INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 10, 925–945 (1990)

UNSYMMETRIC CONJUGATE GRADIENT METHODS AND SPARSE DIRECT METHODS IN FINITE ELEMENT FLOW SIMULATION

D. HOWARD, W. M. CONNOLLEY AND J. S. ROLLETT

Numerical Analysis Group, Oxford University Computing Laboratory, 8-11 Keble Road, Oxford OX1 3QD, U.K.

SUMMARY

A series of numerical experiments on the Cray XMP/48 and on the Cray 2 investigate the robustness and economy of direct and unsymmetric conjugate gradient (CG) type methods for the solution of matrix systems arising from a 3D FEM discretization of fluid flow problems. Computations on a Boussinesq flow model problem with either ILU preconditioned or unpreconditioned unsymmetric CG methods are presented. Such experiments seem to indicate that the unpreconditioned BICG method is robust for moderately non-linear incompressible Navier–Stokes FEM discretizations and that the ILU preconditioned BICG method is very robust and more economic than an unsymmetric frontal solver when the generous memory of the Cray 2 is exploited to store both the matrix and its preconditioner. We cover some of the programming aspects of direct and iterative methods on a supercomputer and find that direct methods have advantages: the crucial CPU-consuming area of code is compact but overwhelming, and its percentage of total CPU usage is independent of the spectral properties of the matrix involved. An optimal implementation of the unsymmetric CG method is more difficult because its work is related to the spectral distribution of the matrix considered and because there is no single portion of the code that overwhelmingly dominates the CPU usuage.

KEY WORDS Bi-conjugate gradient Conjugate gradient Conjugate gradient squared Finite elements Flow simulation Preconditioning Vector processor Frontal solver Navier–Stokes Boussinesq approximation Newton method Picard iteration Galerkin method Petrov–Galerkin

1. INTRODUCTION

The simulation of 3D engineering problems by implicit finite element methods[1] (FEM) represents a computational challenge because non-linear FEM models often require repeated solutions to a linearized system — a banded matrix problem. The choices for the method of solution to this matrix problem can be classified into direct and iterative. Global matrices can be very large and will typically require the use of a supercomputer or parallel architecture for solution. Fast methods such as conjugate gradient (CG) are suitable for symmetric systems. These are direct methods since they have a finite termination using exact arithmetic, but are classified as iterative because the round-off errors lose this theoretical property and because a satisfactory solution can often be obtained in far fewer than the theoretical maximum number of steps.

CG methods often require some form of preconditioning in order to improve upon the distribution of the eigenvalues or singular values of the matrix system to speed up the rate of convergence. The simplest preconditioner is diagonal scaling. More elaborate preconditioners such as incomplete factorization require both the extra work associated with a direct method and the additional storage requirement of the preconditioner. There exist preconditioners which use

0271–2091/90/080925–20$10.00

Received August 1988
Revised November 1989

parts of the global matrix incurring no significant extra work or storage, i.e. those based on SSOR.[2] Their success, however, is limited to a narrow class of problem.

FEM applications which discretize partial differential equations with the Galerkin method, resulting in symmetric positive definite system matrices, and which utilize quadratic or higher-order test and trial functions, can efficiently exploit a family of hierarchic basis functions.[3] These hierarchies are normally based on Legendre polynomials which are orthogonal in 1D and nearly orthogonal in higher dimensions. The resulting system matrices are not only better conditioned than if formed with the standard shape functions but also lend themselves to a special pre-conditioning technique, the block diagonal hierarchic preconditioned CG method.[4] One need only solve the matrix problem formulated by the low-order test and trial functions, forming a preconditioner for the higher-order polynomial approximation. This technique is now well established for symmetric FEM equations.[5]

Unfortunately, the advantages of hierarchic basis functions do not extend readily to FEMs which discretize partial differential equations containing convective terms. The Legendre polynomials are no longer orthogonal in this setting and can result in a system which is more poorly conditioned than by using the standard shape functions. Such applications are typical of the mixed finite element methods in incompressible fluid dynamics. These systems are in general also unsymmetric. Problems of this kind can be expressed in a general way by the solution of the following matrix system at each linearization stage:

$$Ax = b, \quad \text{where } A = \begin{bmatrix} K & C \\ L - C^{\mathrm{T}} & M \end{bmatrix} \quad \text{and} \quad x = \begin{bmatrix} u \\ P \end{bmatrix}. \tag{1}$$

Here K is a submatrix containing diffusive, advective and other linearized terms, C is the divergence submatrix, M is a 'stabilization' submatrix, L is a 'consistency' submatrix, u are the velocity unknowns and P the pressure unknowns. The last two submatrices have been introduced by Hughes *et al.*[6] in order to circumvent the Babuška–Brezzi condition[7,8] and arise from a consistent distortion of the Galerkin test functions at element level. In most cases the matrix K represents the difficulty because the advective terms it contains make it unsymmetric. Stokes flow is a special case in that the global matrix may become symmetric. This is exploited by the Lagrange–Galerkin method, which incorporates part of the advection into the right-hand-side vector b and adds a mass matrix to the Stokes matrix on the left-hand side. For simple constant-viscosity laminar flow one can then replace the continuous reforming and solution of the matrix problem by a 'resolution' of the right-hand-side vector at each linearization stage; in turbulent flow one can reform the matrix and right-hand side but now with repeated solutions to a symmetric matrix problem. Lagrange–Galerkin is appropriate to truly transient flows. For steady flows we believe it more appropriate to step through a linearization towards the steady state with repeated solutions to the unsymmetric matrix problem $Ax = b$. It is to the solution of this class of FEM systems that we apply unsymmetric CG methods, and unsymmetric Gauss elimination implemented on Cray supercomputers, after presenting and mathematically deriving some of the CG and unsymmetric CG methods.

2. THEORY OF CG

In this section we derive the basic CG algorithm starting from the Galerkin formulation. Then we derive the bi-conjugate gradient (BICG), conjugate-gradient-squared (CGS) and conjugate-residual (CGR) algorithms for unsymmetric matrices following Saad and Schultz[9] and Sonneveld *et al.*,[10] where proofs we have omitted may be found.

We consider A, a self-adjoint coercive linear operator from the inner product space V to itself, i.e.

$$\langle v, Aw\rangle = \langle Av, w\rangle \quad \forall v, w \in V,$$
$$\langle v, Av\rangle > 0 \qquad \forall v \in V - \{0\},$$

where $\langle \, . \, , \, . \, \rangle$ is the inner product on V. In practice V is finite-dimensional and we shall assume $V = \mathbb{R}^n$, so we can identify A with its matrix representation. Let us introduce the norm $\| \cdot \|_{A^{-1}}$ which is derived from the inner product

$$\langle x, y\rangle_{A^{-1}} = \langle x, A^{-1} y\rangle.$$

This is an inner product since A is symmetric and positive definite. If $b \in V$ we seek $x \in V$ s.t.

$$Ax = b. \tag{2}$$

Suppose we have an initial guess x_0 which may be zero. We define the residual r to be $r = b - Ax$, and so $r_0 = b - Ax_0$. We then define the Krylov subspace $K^n = \text{span}\,\{r_0, Ar_0, A^2 r_0, \ldots, A^n r_0\}$. The Galerkin method then finds a 'optimum' approximation to (2) by finding the unique $x_n \in x_0 + K^{n-1}$ s.t.

$$r_n = b - Ax_n \in (K^{n-1})^{\perp}. \tag{3}$$

This x_n is optimum in the sense that, for all $z \in x_0 + K^{n-1}$,

$$\begin{aligned} \|b - Az\|_{A^{-1}}^2 &= \|b - Ax_n + A(x_n - z)\|_{A^{-1}}^2 \\ &= \|b - Ax_n\|_{A^{-1}}^2 + \|A(x_n - z)\|_{A^{-1}}^2 + 2\langle b - Ax_n, A^{-1} A(x_n - z)\rangle \\ &= \|b - Ax_n\|_{A^{-1}}^2 + \|A(x_n - z)\|_{A^{-1}}^2, \end{aligned} \tag{4}$$

since the Galerkin condition (3) implies $\langle b - Ax_n, A^{-1} A(x_n - z)\rangle = 0$. So we see that x_n minimizes the A^{-1}-norm of the residual $b - Az$ for all $z \in x_0 + K^{n-1}$.

Conversely, if $x_n \in x_0 + K^{n-1}$ minimizes the A^{-1}-norm of the residual then, for all $z \in K^{n-1}$,

$$\begin{aligned} &\frac{d\|b - A(x_n + \alpha z)\|_{A^{-1}}^2}{d\alpha}\bigg|_{\alpha = 0} = 0 \\ &\Rightarrow \frac{d}{d\alpha}(\|b - Ax_n\|_{A^{-1}}^2 + 2\alpha\langle b - Ax_n, A^{-1}Az\rangle + \alpha^2\langle z, Az\rangle)|_{\alpha = 0} = 0 \\ &\Rightarrow \langle b - Ax_n, z\rangle + \alpha\langle z, Az\rangle|_{\alpha = 0} = 0 \\ &\Rightarrow b - Ax_n \in (K^{n-1})^{\perp}, \end{aligned} \tag{5}$$

i.e. (3) is satisfied.

2.1. *Practical formula for x*

The above is all very well but does not give a practical method for finding the solution x_n. By exploiting various orthogonality relations we can derive a method for finding x_{n+1} from x_n and certain other vectors.

Let us define the vector $\tilde{p}_k$ (the 'search direction') by $\tilde{p}_k = x_{k+1} - x_k$. We can find the length of $\tilde{p}_k$ as follows: we know that x_{k+1} minimizes $\|b - Ax_{k+1}\|_{A^{-1}}$, i.e. if p_k is a vector in the direction $\tilde{p}_k$

and α_k is the ratio of the lengths of $\tilde{p}_k$ and p_k, then

$$\frac{d}{d\alpha}\|b - A(x_k + \alpha p_k)\|^2_{A^{-1}}|_{\alpha=\alpha_k} = 0$$

$$\Rightarrow -2\langle b - Ax_k, p_k\rangle + 2\alpha_k\langle p_k, Ap_k\rangle = 0$$

$$\Rightarrow \alpha_k = \langle b - Ax_k, p_k\rangle/\langle p_k, Ap_k\rangle$$

$$\text{i.e. } \alpha_k = \langle r_k, p_k\rangle/\langle p_k, Ap_k\rangle. \tag{6}$$

Hence, if p_k is in the direction of $\tilde{p}_k$,

$$\tilde{p}_k = p_k\langle r_k, p_k\rangle/\langle p_k, Ap_k\rangle. \tag{7}$$

We now prove two lemmas concerning some properties of the residuals and the search directions.

Lemma 1 (orthogonality and conjugacy of r_k and p_k)

The residuals and the search directions have the following properties.

A. $\langle r_i, r_j\rangle = 0$ if $i \neq j$.
B. $\langle r_i, p_j\rangle = 0$ if $i > j$.
C. $\langle p_i, Ap_j\rangle = 0$ if $i \neq j$.

Proof. The first relation is clear since $r_i \in (K^{i-1})^\perp$ and hence $r_i \in (K^j)^\perp$ $\forall j \leqslant i-1$. For the second, $p_j = (x_{j+1} - x_j) \in K^j$ and $r_i \in (K^{i-1})^\perp$. Further, $Ap_i = r_{i+1} - r_i \in (K^{i-1})^\perp$ so $\langle Ap_i, p_j\rangle = 0$ if $j < i$, and 1C follows because A is self-adjoint. □

Lemma 2 (span of p's and r's)

The span of $\{r_0, r_1, \ldots, r_n\}$ equals the span of $\{p_0, p_1, \ldots, p_n\}$, which equals K^n.

Proof. We prove this by induction. We note that since $x_1 \in x_0 + K^0$ we have $p_0 \in K^0$, so it is true for $n = 0$. Suppose it is true for k. Then $r_{k+1} = r_k + Ap_k$, by definition of p, so $r_{k+1} \in \text{span}\{K^k, AK^k\} \subset K^{k+1}$. Also, $p_{k+1} = x_{k+2} - x_{k+1} \in K^{k+1}$ since $x_l \in x_0 + K^{l-1}$. The assertion then follows from the fact that the p_k are conjugate and therefore linearly independent, whereas the r_k are orthogonal and therefore linearly independent. □

Now let us derive the recurrence formula for p. Since $p_k \in K^k$ we can write

$$p_n = \alpha_n\left(r_n + \sum_{j=0}^{n-1} \beta_n^j p_j\right) \tag{8}$$

for some scalars α_n and β_n^j. Further, from Lemma 1C,

$$\langle r_n, Ap_l\rangle + \beta_n^l\langle p_l, Ap_l\rangle = 0, \quad l = 0, \ldots, n-1. \tag{9}$$

If $l = 0, \ldots, n-2$, $Ap_l \in K^{n-1}$, so that $\langle r_n, Ap_l\rangle = 0$; thus we see that

$$\beta_n^j = 0, \quad j = 0, \ldots, n-2.$$

Therefore

$$p_n = \alpha_n(r_n + \beta_n^{n-1} p_{n-1}). \tag{10}$$

We then define $\beta_n = \beta_{n+1}^n$ and use the fact that p_n, p_{n-1} are conjugate to deduce that

$$\beta_n = -\langle r_{n+1}, Ap_n\rangle/\langle p_n, Ap_n\rangle. \tag{11}$$

Therefore, if we have a search direction and a residual, (6), (11) and (10) define a new search direction, which gives an updated approximation and a new residual. So starting from the Galerkin property (3) we have derived the formulae for CG, i.e.

Algorithm: CG

$r_0 = p_0 = b - Ax_0$.
Repeat until convergence:
$\alpha_n = \langle r_n, p_n\rangle / \langle p_n, Ap_n\rangle$
$x_{n+1} = x_n + \alpha_n p_n$
$r_{n+1} = r_n - \alpha_n Ap_n$
$\beta_n = -\langle r_{n+1}, Ap_n\rangle / \langle p_n, Ap_n\rangle$
$p_{n+1} = r_{n+1} + \beta_n p_n$.

2.2. *Alternative formulae for α and β*

The orthogonality and conjugacy relations (Lemma 1) can be used to derive alternative formulae for α and β. Since $\langle r_n, p_{n-1}\rangle = 0$ and $p_n = r_n + \beta_n p_{n-1}$, (6) can be replaced by

$$\alpha_n = \langle r_n, r_n\rangle / \langle p_n, Ap_n\rangle. \tag{12}$$

To find the alternative for β we note that

$$\langle r_{n+1}, r_{n+1}\rangle = \langle r_n - \alpha_n Ap_n, r_{n+1}\rangle = -\alpha_n \langle Ap_n, r_{n+1}\rangle$$

and

$$\langle r_n, r_n\rangle = \langle p_n, r_n\rangle = \langle p_n, r_{n+1} + \alpha_n Ap_n\rangle = \alpha_n \langle p_n, Ap_n\rangle,$$

so that (11) can be replaced by

$$\beta_n = \langle r_{n+1}, r_{n+1}\rangle / \langle r_n, r_n\rangle. \tag{13}$$

Wong investigated these variants in the context of BICG[11] and found that (6) and (11), the 'natural' versions, were marginally superior in terms of stability with respect to round-off error. When we derive the CGS algorithm we shall need to use the alternative formula for β.

2.3. *A view of CG in terms of polynomials*

A useful way of looking at CG in terms of error analysis and generating new algorithms is to observe that $r_n \in K^n$, so the algorithm can be regarded as building up polynomials in A applied to r_0, i.e. we have $r_n = \phi_n(A)r_0$, similarly, $p_n = \theta_n(A)r_0$, where ϕ_n and θ_n are polynomials of degree n. We can therefore write an equivalent algorithm:

Algorithm: CG–polynomial

$\phi_0 = \theta_0 = 1$.
Repeat until convergence:
$\alpha_n = \langle \phi_n, \theta_n\rangle / \langle \theta_n, \psi\theta_n\rangle$
$\phi_{n+1} = \phi_n - \alpha_n \psi\theta_n$
$\beta_n = -\langle \phi_{n+1}, \psi\theta_n\rangle / \langle \theta_n, \psi\theta_n\rangle$
$\theta_{n+1} = \phi_{n+1} + \beta_n \theta_n$.

Here the inner product on polynomials is defined by

$$\langle \lambda, \mu\rangle = (\lambda(A)r_0)^{\mathrm{T}}(\mu(A)r_0) \tag{14}$$

and ψ is the polynomial such that $\psi(x) = x$. This is an inner product for polynomials of degree less than n, where n is the lowest integer such that $r_n = 0$ (otherwise $\lambda(A)r_0$ might be identically zero for non-zero polynomials). We shall use this form in the subsection below to derive a version of Lemma 1 from the CG algorithm that will also hold for BICG

2.4. Derivation of Galerkin condition from CG algorithm

We can show that the CG algorithm given above implies Lemma 1, and so since the *nth* residual generated by CG satisfies the Galerkin condition (3) it is minimal in the norm $\|:\|_{A^{-1}}$.

Lemma 3

If ϕ_k and θ_k are generated by the CG process, $k = 0, \ldots, m$ (where CG has not converged at step $m-1$), then:

A. $\langle \phi_i, \phi_j \rangle = 0$ if $i \neq j$.
B. $\langle \phi_i, \theta_j \rangle = 0$ if $i > j$.
C. $\langle \theta_i, \psi\theta_j \rangle = 0$ if $i \neq j$.

Proof. We proceed by induction, using the formulae for ϕ_i and θ_i. The statement is certainly true for $m = 0$. Suppose it is true for $m = k-1$. We first extend 3B. If $j < k-1$ then $\langle \phi_k, \theta_j \rangle = \langle \phi_{k-1} - \alpha_{k-1}\psi\theta_{k-1}, \theta_j \rangle = 0$ by the inductive hypotheses 3B and 3C. If $j = k-1$ then

$$\begin{aligned}\langle \phi_k, \theta_j \rangle &= \langle \phi_{k-1} - \alpha_{k-1}\psi\theta_{k-1}, \theta_{k-1} \rangle \\ &= \langle \phi_{k-1}, \theta_{k-1} \rangle - \alpha_{k-1}\langle \psi\theta_{k-1}, \theta_{k-1} \rangle \\ &= 0\end{aligned}$$

by the form of α. We have therefore extended 3B to the case $m = k$. Now consider 3A. If $j < k-1$ then using the newly extended 3B we see that

$$\langle \phi_k, \phi_j \rangle = \langle \phi_k, \theta_j - \beta_{j-1}\theta_{j-1} \rangle = 0.$$

If $j = k-1$ then

$$\begin{aligned}\langle \phi_k, \phi_{k-1} \rangle &= \langle \phi_{k-1} - \alpha_{k-1}\psi\theta_{k-1}, \phi_{k-1} \rangle \\ &= \langle \phi_{k-1}, \theta_{k-1} - \beta_{k-2}\phi_{k-2} \rangle - \alpha_{k-1}\langle \psi\theta_{k-1}, \theta_{k-1} - \beta_{k-2}\theta_{k-2} \rangle \\ &= 0\end{aligned}$$

by 3A, 3B and the form of α. This extends 3A. Lastly, consider 3C. If $j < k-1$ then using 3C and the extended 3A we see that

$$\begin{aligned}\langle \theta_k, \psi\theta_j \rangle &= \langle \phi_k + \beta_{k-1}\theta_{k-1}, \psi\theta_j \rangle \\ &= \langle \phi_k, (\phi_j - \phi_{j+1})/\alpha_j \rangle \\ &= 0.\end{aligned}$$

If $j = k-1$ then

$$\begin{aligned}\langle \theta_k, \psi\theta_{k-1} \rangle &= \langle \phi_k + \beta_{k-1}\theta_{k-1}, \psi\theta_{k-1} \rangle \\ &= \langle \phi_k, \psi\theta_{k-1} \rangle + \beta_{k-1}\langle \theta_{k-1}, \psi\theta_{k-1} \rangle \\ &= 0\end{aligned}$$

by the form of β. This extends 3C and completes the proof. □

2.5. *Derivation of BICG*

In this section we derive the BICG algorithm. Our starting point will be the algorithm CG–polynomial (although BICG can be derived as a Petrov–Galerkin method with test space $L_n = \{r_0, A^T r_0, \ldots (A^T)^n\}$ instead of the same test space as the trial space, K_n; see Reference 12). To derive the algorithm for BICG we shall define a 'pseudo inner product' $\langle . , . \rangle_p$ on polynomials by

$$\langle \lambda, \mu \rangle_p = (r_0)^T(\lambda(A)\mu(A)r_0). \tag{15}$$

This is symmetric and bilinear but *not* positive definite for all A. Clearly, if A is symmetric, (14) and (15) are identical. We can now define a new algorithm, the BICG algorithm, by writing the CG algorithm with the pseudo inner product used to define α and β. For the practical form we shall need to find numbers such as $\langle \phi_n, \phi_n \rangle_p$ for the scalar α. To find these we shall need new vectors to carry along either $\phi_n^2(A)r_0$ or $\phi_n(A^T)r_0$. We chose the latter; defining $\bar{r}_n = \phi_n(A^T)r_0$ and $\bar{p}_n = \theta_n(A^T)r_0$ we arrive at the BICG algorithm:

Algorithm: BICG

$r_0 = p_0 = \bar{r}_0 = \bar{p}_0 = b - Ax_0.$
Repeat until convergence:

$\alpha_n = \langle \bar{p}_n, r_n \rangle / \langle \bar{p}_n, Ap_n \rangle$

$x_{n+1} = x_n + \alpha_n p_n$

$r_{n+1} = r_n - \alpha_n A p_n$

$\bar{r}_{n+1} = \bar{r}_n - \alpha_n A^T \bar{p}_n$

$\beta_n = \langle \bar{r}_{n+1}, Ap_n \rangle / \langle \bar{p}_n, Ap_n \rangle$

$p_{n+1} = r_{n+1} + \beta_n r_n$

$\bar{p}_{n+1} = \bar{r}_{n+1} + \beta_n \bar{r}_n.$

It is valid for unsymmetric matrices, and reduces to CG in the symmetric case. This is simply the CG–polynomial algorithm written out in terms of vectors using the definition of the pseudo inner product. Because of this the orthogonality relations (Lemma 1) still apply. Therefore the Galerkin condition (3) also applies but only in the pseudo inner product. The optimality property (4) no longer applies since the A^{-1} 'inner product' is no longer an inner product and does not define a norm.

2.6. *Derivation of CGS*

We proceed from the assumption that if the residual is $\phi_n(A)r_0$ it would be even better for it to be $\phi_n^2(A)r_0$. We can achieve this by 'squaring' the algorithm CG–polynomial to produce the CGS algorithm. We note that although, if the algorithm converges, we can expect $\|\phi_n(A)r_0\| < \|\phi_0(A)r_0\|$, this does not imply that $\phi_n(A)$ is a contraction and so we are not guaranteed that $\|\phi_n^2(A)r_0\| < \|\phi_n(A)r_0\|$. However, if the matrix is well preconditioned the CGS algorithm seems to work better in practice.

For CGS we shall use the following forms for the scalars α and β that occur in the BICG algorithm:

$$\alpha_n = \langle \bar{p}_n, r_n \rangle / \langle \bar{p}_n, Ap_n \rangle, \tag{16}$$

$$\beta_n = \langle \bar{r}_{n+1}, r_{n+1} \rangle / \langle \bar{r}_n, r_n \rangle. \tag{17}$$

Note that the formula for β is *not* the 'best' version for BICG as recommended by Yong,[11] but, as will become clear below, these are the only ones that can be used with CGS.

The central relations of the CG–polynomial algorithm are

$$\phi_{n+1} = \phi_n - \alpha_n \psi \theta_n, \tag{18}$$

$$\theta_{n+1} = \phi_{n+1} + \beta_n \theta_n. \tag{19}$$

If we square relations (18) and (19) and form $\theta_n \times$ (18) and $\phi_{n+1} \times$ (19) we obtain (using (23) to simplify (20))

$$\phi_{n+1}^2 = \phi_n^2 - 2\alpha_n \psi \theta_n \phi_n + \alpha_n^2 \psi^2 \theta_n^2 \tag{20}$$

$$= \phi_n^2 - \alpha_n \psi(\theta_n \phi_{n+1} + \theta_n \phi_n), \tag{21}$$

$$\theta_{n+1}^2 = \phi_{n+1}^2 + 2\beta_n \theta_n \phi_{n+1} + \beta_n^2 \theta_n^2, \tag{22}$$

$$\phi_{n+1}\theta_n = \phi_n \theta_n - \alpha_n \psi \theta_n^2, \tag{23}$$

$$\phi_{n+1}\theta_{n+1} = \phi_{n+1}^2 + \beta_n \theta_n \phi_{n+1}. \tag{24}$$

We then have the following formula for α and β:

$$\alpha_n = \langle \phi_n, \theta_n \rangle_p / \langle \theta_n, \psi \theta_n \rangle_p = \langle 1, \phi_n \theta_n \rangle_p / \langle 1, \psi \theta_n^2 \rangle_p, \tag{25}$$

$$\beta_n = \langle \phi_{n+1}, \phi_{n+1} \rangle_p / \langle \phi_n, \phi_n \rangle_p = \langle 1, \phi_{n+1}^2 \rangle_p / \langle 1, \phi_n^2 \rangle_p. \tag{26}$$

Defining

$$R_n = \phi^2(A) r_0, \tag{27}$$

$$P_n = \theta^2(A) r_0, \tag{28}$$

$$K_n = \theta_n \phi_n(A) r_0, \tag{29}$$

$$H_n = \phi_{n+1} \theta_n(A) r_0 \tag{30}$$

and X_n to be the nth approximate solution, we see that we have everything required to convert (20–26) into an algorithm:

Algorithm: *CGS*

$R_0 = P_0 = K_0 = b - AX_0$.

Repeat until convergence:

$\alpha_n = r_0^T K_n / r_0^T A P_n$

$H_n = K_n - \alpha_n A P_n$

$R_{n+1} = R_n - \alpha_n A(H_n + K_n)$

$X_{n+1} = X_n + \alpha_n (H_n + K_n)$

$\beta_n = r_0^T R_{n+1} / r_0^T R_n$

$P_{n+1} = R_{n+1} + 2\beta_n H_n + \beta^2 P_n$

$K_{n+1} = R_{n+1} + \beta_n H_n$.

There is an unfortunate clash of commonly used notation here: $K_n = \theta_n \phi_n(A) r_0$ as used here should be distinguished from K^n, the Krylov subspace.

2.7. Choice of formula for β

We shall briefly consider the choice of formula for β. For BICG we would normally choose to use

$$\beta_n = -\langle \phi_{n+1}, \psi\theta_n \rangle / \langle \theta_n, \psi\theta_n \rangle, \tag{31}$$

which is equivalent to (26) in exact arithmetic. In terms of CGS-type variables, the numerator of (31) is given by

$$\beta_n = \langle 1, \psi\phi_{n+1}\theta_n \rangle_p = r_0^T A H_n,$$

but to find AH_n would involve an extra matrix–vector multiplication per iteration step.

Fortunately, we observe that the two formulae (26) and (31) will behave identically until rounding errors become important. With a good enough preconditioner this will not occur. Indeed, one can say that if rounding errors have become significant then convergence is taking too many iterations and is liable to be uneconomic.

2.8. Derivation of CGR

We consider the CGR algorithm, which we shall derive from a Petrov–Galerkin process. We retain the definition of K^n, and as for CG we consider $x_n \in x_0 + K^{n-1}$. However, we define a new space $L^n = AK^n$ and use this as the test space, i.e. we seek x_n s.t.

$$r_n = b - Ax_n \in (L^{n-1})^{\perp}. \tag{32}$$

In this case x_n is optimum in the sense that, for all $z \in x_0 + K^{n-1}$,

$$\begin{aligned} \|b - Az\|^2 &= \|b - Ax_n + A(x_n - z)\|^2 \\ &= \|b - Ax_n\|^2 + \|A(x_n - z)\|^2 + 2\langle b - Ax_n, A(x_n - z)\rangle \\ &= \|b - Ax_n\|^2 + \|A(x_n - z)\|^2, \end{aligned} \tag{33}$$

since the Petrov–Galerkin condition (32) implies $\langle b - Ax_n + A(x_n - z)\rangle = 0$. So we see that x_n minimizes the norm of the residual $b - Az$ for all $z \in x_0 + K^{n-1}$. Similarly we can prove that if x_n minimizes the residual then (32) is satisfied. This is true for unsymmetric A since we do not need to invoke an A^{-1} norm.

As an aside, notice that we could consider this method as a version of CG if we redefine the concept of 'orthogonal' to mean 'conjugate'. Then the Galerkin condition from which we derived CG becomes

$$\langle r_n, v \rangle_A = 0 \quad \forall v \in K^{n-1} \tag{34}$$

$$\Rightarrow \langle r_n, w \rangle = 0 \quad \forall w \in L^{n-1}, \tag{35}$$

which is (32).

In a similar manner to our derivation of CG we derive the CGR algorithm for unsymmetric matrices from (32). As before, $\tilde{p}_k = x_{n+1} - x_n$. We can find the length of $\tilde{p}_k$ as follows. We know that x_{k+1} minimizes $\|b - Ax_{k+1}\|$, so if p_k is a vector in the direction $\tilde{p}_k$ and α_k is the ratio of the

lengths of $\tilde{p}_k$ to p_k, then

$$\frac{d}{d\alpha}\|b - A(x_k + \alpha p_k)\|^2|_{\alpha=\alpha_k} = 0$$

$$\Rightarrow -2\langle b - Ax_k, Ap_k\rangle + 2\alpha_k\langle Ap_k, Ap_k\rangle = 0$$

$$\Rightarrow \alpha_k = \langle b - Ax_k, Ap_k\rangle/\langle Ap_k, Ap_k\rangle$$

$$\text{i.e. } \alpha_k = \langle r_k, Ap_k\rangle/\langle Ap_k, Ap_k\rangle. \tag{36}$$

Hence, if p_k is in the direction of $\tilde{p}_k$,

$$\tilde{p}_k = p_k\langle r_k, Ap_k\rangle/\langle Ap_k, Ap_k\rangle. \tag{37}$$

The following modification of Lemma 1 (whose proof we omit since it is similar to the proof of Lemma 1) holds.

Lemma 4. (*Conjugacy and span of r_k and p_k*)

The residuals and the search directions have the following properties:

A. $\langle r_i, Ar_j\rangle = 0$ if $i > j$.
B. $\langle r_i, Ap_j\rangle = 0$ if $i > j$.
C. $\langle Ap_i, Ap_j\rangle = 0$ if $i \neq j$.
D. $\text{Span}\{r_0, r_1, \ldots, r_n\} = \text{span}\{p_0, p_1, \ldots, p_n\} = K^n$.

A significant deviation from the CG algorithm occurs in the derivation of the new search direction.

As before, we can write

$$p_n = \alpha_n\left(r_n + \sum_{j=0}^{n-1} \beta_n^j p_j\right) \tag{38}$$

for some scalars α_n and β_n^j. Multiplying the above by A, taking the inner product with Ap_l, and using Lemma 4C, we obtain

$$\langle Ar_n, Ap_l\rangle + \beta_n^l\langle Ap_l, Ap_l\rangle = 0, \quad l = 0, \ldots, n-1. \tag{39}$$

However, it is no longer true that $\langle \text{Ar}_n, Ap_l\rangle = 0$, $l = 0, \ldots, n-2$, so we have instead

$$\beta_n^l = -\langle Ar_n, Ap_l\rangle/\langle Ap_l, Ap_l\rangle, \quad l = 0, \ldots, n-1;$$

that is, the summation in (38) does not collapse to one term. This means that the algorithm becomes costly in terms of time and storage if the number of steps to reach convergence is not small. In practice, therefore, we have to either truncate the summation to a certain number of terms or restart the method after m steps, say, when m is the maximum size of summation we can permit. The former option runs the risk of omitting important terms and therefore causing failure or stagnation; the latter throws away the information we have gained in the search directions. In practice the latter option is to be preferred since the former has stagnated on all problems we have tried for which 'cut-off' was exceeded.

2.9. Summary

We have displayed the theory for CG, BICG, CGS and CGR. Of these, CG is the most common method to use for symmetric equations. For non-symmetric equations the choice is less

clear and depends upon the problem being solved. Our experience (see Section 4) is that CGR is not suitable unless reorthogonalization against all previous residuals can be afforded, and that CGS is superior to BICG if the system is well conditioned (well preconditioned). Note that CGR is mathematically equivalent to GMRES and ORTHOMIN in exact arithmetic; the restarted versions are likewise equivalent.

3. THE MODEL PROBLEM

The numerical solution of free convection, of which the 'double-glazing' problem is an example, is of great industrial importance. Applications range from the insulation of a home to the flow of coolant in a nuclear reactor. The model for natural convection in a box cavity, considered in this study, assumes the following:

(1) no flow-induced temperature variations arising from adiabatic compression or expansion or from viscous dissipation
(2) no variation in fluid properties other than density with temperature
(3) variations in density are ignored except in as much as they give rise to a gravitational force (the Boussinesq approximation)
(4) a linear dependence of density on temperature
(5) an incompressible fluid with constant heat capacity per unit volume and no internal heat generation
(6) no forced convection, so that the velocity and temperature distributions are coupled
(7) no attempt to model turbulence effects at high Rayleigh numbers
(8) steady state conditions.

The problem geometry is a square/cubic box in 2D or 3D, filled with air, of side length D, and with one wall heated to T_{hot} and one cooled to T_{cold}. To non-dimensionalize the equations, we define Prandtl and Rayleigh numbers by

$$Pr = \frac{\mu}{\rho\beta}, \qquad Ra = \frac{g\alpha D^3\rho}{\beta\mu}(T^*_{\text{hot}} - T^*_{\text{cold}})$$

and non-dimensional variables by

$$x_i^* = \frac{x_i}{D}, \qquad u_i^* = \frac{u_i D}{\beta}, \qquad T^* = -\frac{\Delta T}{T_{\text{hot}} - T_{\text{cold}}}, \qquad P^* = \frac{PD^2}{\rho\beta^2},$$

where g is the gravitational acceleration acting in the x_2-direction only, α is the coefficient of volumetric expansion and β is the coefficient of thermal diffusivity of the fluid. The system of partial differential equations describing the 'double-glazing' problem becomes

$$\frac{\partial u_i^*}{\partial x_i} = 0, \tag{40}$$

$$u_j^* \frac{\partial u_i^*}{\partial x_j} + \frac{1}{\rho}\frac{\partial P^*}{\partial x_i} - Pr\nabla^2 u^* - RaPrT^* = 0, \tag{41}$$

$$u_j^* \frac{\partial T^*}{\partial x_j} - \nabla^2 T^* = 0. \tag{42}$$

Besides the non-linearity of the convected derivative there exists a second and stronger one; it is controlled by the Rayleigh number in equation (41). The product of Rayleigh and Prandtl

numbers couples the velocity and temperature domains and acts in the local gravitational direction x_2. The equations are solved as a coupled system.

The non-dimensional system (40)–(42) is discretized by the Galerkin FEM. Tri/biquadratic basis functions are used for the velocity and temperature, and tri/bilinear (or linear) for the pressure, in 3D and 2D respectively. The resultant non-linear equations are linearized (leading to an unsymmetric matrix) by either Picard or Newton linearization.

The Picard iteration is for solution of the non-linear system

$$F(x) = 0. \tag{43}$$

In the case of the finite element discretization of the Navier–Stokes equations this can be written as

$$M(x)x = 0 \tag{44}$$

and we can partition x into $(x_r|x_G)^T$, so that the elements in x_G are those for which values are known from boundary conditions. Partitioning $M(x)$ similarly, we get

$$M_r(x)x_r = -M_G(x)x_G. \tag{45}$$

We construct the Picard iteration by solving

$$M_r(x^{(n)})x_r^{(n+1)} = -M_G(x^{(n)})x_G^{(n)}, \tag{46}$$

where the superscripts (n) and $(n+1)$ refer to iterative step numbers.

The Newton iteration differs in that we take

$$J(x^{(n+1)} - x^{(n)}) = -F(x^{(n)}), \tag{47}$$

where J is the Jacobian matrix with elements

$$J_{ij} = \left(\frac{\partial F_i(x)}{x_j}\right)_{x^{(n)}}. \tag{48}$$

The Picard iteration in general converges linearly so that

$$\|x^{(n+1)} - x\| \leqslant C\|x^{(n)} - x\|, \tag{49}$$

where C is a constant and x is the true or exact solution, whereas the Newton iteration gives quadratic convergence,

$$\|x^{(n+1)} - x\| \leqslant C\|x^{(n)} - x\|^2. \tag{50}$$

The counterbalancing advantage of Picard is that the region of x-space from which it will converge to the true solution is much larger than that for Newton. Picard can therefore be helpful when it is difficult to obtain a sufficiently close starting approximation.

The matrix resulting from the Picard iteration involves interaction between each unknown at a given node and all unknowns at those nodes which share the same element(s). As a simple 2D example, an unknown at a corner node where four elements meet interacts with all unknowns in every node of each of the four elements. For the Newton iteration the interactions are between the same sets of unknowns but the matrix elements involve extra terms. Finite difference methods in general give rise to fewer interactions. The matrices are therefore sparser but, because they represent the same fundamental differential equations, are not likely to differ greatly in their condition number. Although the authors have not studied the behaviour of iterative methods with finite difference discretizations, they have no reason to think that rates of convergence would be markedly different. We find that the condition of the matrix we obtain is not sensitive to the

exact type of discretization we use, so long as this satisfies the conditions for a well posed problem.

4. COMPARISON OF DIFFERENT ITERATIVE METHODS

We shall contrast the three ILU-preconditioned CG-like methods derived in Section 2 by comparing their performance on a 2D version of the model problem, with a uniform mesh of 10 elements (biquadratic in u, v, t and continuous bilinear in p) to a side. We use 'continuation' in the Rayleigh number Ra (see Reference 13) to advance from 10^3 to 10^6; we do two Newton linearizations at a given Ra and use the ratio of these two steps to determine the increment in Ra. The convergence criterion used for the methods was

$$\|r_n\| \leqslant \|r_0\| \times 10^{-3}.$$

This may seem lax, but it was adequate to preserve the 'continuation path' or sequence of Rayleigh numbers generated; a tighter convergence criterion would have wasted effort, since intermediate steps need only be solved sufficiently accurately to enable the code to proceed to the next Ra.

Consider Figure 1, which shows the number of iterations required at each Newton step. We see that the CGS method appears superior, although its behaviour is somewhat more erratic than that of the BICG algorithm. The CGR method is on balance worst, particularly for higher Rayleigh numbers. We have used the restarted method, with restart every 10th and 20th step. The CGR methods failed to converge within 150 iterations for the last two Newton steps, although they were converging. Each method has roughly the same cost per iteration, although the restarted CGRn algorithms need up to an extra n inner products and vector additions per iteration.

Notice that CGS and CGR show a pronounced oscillation in the number of iterations needed for convergence, with peaks at even Newton steps. This is because we redo the ILU factorization when we change Ra, and so the second Newton step of the pair is less well preconditioned. The peaks in CGS and CGR seem to indicate that these methods need a good preconditioner to converge.

We have also studied the performance of the ILU-preconditioned unsymmetric CG methods, on a scalar machine, on 2D problems other than the model problem, namely laminar Stokes flow over a backward-facing step (formulated as in (1) with $L = 0$ and $M = 0$). Computational times are significantly lower than for a sparse direct method (i.e. an unsymmetric frontal solver where the maximum frontwidth gets minimized by element-renumbering techniques). The CPU cross-over point (the smallest problem for which the iterative method becomes faster than the direct

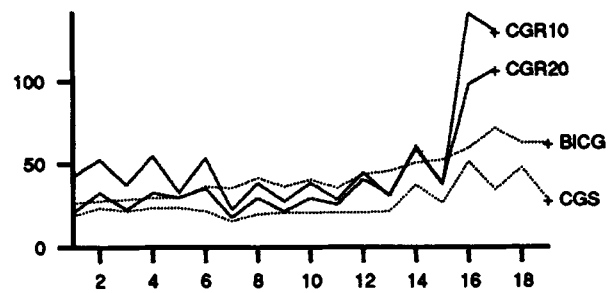


Figure 1. Different iterative methods: number of iterations to convergence plotted against Newton step

solution) appears for smaller sets of equations than in 3D discretizations, and speed-ups of two or three are to be expected with 2D elements even for coarse meshes, even on problems where the frontwidth is small, and even with quadratic six-noded Taylor–Hood triangles, which we have tested. Memory (in-core storage of the sparse matrix and its preconditioner) would appear not to present a problem in 2D. When the meshes are finer the increased sparsity results in greater advantage for the iterative method. We speculate that there may well exist situations where, by using the unsymmetric CG method rather than the frontal method, one could extend the range of Reynolds numbers for which it could be possible to retain the Galerkin test functions (rather than introduce Petrov–Galerkin or upwinding). This is because the 'cell' Reynolds numbers can be kept small through smaller CPU costs for mesh refinement. It has also been observed[14,15] that the performance of the iterative method significantly deteriorates with the Galerkin approximation as the cell Peclet or Reynolds number increases, but that it regains its competitive edge when one adopts a Petrov–Galerkin method.

From the computations thus far we conclude that the ILU-preconditioned CGS method appears best for this class of problems.

5. PROGRAMMING CONSIDERATIONS

In this section we remark on some of the aspects of implementation of the unsymmetric CG methods and of sparse Gauss elimination methods on vector computers (i.e. Cray XMP and Cray 2 series). These represent our experience to date with the implementation of two of the algorithms and are by no means conclusive. The Gauss elimination implementation chosen in this work is the unsymmetric frontal solver developed by Hood[16] based on the frontal philosophy postulated by Irons.[17] This method has an advantage of a smaller active memory and arbitrary nodal numbering when compared to an unsymmetric bandwidth solver. The CG method tested in a 3D context on the Cray machines is the BICG method[18] which has been described in Section 2.5.

We consider the 3D version of the model problem. Our objective is to study 3D finite elements which employ high-order test and trial functions (triquadratic); these give best results for a fixed number of unknowns when the solution is of sufficient smoothness. However, the global matrices resulting from such discretizations are not very sparse for any realistic computation owing to the large support of the functions (i.e. each node has many neighbours). This lack of sparsity is more unfavourable to the iterative method. Conversely, the direct method is at odds with the model problem because the frontwidth always increases with mesh refinement (Section 6). Our objective is to locate the crossover point, the size of problem for which the iterative method becomes more CPU-competitive than the sparse direct elimination method. A crucial consideration is the relative optimality of implementation of the algorithms on the vector computer; this is of significance to the dynamics of the crossover point. Finally, it is also important to understand how much in-core memory is required at the crossover point because this can preclude the practical use of the direct or iterative method.

The CPU growth curve with number of unknowns is much sharper for Gauss elimination than for BICG. However, the frontal philosophy has one notable advantage over CG methods, one which concerns memory management: equations are written to out-of-core memory in succession and are only required once for the back-substitution stage. Unfortunately, the iterative nature of CG requires the presence of the sparse matrix at each iteration. If the method is preconditioned by an incomplete factorization, as much memory again is required to store these factors. Hence we find that the practical realization of these methods depends crucially on the availability of fast and cheap memory. On the Cray 2 we do have enough memory available to investigate quite

large finite element problems; on the Cray XMP memory was a limiting factor. The local memory of MIMD machines, such as the transputer, lends itself to a storage of all the individual element matrices with the sparse vector by matrix multiplications of BICG performed on an element-by-element basis. This procedure, however, is not so attractive on supercomputers since it involves considerable extra memory owing to the equation overlap of individual finite element matrices. On vector computers it is therefore advantageous to store the sparse matrix system in a vector with pointers for column number and row start, as described for the symmetric case in Reference 2. When ILU preconditioning is required it may be desirable to retain some zero contributions occurring naturally (i.e. the pressure contribution to the continuity equation, which can be zero) for better fill-in.

There is a fundamental difference between the efficient coding of a Gauss elimination and CG methods on vector machines such as Cray XMP and Cray 2. The CPU efficiency of the CG method depends on the spectral attributes of the matrix problem under consideration, while the Gauss elimination expends the same amount of work irrelevant of this, and CPU expenditure is solely controlled by the size of the frontwidth and the number of unknowns in the system. A frontal solver is therefore more predictable than CG, where the number of iterations, and hence the relative importance of the sparse matrix by vector multiplications, will vary depending on spectral considerations. This means that areas of code which consume most of the CPU time are more readily identified and optimized in the frontal solver.

In addition to this feature, the subtraction of the pivotal equation multiplied by the Gauss factor from the current equation is by far the overwhelming CPU-consuming operation in Gauss elimination. An example is given by our 3D implementation where the Cray XMP percentage CPU stays above 60% even when the portion of code in charge of this operation is highly optimized. This means that Amdahl's Law, a law of diminishing returns, does not significantly affect our frontal solver implementation — optimize a few loops and you need not worry about optimizing the rest of your code!

An efficient CAL* routine called PIX† was employed for the task of performing the Gauss factor multiplication and equation subtraction mentioned earlier. PIX can be described by the following pseudo-code:

```
for j = a1 to b1                    /* a1 = 1 or lpc + 1; b1 = lpc − 1 or lcol
for i = a2 to b2                    /* a2 = 1 or lpr + 1; b2 = lpr − 1; or lrow
e(i + δ, j + δ) = e(i, j) − p(i) * q(j)   /* δ = 0 or −1
next i
next j
```

where e is the frontal matrix, q is the normalized pivotal row, p is the vector of Gauss factors and *lpr*, *lpc*, *lrow* and *lcol* are the pivotal row, pivotal column, number of rows and number of columns in the matrix respectively. Judicious use of the WHENEQ Cray function in the prefront activity and other loops inside the frontal routine together with forced vectorization of loops in the element assembly stages resulted in a code that consistently sustained a 100 Mflop rate (on a single processor on the Cray XMP 148 with background jobs running on the other processors).

The overwhelming CPU dominance of a single task or operation is less with the BICG method than with Gauss elimination. In addition, the average megaflop rate of the BICG code is inferior to the Gauss elimination code. This is due to the indirect addressing of the sparse matrix by vector

* Cray Assembler Language.
† Written by A. Mills of Cray U.K.

multiply Cray intrinsic function SPAXPY; a contributing factor is that SPAXPY is usually presented with a shorter vector length than PIX—the frontwidth is usually bigger than the number of unknowns neighbouring a node.

We employed Hood's frontal solver equipped with PIX on the Cray XMP and the MA32 direct solver[19] on the Cray 2. However, MA32 is similar to Hood's routine containing the Cray 2 CAL version of PIX.

6. RESOURCE ESTIMATES AND WORK ON THE CRAY 2

The system of equations generated by the FEM become extremely large as the size of the problem increases. Let us estimate the CPU time and storage requirements for both the frontal method and ILU-preconditioned BICG for a regular cuboid or square mesh, in d dimensions ($d = 2$ or 3), with g elements to a side. We shall tentatively assume that the time required is proportional to the operation count; we return to this point later in respect of vectorization and indirect addressing. To simplify the estimates we assume that all rows of the matrix have the same number of entries. In practice, owing to edge effects and different node types, most rows will have fewer than the maximum, so the average number will be assumed to be half the maximum. We can obtain the following estimates for the variables e, n, z, b, s, f, t_f, t_i, t_b:

dimension	d		
elements to a side	g		
maximum entries per row	e_m		$= 5^d(d+1)$
Average entries per row	e		$= 5^d(d+1)/2$
number of equations	n		$= (1+d)2^d g^d$
number of entries in matrix	z	$= en$	$= (1+d)^2(10g)^d/2$
half-bandwidth	b		$= g^{d-1}(1+d)2^{d-1}$
space for L, U factors (each)	s	$= bn$	$= (1+d)^2 2^{2d-1} g^{2d-1}$
space for front	f	$= b^2$	$= g^{2(d-1)}(1+d)^2 2^{2d-2}$
time for frontal solution	t_f	$= b^2 n$	$= (2g)^{3d-2}(1+d)^3$
time for ILU decomposition	t_i	$= e^2 n/2$	$= 5^{2d}(1+d)^3 2^d g^d/8$
Time for one ILU-BICG step	t_b	$= 4en$	$= 2[10^d(1+d)^2 g^d]$.

Then we can construct tables showing the growth laws for solution time and storage space in two and three dimensions.

First consider the storage requirements shown in Table I. Note first that for the frontal method the matrix itself is never fully assembled and the L, U factors are not stored in memory but on disc, so that the storage-s is not required in core; however, twice this many reals must be transferred to and from the disc. In 3D the asymptotically dominating figure is the front space f, whereas in 2D the dominant figure is the matrix storage z. In 3D the point where the front takes

Table I. Storage estimates

	Storage required	2D	3D
Iterative	$2z$	$900g^2$	$16000g^3$
Direct	f	$36g^2$	$256g^4$

more storage than the whole matrix occurs at around $g = 64$ which is an impracticably large problem for current computers. Now consider the time requirements shown in Table II. We see that we need one more number to complete the estimates—the number of iterations needed to solve the equations. This cannot be predicted in the way that the others can, but experience in 2D and 3D indicates that $5g$ is a suitable estimate, depending on the hardness of the problem and the convergence required. Multiplication of t_b by $5g$ still leaves t_f, the frontal solve time, as the dominant cost.

If we consider the table of actual solution times,[23] Table III (for solution to the 3D model using finite elements with triquadratic velocity/temperature and trilinear continuous pressure), we see that these theoretical calculations are considerably distorted in practice. The growth of solution time for the BICG method is clearly slower than that of the frontal method, but the growth laws are distorted by the vector efficiency achieved by the BICG and frontal methods. The former rapidly achieves its full vector length (the number of entries per matrix row), even for small problems, whereas the growing front size of the latter makes for more efficient vector lengths on larger problems, and to some extent disguises the growth in number of operations. We note that the frontal code uses direct addressing for the elimination process whereas the BICG method references the matrix via indirect addressing, which slows the process; also, a crucial section of the frontal solver was coded in CAL whereas the BICG routines were written in Fortran.

7. UNPRECONDITIONED BICG ON THE CRAY XMP/48

The memory limitations of the Cray XMP/48 discouraged the use of a preconditioner and encouraged a study of the robustness or reliability of the unpreconditioned BICG method. On a Cray 2 machine much more memory is available to store both the matrix and its preconditioner.

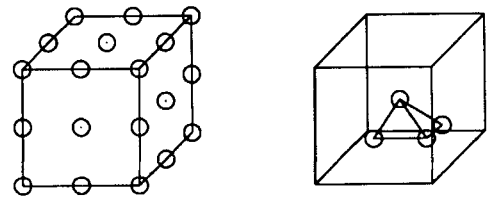
The model problem was solved with the 3D element of Figure 2, triquadratic velocity/temperature and linear discontinuous pressure: we solve for the discontinuous pressure unknowns (no penalty method). This type of 3D element had been applied with success by one of us

Table II. Time estimates

	Time required	2D	3D
Iterative	t_i	$8500g^2$	$1000000g^3$
	$t_b \times 5g$	$9000g^3$	$160000g^4$
Direct	t_f	$450g^4$	$8000g^7$

Table III. Actual solution times (Cray 2) and storage, 3D

Gauge	n	z (1000s)	Solve time		
			Direct	Iterative	
				ILU	BICG
3	684	90	0·71	0·66	0·55
4	1721	281	3·5	2·51	1·93
5	3492	646	13·78	6·71	5·6
6	6195	1250	50·0	13·4	12·7
7	10028	2110	?	23·0	24·3



velocity/temperature pressure

Figure 2. Triquadratic velocity/temperature, linear discontinuous pressure brick element

(with or without a discrete penalty method) to calculate laminar and turbulent flow through a tight 90° bend of circular cross-section based on the laser Doppler experiments of Enayet *et al.*[20] and using a direct frontal solver on the Cray XMP/48.[21,22] Elements of this kind were arranged into a box of gauge 3 (27 elements) with grading 1:2:1, and taken through a set of intermediate Rayleigh number continuation steps to a target of $Ra = 31\,000$. Here we used the Picard linearization. For the reasons discussed in Section 3 we find that the continuation process is not crucial to the Picard linearization, but that it becomes more important for problems which employ Newton's method, which has a much smaller 'ball of attraction' than the Picard linearization.

After the boundary conditions are eliminated, a matrix of 729 unconstrained temperature, pressure and velocity degrees of freedom is solved at each Picard linearization. Table IV compares the number of internal iterations or work required by BICG at each Picard step to an estimate for the spectral condition of the matrix. The estimate is the ratio between the largest and smallest pivots encountered in the frontal elimination corresponding to the same matrix problem. The pivots can give a warning of small eigenvalues, but when the matrix is unsymmetric this is only a rough guide since the computation of singular values is required. Nevertheless, from this table we notice a trend of increased BICG work with higher Rayleigh number and poorer spectral condition ratio. We find that the Average BICG steps per iteration are 276 for $Ra = 1000$, 312 for $Ra = 4000$, 392 for $Ra = 10\,000$, 439 for $Ra = 19\,000$ and 526 for $Ra = 31\,000$. The number of unconstrained degrees of freedom in this problem is 729 and the results seem to indicate that at $Ra = 31\,000$ we are slightly exceeding this number and violating the exact termination property of CG methods owing to computer round-off error. Beyond $Ra = 31\,000$ convergence cannnot be attained even with the direct method, probably owing to a numerical bifurcation which occurs much earlier than the real, or physical, bifurcation and is caused by the coarseness of the mesh. Hence we experience that for problems with moderate Rayleigh numbers the unpreconditioned BICG method is as robust and as reliable as the frontal elimination method with pivoting.

The termination criterion for convergence of the BICG method was derived from a comparison between direct and iterative methods. The problem degrees of freedom not converged at each Picard linearization are those which fail to satisfy a convergence estimator. The following are two likely candidates for estimator:

$$\Delta_1 = \max_j \frac{\max_i |{}^j\varphi_i^n - {}^j\varphi_i^{n-1}|}{\max_i |{}^j\varphi_i^{n-1}|}, \qquad \Delta_2 = \max_{i,j} \left| \frac{{}^j\varphi_i^n - {}^j\varphi_i^{n-1}}{{}^j\varphi_i^{n-1}} \right|, \tag{51}$$

where ${}^j\varphi_i$ stands for a degree of freedom numbered i, of type j (i.e. pressure, velocity, etc.), and n is the current linearization step. A converged iterative process is defined as one in which all degrees of freedom have satisfied one of the above criteria to a value less than some tolerance (i.e. 0·05 or 5% change).

Table IV. Unpreconditioned BICG versus frontal: spectral analysis and BICG iterations

Picard step	Ra	Frontal			BICG	
		Piv_{min}	Piv_{max}	$\rho(A)$	norm residual	work
2	1000	$0{\cdot}696 \times 10^{-3}$	$0{\cdot}269 \times 10^{+1}$	$0{\cdot}387 \times 10^{+4}$	$0{\cdot}989 \times 10^{-7}$	258
3	1000	$0{\cdot}695 \times 10^{-3}$	$0{\cdot}269 \times 10^{+1}$	$0{\cdot}387 \times 10^{+4}$	$0{\cdot}872 \times 10^{-7}$	295
4	4000	$0{\cdot}695 \times 10^{-3}$	$0{\cdot}269 \times 10^{+1}$	$0{\cdot}387 \times 10^{+4}$	$0{\cdot}819 \times 10^{-7}$	294
5	4000	$0{\cdot}687 \times 10^{-3}$	$0{\cdot}269 \times 10^{+1}$	$0{\cdot}392 \times 10^{+4}$	$0{\cdot}959 \times 10^{-7}$	336
6	4000	$0{\cdot}686 \times 10^{-3}$	$0{\cdot}269 \times 10^{+1}$	$0{\cdot}393 \times 10^{+4}$	$0{\cdot}772 \times 10^{-7}$	307
7	10000	$0{\cdot}687 \times 10^{-3}$	$0{\cdot}270 \times 10^{+1}$	$0{\cdot}392 \times 10^{+4}$	$0{\cdot}256 \times 10^{-7}$	332
8	10000	$0{\cdot}664 \times 10^{-3}$	$0{\cdot}270 \times 10^{+1}$	$0{\cdot}406 \times 10^{+4}$	$0{\cdot}536 \times 10^{-7}$	478
9	10000	$0{\cdot}669 \times 10^{-3}$	$0{\cdot}270 \times 10^{+1}$	$0{\cdot}404 \times 10^{+4}$	$0{\cdot}581 \times 10^{-7}$	366
10	19000	$0{\cdot}673 \times 10^{-3}$	$0{\cdot}271 \times 10^{+1}$	$0{\cdot}402 \times 10^{+4}$	$0{\cdot}683 \times 10^{-7}$	380
11	19000	$0{\cdot}637 \times 10^{-3}$	$0{\cdot}271 \times 10^{+1}$	$0{\cdot}425 \times 10^{+4}$	$0{\cdot}354 \times 10^{-7}$	504
12	19000	$0{\cdot}650 \times 10^{-3}$	$0{\cdot}272 \times 10^{+1}$	$0{\cdot}418 \times 10^{+4}$	$0{\cdot}601 \times 10^{-7}$	435
13	31000	$0{\cdot}654 \times 10^{-3}$	$0{\cdot}272 \times 10^{+1}$	$0{\cdot}416 \times 10^{+4}$	$0{\cdot}749 \times 10^{-7}$	413
14	31000	$0{\cdot}613 \times 10^{-3}$	$0{\cdot}273 \times 10^{+1}$	$0{\cdot}445 \times 10^{+4}$	$0{\cdot}562 \times 10^{-7}$	623
15	31000	$0{\cdot}633 \times 10^{-3}$	$0{\cdot}274 \times 10^{+1}$	$0{\cdot}433 \times 10^{+4}$	$0{\cdot}912 \times 10^{-7}$	476
16	31000	$0{\cdot}631 \times 10^{-3}$	$0{\cdot}274 \times 10^{+1}$	$0{\cdot}434 \times 10^{+4}$	$0{\cdot}603 \times 10^{-7}$	463
17	31000	$0{\cdot}626 \times 10^{-3}$	$0{\cdot}274 \times 10^{+1}$	$0{\cdot}437 \times 10^{+4}$	$0{\cdot}280 \times 10^{-7}$	538
18	31000	$0{\cdot}632 \times 10^{-3}$	$0{\cdot}274 \times 10^{+1}$	$0{\cdot}433 \times 10^{+4}$	$0{\cdot}597 \times 10^{-7}$	523
19	31000	$0{\cdot}628 \times 10^{-3}$	$0{\cdot}274 \times 10^{+1}$	$0{\cdot}436 \times 10^{+4}$	$0{\cdot}397 \times 10^{-7}$	484
20	31000	$0{\cdot}629 \times 10^{-3}$	$0{\cdot}274 \times 10^{+1}$	$0{\cdot}435 \times 10^{+4}$	$0{\cdot}696 \times 10^{-7}$	487
21	31000	$0{\cdot}630 \times 10^{-3}$	$0{\cdot}274 \times 10^{+1}$	$0{\cdot}435 \times 10^{+4}$	$0{\cdot}941 \times 10^{-7}$	734

The second criterion Δ_2 is unstable in the presence of a discrete zero value of the solution. Δ_1 is, however, stable but is not as appropriate as Δ_2 when we are interested in the local accuracy and flow orientation in regions where the velocities may be small in comparison with the bulk flow. We choose Δ_2 in this experiment and denote the number of problem unknowns not satisfying this condition, to a given tolerance, by N_2 in Table V. This gives values of N_2 for various choices of the BICG norm residual termination criterion and compares them with the solution by the frontal direct method at each Picard linearization at $Ra = 1000$. It can be seen that only a norm residual of 10^{-7} or smaller (or square of norm residual smaller than 10^{-14}) is suitable to match the results by the direct solver. It is debatable whether the choice of a larger norm residual convergence criterion early in the Rayleigh continuation process followed by a very small criterion at the target Rayleigh number will reduce the overall cost of a computation. We believe that will depend on the right balance between the work required for extra linearizations incurred by a larger residual convergence criterion and total BICG steps saved at all the intermediate continuation stages. As shown in the table, a large value of the residual criterion can lead to poor convergence of the non-linear process.

Finally for this Cray-XMP/48 example we show the singular value distribution of two Rayleigh number cases. Figure 3 plots logarithm of singular value versus singular value number — $Ra = 1000$ as solid line and $Ra = 10000$ as a dotted line — computed using LINPACK. Both of these lines exhibit a long 'plateau' where the singular values hardly change, but the $Ra = 10000$ curve has a sharper slope and requires more iterations of BICG. There is a sharp drop in the curve which corresponds to the pressure unknowns. These act as Lagrange multipliers for the incompressibility constraint and can be seen to make the matrix problem a 'difficult' one.

Table V. Effect of changing the BICG norm residual criterion; Δ_2 criterion = 5%; Ra = 1000

	N_2 convergence measure				
		BICG norm residual criterion			
Picard Step	Frontal direct	$3{\cdot}2 \times 10^{-5}$	$1{\cdot}0 \times 10^{-5}$	$1{\cdot}0 \times 10^{-6}$	$1{\cdot}0 \times 10^{-7}$
2	402	435	412	418	402
3	232	270	242	240	232
4	28	62	59	35	28
5	4	33	36	11	4
6	4	29	32	8	4
7	0	36	30	5	0
8		32	24	6	
9		35	28	4	
10		39	27	2	
11		30	30	7	
12		30	18	0	

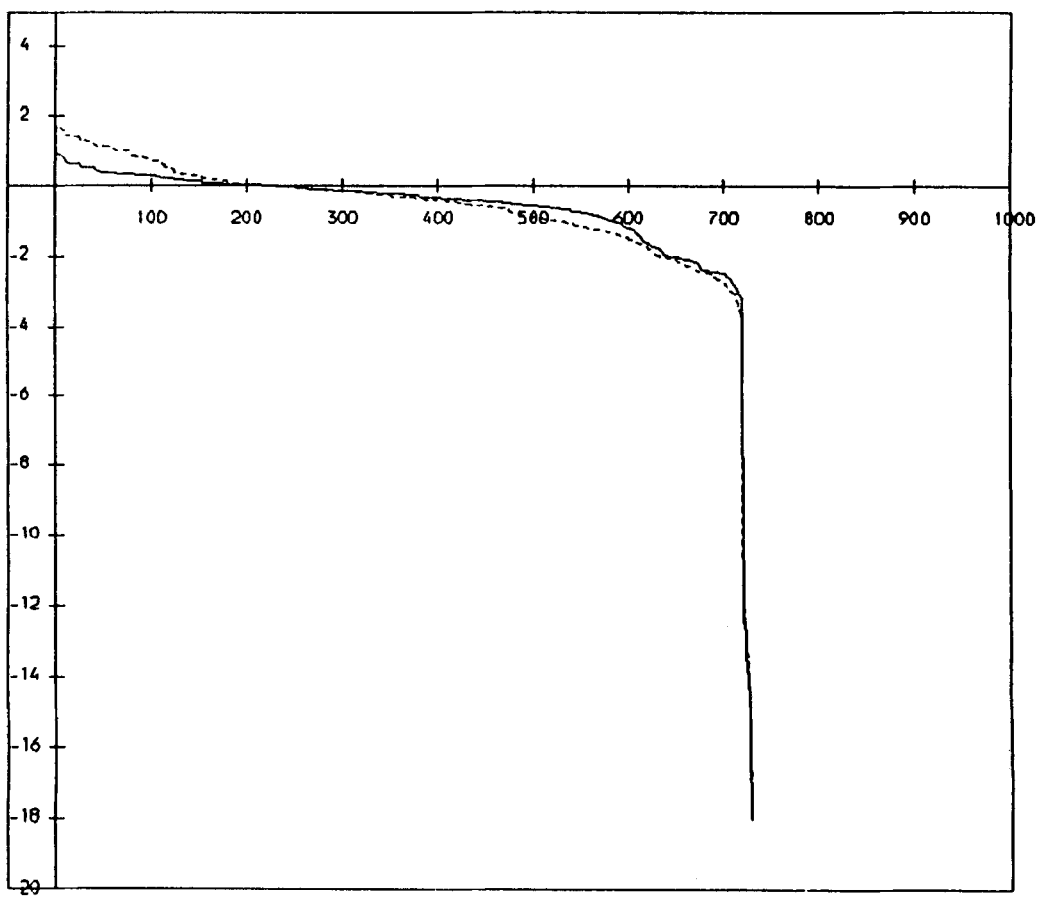


Figure 3. Singular value decomposition for two Rayleigh numbers at two Picard linearization stages in Table IV

ACKNOWLEDGEMENTS

W. M. Connolley wishes to acknowledge support from an SERC/CASE studentship. D. Howard wishes to acknowledge funding from Rolls Royce through a Pembroke College Junior Research Fellowship. We also wish to thank the following for use of their computing facilities: AERE, Harwell (Cray 2) and RAL (Cray XMP/48).

REFERENCES

1. O. Zienkiewicz, *The Finite Element Method*, 3rd edn, McGraw-Hill, New York, 1977.
2. O. Axelsson and V. A. Barker, *Finite Element Solution of Boundary Value Problems*, Academic Press, New York, 1984.
3. A. Peano, 'Hierarchies of conforming finite elements for plane elasticity and plate bending', *Comput. Math. Appl.*, **2**, 211–224 (1976).
4. O. C. Zienkiewicz and A. Craig, 'Adaptive refinement, error estimates, multigrid solution and hierarchic finite element method concepts', Chap. 2 of: Accuracy estimates and adaptive refinements in finite element computations (ARFEC), pp. 25–60, I. Babuška, O. Z. Zienkiewicz, J. Gago, and E. R. de Oliveira (Eds.) John Wiley & Sons, 1986.
5. M. A. Crisfield, 'New solution procedures for linear and non-linear finite element analysis', in J. R. Whiteman (ed.), *MAFELAP 1984*, Academic Press, New York, 1985, pp. 49–81.
6. T. J. R. Hughes, L. P. Fraca and M. Balestra, 'A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška–Brezzi condition: A stable Petrov–Galerkin formulation of the Stokes problem accommodating equal-order interpolations', *Comput. Methods Appl. Mech. Eng.*, **59**, 85–99 (1986).
7. I. Babuška, 'The finite element method with Lagrange multipliers', *Numer. Methods*, **20**, 179–192 (1973).
8. F. Brezzi, 'On the existence, uniqueness, and approximation of saddle point problems arising from Lagrange multipliers', *RAIRO*, **8**, 129–151 (1974).
9. Y. Saad and M. H. Schultz, 'Conjugate gradient-like algorithms solving nonsymmetric linear systems', *Yale Research Report YALEU/DCS/RR-283*, 1983.
10. P. Sonneveld, P. Wesselling and P. M. deZeeuw, 'Multigrid and conjugate gradient methods', in D. J. Paddon and H. Holstein (eds.), *Multigrid Methods for Integral and Differential Equations*, 1985, IMA Conference Series, no. 3, Clarendon Press, Oxford, 1985.
11. Y. S. Wong, 'Iterative methods for problems in numerical analysis', *D.Phil. Thesis*, Numerical Analysis Group, Oxford University Computing Laboratory, 1978.
12. Y. Saad, 'Krylov subspace methods on supercomputers', *Research Institute for Advanced Computer Science, NASA Ames Research Center, Technical Report 88.40*, 1988.
13. S. Sivaloganathan, 'Iterative methods for large sparse systems of equations', *D.Phil. Thesis*, Numerical Analysis Group, Oxford University Computing Laboratory, 1982.
14. D. Benson, personal communication.
15. T. Murdoch, personal communication.
16. P. Hood, 'Frontal solution program for unsymmetric matrices', *Int. j. numer. methods eng.*, **10**, 379–399 (1978).
17. B. M. Irons, 'A frontal solution program', *Int. j. numer. methods eng.*, **2**, 5–32 (1970).
18. R. Fletcher, 'Conjugate gradient methods for indefinite systems', in G. A. Watson (ed.), *Proc. Dundee Conf. on Numerical Analysis; Lecture Notes in Mathematics, Vol. 506*, Springer, Berlin, pp. 73–89, 1976.
19. I. Duff, 'MA32-A package for solving sparse unsymmetric matrices using the frontal method', *Harwell Report AERE R-10079*, HMSO, London, 1981.
20. M. M. Enayet, M. M. Gibson, A. M. K. P. Taylor and M. Yianneskis, 'Laser doppler measurements of laminar and turbulent flow in a pipe bend', *NASA Contract Report 3551, Contract NASW-3258*, May 1982.
21. D. Howard, 'Numerical techniques for simulation of three dimensional swirling flow', *Ph.D. Thesis*, Department of Civil Engineering, Univerisity College of Swansea, 1988.
22. D. Howard, 'Finite element computation of flows through bends', *European Research Community on Flow and Combustion (ERCOFTAC) Bulletin, No. III*, July 1989.
23. W. M. Connolley, 'Preconditioning of iterative methods for linearised or linear systems', *D.Phil. Thesis*, Numerical Analysis Group, Oxford University Computing Laboratory, in preparation.