR is studied with respect to the inexact Newton convergence parameter (γ_n), the level of fill-in used in the ILU(k) preconditioner and the use of mesh sequencing. In addition, the performance of the algorithm using TFQMR is compared with the performance obtained using the CGS algorithm. Note that the TFQMR algorithm provides an upper bound for the residual norm that was not used in this study. Use of this upper bound could make the TFQMR algorithm less expensive, because the calculation of the residual norm could be postponed until this upper bound was small enough.

Table II. Performance data using LINPACK banded Gaussian elimination for $Ra = 10^4$

Grid	Newton iterations (n)	CPU time (s)
15×15	7	3·52
30×30	7	30·72
60×60	7	262·82
$15 \times 15 > 30 \times 30 > 60 \times 60$	7, 4, 4	172·9

3.3.1. Direct Newton iteration results. Performance data using LINPACK banded Gaussian elimination are presented in Table II for $Ra = 10^4$. The required number of Newton iterations (n) as well as the required CPU time are listed for three different mesh sizes. The last row represents data using mesh sequencing. In this case the required number of iterations on each grid is listed in the second column. Table II shows that the required CPU time increases significantly as the grid dimensions are doubled in both directions. Note also that the use of mesh sequencing reduced the required CPU time by roughly 34%. Mesh sequencing enables this saving by providing a better initial guess on the finest grid. This results in fewer iterations when the CPU cost per iteration is high. For this problem the CPU cost of a single iteration on the finest grid was equivalent to approximately 75 iterations on the coarest grid. This behaviour is consistent with previous results[1] where CPU savings of approximately 45% were observed. Differences between these results and those of Reference 1 are due to the use of different convergence criteria.

3.3.2. Convergence parameter γ_n. The efficiency of an inexact Newton iteration is closely tied to the proper selection of γ_n. If γ_n is chosen too small, needless extra work will be performed when the Newton iteration is not within the radius of convergence of the algorithm. Conversely, if γ_n is chosen too large, the convergence of the Newton iteration will be slow. Here we investigate two options for setting γ_n. The first is to set γ_n to a constant value and the second is to let γ_n vary on each Newton iteration using an expression similar to that proposed in Reference 22. Table III demonstrates the effect of varying γ_n on algorithm performance. The results presented in Table III were obtained for $Ra = 10^4$ on a 60×60 grid starting from a flat initial guess ($u = v = 0$, $T = 0{\cdot}5$) using ILU(2)-preconditioned TFQMR to solve equation (3). The expression

Table III. Effect of varying γ_n on algorithm performance (60×60 grid, flat initial guess)

γ_n	n	$\bar{m}$	CPU time (s)
10^{-4}	7	62	568·8
10^{-3}	8	51	536·6
10^{-2}	9	38	474·4
10^{-1}	11	25	408·76
$\frac{1}{2}$	24	12	545·9
$(\frac{1}{2})^{\mathrm{Min}(n,10)}$	9	32	403·8
$(\frac{1}{4})^{\mathrm{Min}(n,10)}$	8	40	441·7

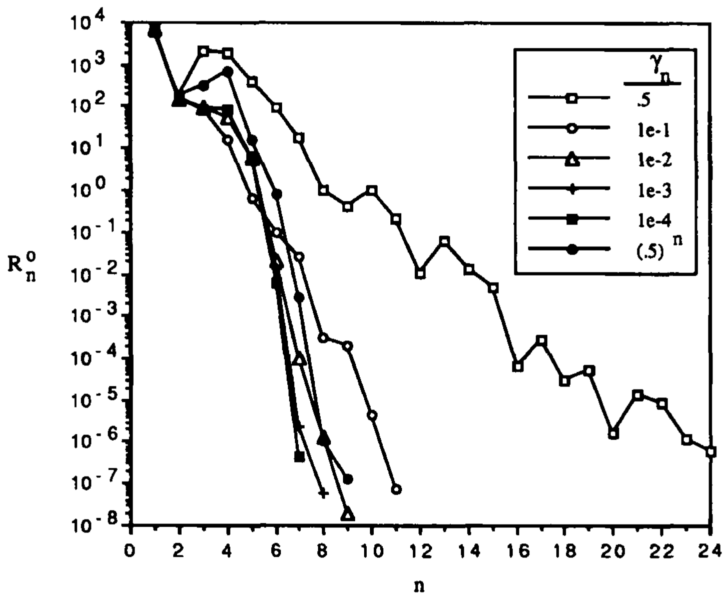

Figure 2. Effect of γ_n on algorithm convergence

used for γ_n, the required number of Newton iterations (n), the average TFQMR iterations per Newton iteration ($\bar{m}$) and the total CPU time are given. The results suggest that $\gamma_n < 10^{-3}$ is too restrictive during the initial Newton iterations. Conversely, $\gamma_n > 10^{-1}$ was not sufficiently restrictive when the Newton iteration was close to the true solution, resulting in a large number of required Newton iterations. The best overall results were obtained using $\gamma_n = (\frac{1}{2})^{\mathrm{Min}\{n,10\}}$. For this selection γ_n initially assumes the value of $\frac{1}{2}$ and is reduced with the Newton iteration number until it reaches a minimum value of the order of 10^{-3}. Thus γ_n becomes more restrictive as the true solution is approached. The effect of varying γ_n on the algorithm convergence behaviour is shown in Figure 2. Observe that $\gamma_n = 0{\cdot}5$ results in slow convergence, while reduced γ_n-values yield much faster (superlinear) convergence. Another important observation is that $\gamma_n = (\frac{1}{2})^{\mathrm{Min}\{n,10\}}$, although large initially, still produces very favourable convergence behaviour. Similar performance was observed for coarser grids and other levels of ILU fill-in.

Analogous results obtained using mesh sequencing are shown in Table IV, where a 60 × 60

Table IV. Effect of varying γ_n on algorithm performance

γ_n	n	$\bar{m}$	CPU time (s)
10^{-4}	7, 4, 4	10, 18, 50	321·7
10^{-3}	7, 4, 4	8, 16, 45	275·4
10^{-2}	8, 5, 5	6, 12, 34	272·8
10^{-1}	9, 7, 8	3, 7, 20	293·9
$(\frac{1}{2})^{\mathrm{Min}\{n,10\}}$	7, 7, 8	5, 7, 31	390·5
$(\frac{1}{4})^{\mathrm{Min}\{n,10\}}$	7, 6, 6	6, 12, 33	321·7

Table V. Effect of higher levels of ILU fill-in on algorithm performance (flat initial guess on each grid)

	15 × 15 grid			30 × 30 grid			60 × 60 grid		
k	n	m	t	n	m	t	n	m	t
0	8	11	3·96	9	37	47·51	9	121	563·6*
1	8	5	4·95	9	16	46·98	8	60	521·6
2	7	5	6·23	8	11	40·54	9	32	403·8
3	9	6	14·63	8	13	70·76	9	26	487·3

* Indicates the TFQMR iteration limit of 200 was encountered.

grid solution for $Ra = 10^4$ was obtained using a 15×15 and 30×30 grid sequence. In this case values for n and $\bar{m}$ are presented for each grid. A comparison of Tables III and IV again demonstrates the benefits of mesh sequencing. For example, using $\gamma_n = 10^{-3}$, mesh sequencing enabled a CPU time saving of approximately 42%. Table IV also suggests that when using mesh sequencing, a more restrictive convergence criterion is needed during the initial Newton iterations in order to take advantage of the improved initial guesses on the finer grids. The adaptive convergence selections did not work as well with mesh sequencing. In fact, a constant value of $\gamma_n = 10^{-2}$ yielded the best overall results. The potential saving using $\gamma_n = (\frac{1}{2})^{\mathrm{Min}(n,10)}$ on the initial grid is not warranted, because the CPU cost of the initial grid solution is typically a small fraction of the cost of the total calculation.

Based on the results presented in Tables III and IV, it appears that the selection of $\gamma_n = 10^{-2}$ is advisable when a good initial guess is available. However, when a good initial guess is not available, the adaptive convergence criterion of $\gamma_n = (\frac{1}{2})^{\mathrm{Min}(n,10)}$ appears the best overall selection.

3.3.3. Effect of fill-in using ILU(k) preconditioning. Effective preconditioning is essential in improving the robustness of the TFQMR iteration. One measure of an effective preconditioner is how well it approximates the system matrix. For an incomplete LU factorization, allowing more fill-in will most likely improve this approximation. The drawback, however, is higher CPU and memory storage cost. This suggests an optimal level of fill-in that balances CPU time and memory considerations against preconditioner effectiveness. Table V demonstrates the effect of different levels of fill-in (k) versus grid size for $Ra = 10^4$. Listed for each grid are the required number of Newton iterations (n), the average TFQMR iterations per Newton iteration ($\bar{m}$) and the total CPU time (t). The solution on each grid was obtained from a flat initial guess ($u = v = 0$, $T = 0{\cdot}5$). Although the use of $k > 0$ on the 15×15 grid reduced the average TFQMR iterations, the total CPU time actually increased. For the 30×30 grid a small reduction in CPU time was observed using ILU(1) and ILU(2). On the 60×60 grid the benefits of $k > 0$ became more significant. ILU(2) preconditioning reduced the average TFQMR iterations by a factor of four and the total CPU time by approximately 30% compared with ILU(0) preconditioning. Note that the CPU performance of ILU(3) was poor on the coarser grids and was not as efficient as ILU(2) on the 60×60 grid. For this problem, use of ILU(2) preconditioning provides a good compromise between CPU time and memory considerations and preconditioner effectiveness.

3.3.4. TFQMR performance benchmark. This subsection is intended to benchmark the performance of the inexact Newton algorithm using the TFQMR algorithm[29] against the use of the CGS algorithm.[28] In addition, the performance of the inexact Newton iteration is

Table VI. Comparison of CPU performance of two inexact Newton algorithms and the direct-Newton algorithm (flat initial guess on each grid)

	CPU time (s)				
Grid	Direct-Newton	Newton-TFQMR	Newton-CGS	R_{TFQMR}	R_{CGS}
15 × 15	3·52	3·96	2·25	1·125	0·64
30 × 30	30·72	40·54	28·3	1·32	0·92
60 × 60	262·82	403·76	221·54	1·54	0·84

compared with a direct Newton iteration using LINPACK banded Gaussian elimination. Table VI compares the CPU performance of these different algorithms versus grid size for $Ra = 10^4$. The inexact Newton algorithms use ILU(0) on the coarsest grid and ILU(2) on the two finer grids. $\gamma_n = (\frac{1}{2})^{\text{Min}(n, 10)}$ is used as the inner iteration convergence criterion. The last two columns of Table VI present ratios of the required CPU time using the iterative solvers to the required CPU time using the direct solver. Thus R_{TFQMR} represents the ratio of total CPU time using the Newton-TFQMR algorithm to the total CPU time using the direct-Newton algorithm. Similarly, R_{CGS} represents the ratio of total CPU time using the Newton-CGS algorithm, to the total CPU time, using the direct-Newton algorithm. The results indicate that the Newton-CGS algorithm was more efficient than both the direct-Newton and Newton-TFQMR algorithms. The Newton-TFQMR algorithm was less efficient than the direct Newton iteration, but still competitive. For both inexact Newton algorithms, forward–backward solve operations associated with the ILU preconditioning dominated the CPU time. Our implementation of the CGS algorithm requires three of these operations per iteration, while the TFQMR algorithm requires five such operations. In addition, the TFQMR algorithm performs three more vector additions per iteration than the CGS algorithm. These differences roughly account for the increased CPU times observed for the TFQMR algorithm, since the iteration counts for the two algorithms were similar. As noted previously, the TFQMR algorithm could be made more efficient by making use of the available upper bound for the residual norm.

The reduced CPU efficiency of TFQMR compared with CGS is compensated with improved robustness. Recall from the discussion in Section 1 that CGS displays rather erratic convergence behaviour. In Section 2.4 we alluded to the potential problems that may arise when this erratic convergence behaviour is encountered within an inexact Newton iteration. In fact, if ILU(0) preconditioning is used to solve a higher-Ra problem ($Ra = 10^5$) on a 60 × 60 grid with $\gamma_n = 10^{-3}$, the Newton–CGS algorithm fails after only the second Newton iteration, while the Newton-TFQMR algorithm converges after 13 Newton iterations. The cause of the Newton-CGS failure is the erratic convergence behaviour of the CGS algorithm, which has been observed elsewhere.[12,28] Recall from Section 2.4 that we impose an upper limit on the number of inner iterations equal to 200. On the second Newton iteration both the CGS and TFQMR algorithms encountered this upper limit. While the erratic convergence behaviour of CGS returned a very poor approximate solution to equation (3), the TFQMR algorithm returned an acceptable solution that allowed the eventual convergence of the algorithm. This behaviour illustrated in Figures 3 and 4, which show the convergence behaviour of both CGS and TFQMR during the first and second Newton iterations. Figure 3 shows that both algorithms converged to the desired tolerance on the first Newton iteration, although the convergence of the CGS algorithm is very erratic. Note how the TFQMR algorithm successfully controls and smooths the erratic CGS

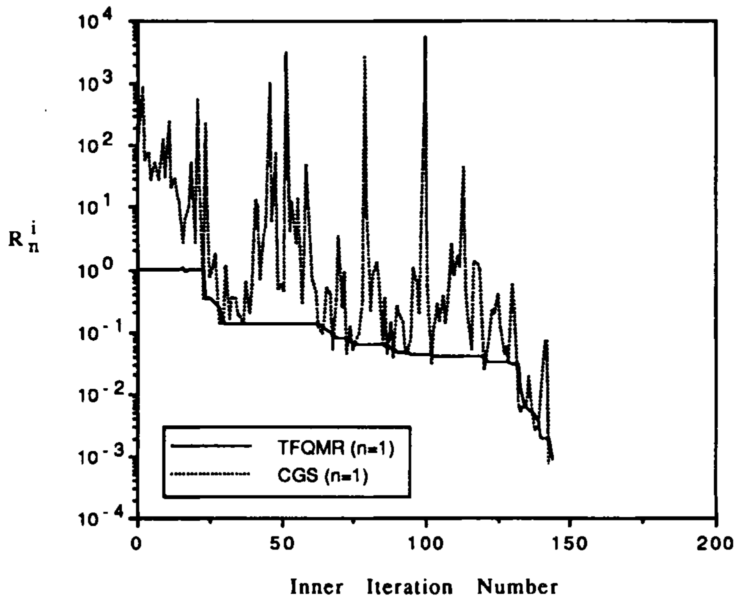

Figure 3. Convergence comparison between TFQMR and CGS on first Newton iteration

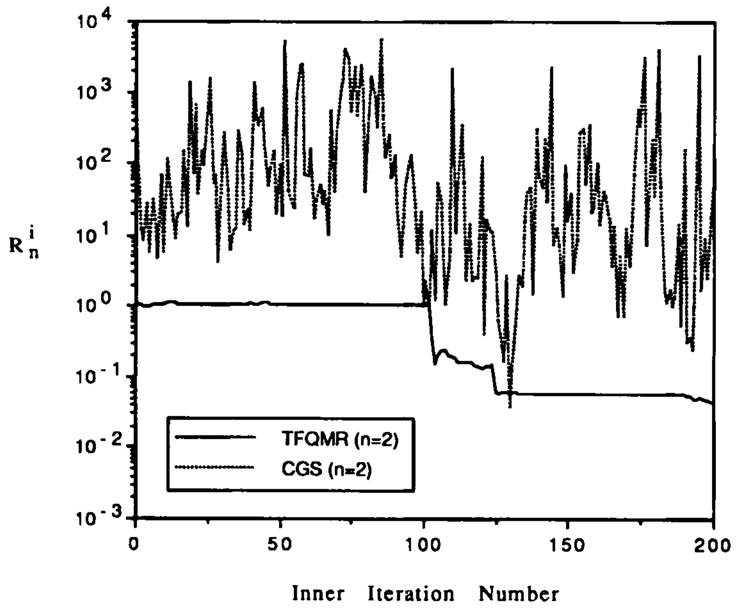

Figure 4. Convergence comparison between TFQMR and CGS on second Newton iteration

Table VII. Comparison with benchmark solution of De Vahl Davis[32]

	Benchmark solution	Current solution			
		15×15	30×30	60×60	120×120
$u_{\max}$	22·7859	22·663	22·738	22·784	22·788
y	0·177	0·167	0·183	0·175	0·179
$v_{\max}$	27·6296	27·599	27·657	27·594	27·640
x	0·881	0·900	0·883	0·875	0·879

convergence behaviour. Figure 4 shows that during the second Newton iteration neither algorithm converged to the desired tolerance after 200 iterations. The CGS algorithm terminated with R_n^i approximately equal to 45. This means that the residual norm of the linear system (equation (3)) was 45 times larger than if the Newton update was assumed zero. This poor update resulted in the failure of the algorithm on the subsequent Newton iteration. The TFQMR algorithm, on the other hand, terminated with R_n^i approximately equal to 0·043, which represented an acceptable approximate solution to equation (3).

This behaviour might be avoided in some instances by using better preconditioning to improve the convergence behaviour of the iterative algorithms. However, this option may not always be feasible owing to memory limitations or to the lack of a better-known preconditioner. For this reason we feel that the advantage of robustness associated with the use of TFQMR is more important than the slight CPU efficiency advantage obtained with CGS.

3.4. *Solutions*

The solutions to the natural convection problem for $Ra = 10^4$ are compared with the benchmark solution of De Vahl Davis[32] in Table VII. Note that the velocity data from Reference 32 were multiplied by the factor P_R^{-1} to account for the different choices in scaling. Additionally, the positions were adjusted to account for the reversed circulation direction in Reference 32. Table VII presents the maximum horizontal velocity component (u) and its corresponding y-location along the line $x = 0{\cdot}5$, and the maximum vertical velocity component (v) and its x-location along the line $y = 0{\cdot}5$. Data from four different grids of increasing refinement are compared with the benchmark solution of Reference 32. Table VII shows the improved agreement with the benchmark solution as the grid is refined. The agreement between the benchmark solution and the 120×120 grid solution is very good.

4. CONCLUSIONS

Fully implicit inexact Newton algorithms were used to solve the well-known natural convection model problem governed by the steady, incompressible Navier–Stokes and energy equations. An efficiently evaluated numerical Jacobian was used to simplify implementation and mesh sequencing was used to improve robustness and CPU efficiency. The TFQMR and CGS iterative algorithms are used to form the inexact Newton algorithms. Right ILU(k) preconditioning was used to improve the performance of the iterative solvers. The inexact Newton algorithms were found to be CPU competitive with a direct Newton iteration using LINPACK banded Gaussian elimination, yet significantly more efficient from a memory standpoint. In certain circumstances the Newton–TFQMR algorithm was shown to be more robust than the Newton-CGS algorithm.

The inexact Newton algorithm allows less work in solving the linear systems during the initial Newton iterations when far from the true solution, but requires more accurate solutions as the true solution is approached. We investigated several choices for the convergence parameter γ_n which controls this behaviour. Among these choices, $\gamma_n = (\frac{1}{2})^{\mathrm{Min}\{n,10\}}$ was the best choice when a good initial guess was not available, but $\gamma_n = 10^{-2}$ worked best overall when a good initial guess was available.

Effective preconditioning is an essential ingredient in the successful use of conjugate-gradient-like algorithms. Thus in the case of ILU(k) preconditioning we tried to determine the optimal level of fill-in that balances CPU efficiency with preconditioner effectiveness. We found that ILU(2) preconditioning provided a good compromise between CPU efficiency, memory considerations and preconditioner effectiveness for moderately refined grids. For coarse grids (i.e. 15×15) we found that ILU(0) preconditioning was sufficient.

In the future we plan to investigate matrix-free implementations of the inexact Newton method using several combinations of preconditioners and conjugate-gradient-like algorithms.[20–22] With regard to the incompressible Navier–Stokes equations, we hope to investigate the effect of solving a pressure equation in place of the continuity equation in order to avoid zeros on the main diagonal of the Jacobian matrix. Additionally, the effects of using higher-order differencing and body-fitted co-ordinates on algorithm performance will also be considered.

ACKNOWLEDGEMENTS

This work is supported through the EG&G Idaho Long Term Research Initiative in Computational Mechanics under DOE Idaho Field Office Contract DE-AC07-76ID01570. The authors would like to acknowledge the careful reading and valuable suggestions made by the referees.

APPENDIX: NOMENCLATURE

a	perturbation constant
b	perturbation constant
d	damping constant
$\mathbf{F}$	governing equation vector
Gr	Grashof number
$\mathbf{J}$	Jacobian matrix
k	level of fill-in in ILU preconditioner
m	inner iteration counter
n	Newton iteration counter
Nu	Nusselt number
Pr	Prandtl number, ν/α
Ra	Rayleigh number
u	dimensionless principal velocity, $\tilde{u}/\bar{u}$
v	dimensionless transverse velocity, $\tilde{v}/\bar{u}$
$\mathbf{x}$	state variable vector
$\delta\mathbf{x}$	update vector
Δx_j	perturbation in jth component of state vector
x	principal co-ordinate variable

Δx	grid spacing in x-direction
y	transverse co-ordinate variable
Δy	grid spacing in y-direction

Greek letters

α	thermal diffusivity
γ	tolerance for iterative linear equation solver
η	time step control constant
ν	kinematic viscosity

Subscript

n	Newton iteration number

Superscripts

i	inner iteration
n	Newton iteration number
o	outer iteration

Operators

$[\ ]^{\mathrm{T}}$	transpose of $[\ ]$
$\lVert\cdot\rVert$	Euclidean norm
$\lVert\cdot\rVert_{\infty}$	L_{∞}-norm

REFERENCES

1. D. A. Knoll and P. R. McHugh, 'A fully implicit direct Newton solver for the Navier–Stokes equations', *Int. J. Numer. Methods Fluids*, **17**, 449–461 (1993).
2. O. C. Zienkiewicz, *The Finite Element Method*, 3rd edn, McGraw-Hill, London, 1977.
3. E. O. Einset and K. F. Jensen, 'A finite element solution of three-dimensional mixed convection gas flows in horizontal channels using preconditioned iterative matrix methods', *Int. J. Numer. Methods Fluids*, **14**, 817–841 (1992).
4. O. Dahl and S. O. Wille, 'An ILU preconditioner with coupled node fill-in for iterative solution of the mixed finite element formulation of the 2D and 3D Navier–Stokes equations', *Int. J. Numer. Methods Fluids*, **15**, 525–544 (1992).
5. J. W. MacArthur and S. V. Patankar, 'Robust semidirect finite difference methods for solving the Navier–Stokes and energy equations', *Int. J. Numer. Methods Fluids*, **9**, 325–340 (1989).
6. S. P. Vanka, 'Block-implicit calculation of steady turbulent recirculating flows', *Int. J. Heat Mass Transfer*, **28**, 2093–2103 (1985).
7. V. Venkatakrishnan, 'Viscous computations using a direct solver', *Comput. Fluids*, **18**, 191–204 (1990).
8. P. R. McHugh and D. A. Knoll, 'Fully implicit solution of the benchmark backward facing step problem using finite volume differencing and inexact Newton's method', *HTD*, Vol. 222, pp. 77–87, *Proc. ASME Winter Annual Meeting*, Anaheim, CA, November 1992, ASME, New York, 1992.
9. P. Chin, E. F. D'Azevedo, P. A. Forsyth and W.-P. Tang, 'Preconditioned conjugate gradient methods for the incompressible Navier–Stokes equations', *Int. J. Numer. Methods Fluids*, **15**, 273–295 (1992).
10. P. Hood, 'Frontal solution program for unsymmetric matrices', *Int. J. Numer. Methods Eng.*, **10**, 379–399 (1976).
11. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere, New York, 1980.
12. R. W. Freund, G. H. Golub and N. Nachtigal, 'Iterative solution of linear systems', *Numerical Analysis Project, Manuscript NA-91-05*, Computer Science Department, Stanford University, 1991.
13. C. H. Tong, 'A comparative study of preconditioned Lanczos methods for nonsymmetric linear systems', *Sandia National Laboratories Rep. SAND91-8240, UC-404*, 1992.
14. B. M. Averick and J. M. Ortega, 'Solutions of nonlinear Poisson-type equations', *Appl. Numer. Math.*, **8**, 443–455 (1991).
15. R. S. Dembo, S. C. Eisenstat and T. Steihaug, 'Inexact Newton methods', *SIAM J. Numer. Anal.*, **19**, 400–408 (1982).

16. M. P. Reddy *et al.*, 'Penalty finite element analysis of incompressible flows using element by element solution algorithms', *Comput. Methods Appl. Mech. Eng.*, **100**, 169–205 (1992).
17. Y. Saad and M. H. Schultz, 'Conjugate gradient-like algorithms for solving nonsymmetric linear systems', *Math. Comput.*, **44**, 417–424 (1985).
18. S. F. Ashby, T. A. Manteuffel and P. E. Saylor, 'A taxonomy for conjugate gradient methods', *SIAM J. Numer. Anal.*, **27**, 1542–1568 (1990).
19. V. Faber and T. Manteuffel, 'Necessary and sufficient conditions for the existence of a conjugate gradient method', *SIAM J. Numer. Anal.*, **21**, 352 (1984).
20. A. Ern, V. Giovangigli, D. E. Keyes and D. Smooke, 'Towards polyalgorithmic linear system solvers for nonlinear elliptic problems', *SIAM J. Sci. Comput.*, 681–703 (1994).
21. C. W. Gear and Y. Saad, 'Iterative solution of linear equations in ODE codes', *SIAM J. Sci. Stat. Comput.*, **4**, 583–601 (1983).
22. P. N. Brown and A. C. Hindmarsh, 'Matrix-free methods for stiff systems of ODEs', *SIAM J. Numer. Anal.*, **23**, 610–638 (1986).
23. P. N. Brown and Y. Saad, 'Hybrid Krylov methods for nonlinear systems of equations', *SIAM J. Sci. Stat. Comput.*, **11**, 450–481 (1990).
24. S. Ashby, T. Manteuffel and P. Saylor, *Preconditioned Polynomial Iterative Methods, a Tutorial*, University of Colorado at Denver, April 1992.
25. Y. Saad and M. H. Schultz, 'GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems', *SIAM J. Sci. Stat. Comput.*, **7**, 856 (1986).
26. C. Lanczos, 'Solution of systems of linear equations by minimized iterations', *J. Res. NBS*, **49**, 33–53 (1952).
27. R. Fletcher, 'Conjugate gradient methods for indefinite systems', in G. A. Watson (ed.), *Lecture Notes in Mathematics*, Vol. 506, *Proc. Dundee Conf. on Numerical Analysis*, Dundee, 1975, Springer, Berlin, 1976, pp. 73–89.
28. P. Sonneveld, 'CGS, a fast Lanczos-type solver for nonsymmetric linear systems', *SIAM J. Sci. Stat. Comput.*, 7, 856–869 (1986).
29. R. W. Freund, 'A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems', *SIAM J. Sci. Comput.*, **14**, 470–482 (1993).
30. R. W. Freund and N. M. Nachtigal, 'QMR: a quasi-minimal residual method for non-Hermitian linear systems', *IPS Res. Rep. 91-05*, Interdisciplinary Project Center for Supercomputing, ETH-Zentrum, Zurich, 1991.
31. R. W. Freund, M. H. Gutknecht and N. M. Nachtigal, 'An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices', *Tech. Rep. 91-09*, RIACS, NASA Ames Research Center, Moffett Field, CA, 1991.
32. G. De Vahl Davis, 'Natural convection of air in a square cavity: a benchmark numerical solution', *Int. J. Numer. Methods Fluids*, **3**, 249–264 (1983).
33. M. D. Smooke, 'Solution of burner-stabilized premixed laminar flames by boundary value methods', *J. Comput. Phys.*, **48**, 72–105 (1982).
34. A. Brandt, 'Multigrid techniques: 1984 guide with applications to fluid dynamics', *Tech. Rep.*, von Karman Institute, 1984.
35. J. A. Meijerink and H. A. van der Vorst, 'An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix', *Math. Comput.*, **31**, 148–162 (1977).
36. J. A. Meijerink and H. A. van der Vorst, 'Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems', *J. Comput. Phys.*, **44**, 134–155 (1981).
37. J. W. Watts, 'A conjugate gradient-truncated direct method for the iterative solution of the reservoir simulation pressure equation', *Soc. Petrol Eng. J.*, **21**, 345–353 (1981).
38. M. Sangback and A. T. Chronopoulos, 'Implementation of iterative methods for large sparse nonsymmetric linear systems on a parallel vector machine', *Int. J. Supercomput. Appl.*, **4**, 9–24 (1990).
39. Y. Saad, 'SPARSKIT, a basic tool kit for sparse matrix computations', *RIACS Tech. Rep. 90.20*, May, 1990.
40. E. Anderson and Y. Saad, 'Solving sparse triangular linear systems on parallel computers', *Int. J. High Speed Comput.*, **1**, 73–95 (1989).
41. M. A. Heroux, Phuong Vu and Chao Yang, 'A parallel preconditioned conjugate gradient package for solving sparse linjear systems on a CRAY Y-MP', *Appl. Numer. Math.*, **8**, 93–115 (1991).
42. D. S. Kershaw, 'On the problem of unstable pivots in the incomplete LU-conjugate gradient method', *J. Comput. Phys.*, **38**, 114–123 (1980).
43. P. M. Gresho, 'Some current CFD issues relevant to the incompressible Navier–Stokes equations', *Comput. Methods Appl. Mech. Eng.*, **87**, 201–252 (1991).
44. L. D. Landau and E. M. Lifshitz, *Fluid Mechanics*, 2nd edn, Vol. 6, 1987, Pergamon Press, Elmsford, New York, pp. 217–221.