PRECONDITIONED CONJUGATE GRADIENT METHODS FOR THE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS*

P. CHIN†, E. F. D'AZEVEDO‡, P. A. FORSYTH† AND W.-P. TANG†

* This work was supported by the Natural Sciences and Engineering Research Council of Canada, by the Information Technology Research Centre, which is funded by the Province of Ontario, and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc., through an appointment to the U.S. Department of Energy Postgraduate Research Program administered by Oak Ridge Associated Universities.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1.

‡ Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, Tennessee.

Abstract. A robust technique for solving primitive-variable formulations of the incompressible Navier-Stokes equations is to use Newton iteration for the fully-implicit nonlinear equations. A direct sparse matrix method can be used to solve the Jacobian but is costly for large problems; an alternative is to use an iterative matrix method. This paper investigates effective ways of using a conjugate gradient type method with an incomplete LU factorization preconditioner for two-dimensional incompressible viscous flow problems. Special attention is paid to the ordering of unknowns, with emphasis on a minimum updating matrix (MUM) ordering. Numerical results are given for several test problems.

Key Words. Navier-Stokes, ILU(ℓ), preconditioned conjugate gradient

AMS(MOS) subject classification. 65F10, 76D05

Running Title. PCG methods for N-S equations

1. Introduction. Primitive-variable formulations of incompressible Navier-Stokes fluid-flow problems are notoriously difficult to solve numerically. A robust technique is to use Newton iteration for the fully-coupled nonlinear equations [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. This technique is commonly used at present since it requires few (if any) iteration parameters and converges more quickly than segregated approaches such as SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) [11, 12]. Steady-state problems can be solved by integrating the time-dependent equations to a very large time. When Newton iteration is used to solve the fully-implicit equations, large timesteps can be taken, so that only a small number of timesteps is required to reach the steady state. This method is easily applicable to many situations, including free-surface buoyancy-induced flows [6], generalized Newtonian fluids [5], unstructured finite-element type grids [3, 8], and body-fitted coordinate systems [9].

A direct matrix method can be used to solve the Jacobian [1, 2, 4, 6, 7, 8, 9, 10] but this is prohibitively expensive for three-dimensional problems. An alternative is to use an iterative matrix method. Recently, several authors have applied conjugate gradient type methods to fluid flow problems [3, 5, 6, 8, 13]. The conclusions of these articles vary. Some authors find that iterative methods are cost-effective compared to direct solvers, while others conclude that iterative methods are not competitive with direct solution methods.

The situation is complicated by difficulties arising from the construction of the Jacobian. Because the mass conservation equation does not contain any pressure terms, many discretization methods result in zeros on the diagonal of the Jacobian. Thus, care

must be taken in ordering the unknowns to avoid a zero pivot during the factorization of the Jacobian. Several authors have noticed that the ordering of unknowns affects the convergence properties of iterative matrix methods [14, 15, 16]. The poor performance shown by iterative solvers in some studies may be due to lack of attention to the ordering. Convergence may be also be affected by how velocities on cell interfaces are calculated. Discretizations using upstream weighting generally produce more diagonally dominant Jacobians. However, some diagonal dominance may be lost when more complicated weighting schemes are used, and unpredictable behavior may occur in such cases. The objective of this paper is to determine effective ways of using preconditioned conjugate gradient (PCG) type methods with full Newton iteration for incompressible viscous flow. An incomplete LU (ILU) factorization will be used as a preconditioner. Several ordering strategies, including a minimum updating matrix (MUM) [16] ordering, will be examined. Also, various methods for discretizing the convection flux and their effect on the performance of the iterative solver will be considered. The acceleration techniques used in this work will be restricted to CGS [17] and CGSTAB [18]; no attempt is made to carry out an exhaustive comparison of different acceleration schemes.

As model problems, we consider the incompressible Navier-Stokes equations on a selection of two-dimensional regions. A standard finite-volume discretization defined on a staggered grid is used [11]. We believe that the conclusions we make are essentially not affected by the basic discretization method (*i.e.* finite-volume or finite-element). In another article, a few tests on some Jacobians generated from finite-element discretizations are reported [16]. These results are qualitatively similar to the results reported

here.

In practice, a sequence of matrix problems is solved, one for each Newton iteration. Consequently, in the course of solving the nonlinear equations, the iterative solver is tested on a variety of matrices. The initial matrices will have very poor initial guesses, and hence a large number of inner iterations (*i.e.* iterations performed by the matrix solver) is generally required. In contrast, in the later stages of the Newton iteration, a very good initial guess is available, and few inner iterations are required to meet the convergence tolerance. We have found that examining the behavior of the iterative methods for a single Jacobian can be misleading. For example, some orderings produce good convergence for the first few Jacobian solves but then fail repeatedly for the fourth or fifth Newton iteration. Consequently, results will be reported in terms of total CPU time and total inner iterations for an entire sequence of Newton iterations.

2. Governing equations and solution strategy. The laws governing two-dimensional incompressible fluid flow are the conservation of momentum (Navier-Stokes) equations,

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(uu) + \frac{\partial}{\partial y}(vu) + \frac{\partial p}{\partial x} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = 0, \tag{1}$$

$$\frac{\partial v}{\partial t} + \frac{\partial}{\partial x}(uv) + \frac{\partial}{\partial y}(vv) + \frac{\partial p}{\partial y} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) = 0, \tag{2}$$

and the conservation of mass equation,

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{3}$$

Here, u and v are the velocities in the x- and y- directions, and p is the pressure. Equations (1), (2) and (3) are in nondimensionalized form, with a single parameter, the Reynolds number, Re.

The steady-state solution to the conservation equations is obtained by using a timestepping method. The equations are discretized using a finite-volume approach as described in [11]. Pressures and velocities are given at an initial time. During each timestep, Newton's method is used to solve the nonlinear discretized equations. An initial guess for the unknowns at each step is taken to be the values of the unknowns at the previous step.

2.1. Discretization. The equations are discretized over a staggered grid (Figure 1) [11]. The region is divided into rectangular cells; pressure unknowns are placed in the centers of the cells, and velocity unknowns are placed at the faces. The velocity is given at the boundaries of the region. Since the system is incompressible, one of the pressure unknowns is set to an arbitrary value. The mass conservation equation is integrated over each cell (Figure 2a) with dimensions $\Delta x \times \Delta y$ to give

$$(u_{i+1,j} - u_{i,j})\Delta y + (v_{i,j+1} - v_{i,j})\Delta x = 0. \tag{4}$$

The Navier-Stokes equations, (1) and (2), are integrated over "staggered" cells containing u and v at their centers (Figures 2b and 2c). Using the notation of [11], the two equations can be written more generally as

$$\frac{\partial \phi}{\partial t} + \frac{\partial J_x}{\partial x} + \frac{\partial J_y}{\partial y} = S, \tag{5}$$

where

$$J_x = u\phi - \frac{1}{\mathrm{Re}}\frac{\partial \phi}{\partial x}, \tag{6}$$

$$J_y = v\phi - \frac{1}{\mathrm{Re}}\frac{\partial \phi}{\partial y}. \tag{7}$$

The variable ϕ represents u or v, and S represents the source term (in this case, the pressure differential). Integrating Equation (5) over a cell containing ϕ at its center, with dimensions $\Delta x \times \Delta y$, gives

$$\frac{(\phi_{i,j}^{n+1} - \phi_{i,j}^{n})}{\Delta t}\Delta x \Delta y + (J_{i+1/2,j} - J_{i-1/2,j})\Delta y + (J_{i,j+1/2} - J_{i,j-1/2})\Delta x = S^* \tag{8}$$

where

$$S^* = (p_{i,j} - p_{i-1,j})\Delta y$$

when $\phi = u$, and

$$S^* = (p_{i,j} - p_{i,j-1})\Delta x$$

when $\phi = v$. $J_{i+1/2,j}$ and $J_{i-1/2,j}$ represent the values of J_x at the right and left interfaces of the cell. $J_{i,j+1/2}$ and $J_{i,j-1/2}$ represent values of J_y at the top and bottom interfaces. The equations are fully implicit; that is, all variables, except for the $\phi_{i,j}^n$ in the time derivative term, are solved at the new time ($\phi_{i,j} \equiv \phi_{i,j}^{n+1}$).

2.2. Weighting techniques. Many methods are available to discretize the convection-diffusion fluxes J_x and J_y. For example, if we wish to find the value of J_x at a cell interface between two grid points $\phi_{i,j}$ and $\phi_{i+1,j}$, we can write $J_{i+1/2,j}$ as follows:

$$J_{i+1/2,j} = (k + \max(-\mathrm{Re}_c, 0))\frac{\phi_{i,j} - \phi_{i+1,j}}{\mathrm{Re}\ h} + u\phi_{i,j} \tag{9}$$

where h is the distance between the two grid points, u is the average velocity through the cell and $\mathrm{Re}_c = \mathrm{Re}\ uh$ is the cell Reynolds number. Different values of k can be chosen to implement various weighting schemes (see [11] for a complete discussion of this topic).

For example, $k = 1 - 0.5|\mathrm{Re}_c|$ corresponds to central weighting (when the interface is midway between the grid points) and $k = 1$ corresponds to upstream weighting. It is well-known that for large cell Reynolds numbers, nonphysical oscillatory solutions are obtained with central weighting. Upstream weighting tends to be overly diffusive. A compromise between these two techniques is the hybrid scheme ($k = \max[0, 1-0.5|\mathrm{Re}_c|]$) which is essentially central weighting for $|\mathrm{Re}_c| \leq 2$ and upstream weighting, with the diffusion term $\frac{1}{\mathrm{Re}}\frac{\partial \phi}{\partial x}$ omitted from J_x, for $|\mathrm{Re}_c| > 2$.

Another alternative is the exponential scheme. This scheme bases the calculation of J_x upon the solution to the local one-dimensional convection-diffusion problem,

$$\frac{dJ_x}{dx} = 0, \tag{10}$$

between the two adjacent grid points. The exact solution is given by:

$$\phi(x) = \phi_{i,j} + (\phi_{i+1,j} - \phi_{i,j})\frac{exp(\mathrm{Re}_c\ x/h) - 1}{exp(\mathrm{Re}_c) - 1}. \tag{11}$$

This leads to

$$k = \frac{|\mathrm{Re}_c|}{\exp(|\mathrm{Re}_c|) - 1} \tag{12}$$

in equation (9). As equation (12) shows, the exponential scheme involves expensive computation of exponentials, and therefore, it is not widely-used. Power-law weighting is a popular approximation to the exponential scheme that is less expensive to compute. With power-law weighting,

$$k = \max[0, (1 - 0.1|\mathrm{Re}_c|)^5]. \tag{13}$$

More complex weighting schemes such as QUICK [19] can also be used. Because we wish to restrict our discussion to commonly-used easy-to-implement weighting schemes, most of our tests are carried out with power-law weighting. Some tests are performed with upstream and hybrid weighting as well.

Power-law weighting has been used successfully in the past since it produces frozen coefficient equations with positive coefficients [11]. However, when the Jacobian is constructed for Newton iteration, derivatives of k (equation (13)) appear whenever $|\mathrm{Re}_c| \leq 10$. These entries may cause the off-diagonal entries in the Jacobian to change sign and may also reduce diagonal dominance, thus causing difficulties with convergence of the iterative matrix method. This problem is aggravated on fine grids with small cell Reynolds numbers, but can be avoided by using upstream weighting ($k = 1$) since full upstream weighting does not result in any derivatives of k appearing in the Jacobian. If using power-law weighting causes convergence problems, we can still obtain the power-law solution in one of several ways:

1. (PLR – Power-Law Right-hand-side) Upstream weighting is used for the Jacobian and power-law weighting is used for the right-hand-side vector throughout the solution process.
2. (PLL – Power-Law on Last step) Upstream weighting is used for both the Jacobian and right-hand-side vector on all timesteps except the last, on which power-law weighting is used for the Jacobian and right-hand-side vector.
3. (PLRL – Power-Law Right-hand-side on Last step) Same as method 2, except upstream weighting is used for the Jacobian on *all* the timesteps, including the

last.

We would expect that using a Jacobian and a right-hand-side vector created from different discretizations would result in slower convergence of Newton's method. This suggests that method 2 (PLL) is the best choice since, for every matrix equation, the Jacobian and right-hand-side vector are formed from the same discretization. This method is essentially finding an upstream solution and then using it as a good initial guess for computing the power-law solution. Since a good initial guess is available, the Jacobian formed with power-law weighting on the last timestep should be relatively easy to solve. However, the two other methods (PLR and PLRL) are probably more robust since they use an upstream Jacobian throughout the solution procedure. These observations are supported by the numerical experiments reported in Section 6.

Although some of some of the above methods use a different weighting technique for the Jacobian and right hand side, all methods will converge to the same solution (to within convergence tolerance) to the nonlinear discrete algebraic equations, after the last timestep.

In other words, for a given weighting method used to construct the right hand side, the same algebraic solution to the steady state problem is obtained, but different techniques are used to obtain the solution. Thus, PLR, PLL, PLRL all converge to the same power law solution. On the other hand, use of upstream weighting for the right hand side on the last timestep, will, of course, yield the solution to the upstream weighted discrete equations, which is different from the solution obtained using power law weighting (for a finite grid size).

Of course, there is much debate concerning the relative merits of upstream, hybrid, and power law weighting schemes, in terms of accuracy of the solution (i.e. comparison of the solution of the discrete equations to the true solution of the partial differential equations). It is not the purpose of this paper to add to this debate. For a given discretization method, and grid size, we are concerned with obtaining the solution to the discrete equations as efficiently as possible.

Although the methods above describe how to obtain the power-law solution, they can be used to efficiently obtain the hybrid and exponential solutions as well.

3. Ordering. The performance of the PCG method is greatly affected by the order of the unknowns [14, 15, 16]. Usually, the alignment of unknowns and equations is such that the k-th equation is the one obtained by integrating over a cell containing the k-th unknown at its center:

unknown	corresponding equation
$p_{i,j}$	mass conservation equation about cell (i,j)
$u_{i,j}$	u-momentum equation about $u_{i,j}$
$v_{i,j}$	v-momentum equation about $v_{i,j}$

Because the discrete mass conservation equation (4) does not contain any pressure unknowns, this alignment can lead to zeros on the diagonal of the Jacobian matrix. With direct matrix solvers, pivoting can be used to prevent a division by zero during Gaussian elimination, but it is time-consuming to continually update the data structures. Alternatives are to realign the equations so that no zeros appear on the diagonal [2] or to add nonzeros to the diagonal using a penalty function [4]. However, with the iterative solver used in our experiments, zeros on the diagonal cause no problem as long

as all diagonal locations are filled in with nonzeros before they are used as pivots. A pressure unknown will appear in the discrete momentum equation (5) corresponding to any of the velocity unknowns surrounding it (Figure 2a). If any of these velocity unknowns are incompletely eliminated before the pressure unknown, then fill-in from the momentum equation will produce a nonzero diagonal element in the row of the matrix corresponding to the mass conservation equation.

3.1. Natural and pressure-last orderings. With natural (or "*uvp*") ordering, the grid is traversed left to right, bottom to top. At each cell (Figure 2a), the u component at the left face is ordered first, the v component at the bottom face is ordered next, and the pressure is ordered last. If either face is on a boundary, the velocity at the face is known and consequently is ignored in the ordering process. For any interior cell (*i.e.* one that has no faces on a boundary), the *uvp* ordering guarantees that a neighboring velocity unknown will be eliminated before the pressure unknown is eliminated. However, problems arise when both the bottom face and the left face of a cell land on boundaries (Figure 3a). If the velocities at the other faces have not been eliminated, then the matrix solver will fail because of a zero pivot in the incomplete factorization. If there is only one such cell, we can resolve this problem by removing the pressure unknown in it. (Recall that one of the pressure unknowns must be set to an arbitrary value.) However, if the boundaries do not form a simple rectangle then there may be more than one such cell (Figure 3b).

An ordering technique that does not suffer from this problem is the p-last ordering. With this technique, all the pressure unknowns are ordered last. Thus, all the velocities

are eliminated before any of the pressures are eliminated, and the diagonal elements corresponding to the continuity equations are filled in. In fact, once the velocities have been eliminated, the equations that remain are similar to the pressure equations obtained by using a segregated solution technique such as SIMPLE.

It is possible to combine natural ordering with the robustness of the p-last ordering. The *uvp* ordering is used with one modification. Whenever cells with bottom and left faces on boundaries are encountered, the pressure unknowns for (only) those cells are ordered last. We call this ordering technique the *uvp** ordering.

3.2. Minimum updating matrix ordering. The following sections give a brief description of minimum discarded fill and minimum updating matrix orderings applied to ILU preconditionings. More details are given in [15, 16].

3.2.1. ILU(ℓ) factorization. Assume after k steps of an LU factorization process of a sparse matrix A , we have the following decomposition:

$$A = \begin{bmatrix} L_k & 0 \\ R_k & I_{n-k} \end{bmatrix} \begin{bmatrix} D_k & 0 \\ 0 & A_k \end{bmatrix} \begin{bmatrix} U_k & Q_k \\ 0 & I_{n-k} \end{bmatrix}, \tag{14}$$

where L_k and U_k are $k \times k$ lower and upper triangular respectively with unit diagonal, D_k is $k \times k$ diagonal, R_k is $(n-k) \times k$, Q_k is $k \times (n-k)$, I_{n-k} is the $(n-k) \times (n-k)$ identity, and A_k is the $(n-k) \times (n-k)$ submatrix that remains to be factored. We assume in the following that A need not be symmetric but has a symmetric incidence matrix.

Some new nonzero fill-in entries in A_k might be created in the factorization process. We apply the concept of "fill level" to these nonzero entries to characterize how the fill-

in is introduced. All original nonzero entries in A have fill level zero. Fill-in created by eliminating the first node has fill level one. The level of any new fill-in can be determined by the level of the matrix entry which creates this fill-in. More precisely, let $a_{ij}^{(k)}$ be a nonzero element of matrix A_k. Initially, we have

$$\text{level}_{ij}^{(0)} = \begin{cases} 0 & \text{if } a_{ij} \neq 0, \\ \infty & \text{otherwise.} \end{cases} \tag{15}$$

At each step of the elimination process, the fill levels are modified as

$$\text{level}_{ij}^{(k)} := \min\left(\text{level}_{ik}^{(k-1)} + \text{level}_{kj}^{(k-1)} + 1,\ \text{level}_{ij}^{(k-1)}\right). \tag{16}$$

In an ILU(ℓ) factorization, only fill-in entries with fill level less than $\ell+1$ are kept, *i.e.* $\text{level}_{ij}^{(k)} \leq \ell$, $k = 1 \ldots n-1$. All other fill entries are discarded.

Now consider the k-th step in an ILU(ℓ) factorization of A. If $a_{kk}^{(k-1)}$ is chosen for the next pivot element, then

$$A_{k-1} = \begin{bmatrix} a_{kk}^{(k-1)} & \beta_k^t \\ \alpha_k & B_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \alpha_k/a_{kk}^{(k-1)} & I_{n-k} \end{bmatrix} \begin{bmatrix} a_{kk}^{(k-1)} & 0 \\ 0 & E_k \end{bmatrix} \begin{bmatrix} 1 & \beta_k^t/a_{kk}^{(k-1)} \\ 0 & I_{n-k} \end{bmatrix} \tag{17}$$

where

$$E_k = B_k - \alpha_k \beta_k^t / a_{kk}^{(k-1)}. \tag{18}$$

Since the fill level of each element b_{ij} in B_k is known and $\text{level}_{ij}^{(k-1)} \leq \ell$, using (15) and (16), the fill level of each element e_{ij} in E_k can be determined. In the incomplete factorization of matrix A, some of the entries in the factor are discarded to prevent excessive fill and computation. Let matrix F_k contain the discarded entries. Some fill

entries with level $> \ell$ may have been created in E_k after the k-th step of the factorization. The factorization proceeds with a perturbed A_k

$$A_k = B_k - M_k - F_k, \quad M_k = \alpha_k {\beta_k}^t / a_{kk}^{(k-1)} = \left[m_{ij}^{(k)}\right], \tag{19}$$

where

$$F_k = \left[f_{ij}^{(k)}\right], \quad f_{ij}^{(k)} = \begin{cases} 0 & \text{if } b_{ij}^{(k)} \neq 0 \\ -m_{ij}^{(k)} & \text{if level}_{ij}^{(k)} > \ell \\ 0 & \text{otherwise} \end{cases} . \tag{20}$$

3.2.2. Minimum updating matrix ordering. We present a graph model [20, 21] for describing the factorization process as a series of node eliminations. The graph model is invaluable in providing an insight into the minimum discarded fill ordering.

To simplify notation, we assume the elimination sequence is $v_1, v_2, \ldots, v_n$. Let graph $G_k = (\mathcal{V}_k, \mathcal{E}_k)$, $k = 0, 1, \ldots, n-1$ be the graph corresponding to matrix $A_k = \left[a_{ij}^{(k)}\right]$ of (19). The vertex set and edge set are defined as

$$\mathcal{V}_k = \{v_{k+1}, v_{k+2}, \ldots, v_n\}, \quad \mathcal{E}_k = \left\{(v_i, v_j) \mid a_{ij}^{(k)} \neq 0\right\} . \tag{21}$$

We assume each vertex has a self-loop edge (v_i, v_i) and each edge (v_i, v_j) has a value of $a_{ij}^{(k)}$.

A minimum discarded fill (MDF) ordering for ILU(ℓ) decomposition is to choose the next pivot node v_i such that the size of the discarded fill (*i.e.* the sum of the squares of the discarded fill entries) in E_k is minimized. It has been shown that MDF ordering is effective for problems having a small molecule [16]. However MDF ordering is very expensive for a sparse matrix with a large molecule. Jacobians formed from

the Navier-Stokes equations have this characteristic (*i.e.* a large number of nonzeros in each row). MDF ordering requires a search through the nested linked lists of the sparse matrix A_k to identify the fills and their levels in order to determine the discarded fill in M_k. This kind of searching is costly. Instead, a cheaper heuristic, minimum updating matrix (MUM) ordering, is used for the Navier-Stokes application. The basic idea of this ordering is to choose the next pivot node v_i such that the Frobenius norm (the sum of the squares of the matrix entries) of the matrix $\widetilde{M}_k$ is minimized, where the matrix $\widetilde{M}_k$ is defined as M_k without the diagonal entries. The motivation is simple: if $\|\widetilde{M}_k\|_{\mathcal{F}}$ is small, the discarded fill entries in $\widetilde{M}_k$ must be small, since the discarded fill entries are a subset of the entries in $\widetilde{M}_k$. This approach avoids searching nested sparse linked lists in an MDF -type ordering and has proven to be a cost-effective ordering strategy for sparse matrices with large molecules[16]. The computation of the norm of update matrix is

$$(22)\quad updatenrm(v_k) = \|\widetilde{M}_k\|_F, \quad \widetilde{m}_{ij}^{(k)} = \begin{cases} 0 & \text{if } i = j \\ m_{ij} & \text{otherwise} \end{cases}, \quad M_k = \alpha_k \beta_k^t / a_{kk}^{(k-1)}.$$

See Algorithm 1 for the description of the MUM ordering algorithm:

Note that this definition of MUM ordering always uses the same level for the ILU update matrix $\widetilde{M}_k$ in the ordering algorithm as is used in the ILU factors for the PCG iteration. This does not actually have to be the case, but we will not pursue this distinction in the current work.

The solution of the Navier-Stokes equations requires the solution of a series of matrix problems. It is natural to perform a MUM ordering at the beginning of the

Initialization:

$A_0 := A$
for each $a_{ij} \neq 0$
$\text{level}_{ij}^{(0)} := 0$
end
for each node v_i
Compute the norm of the updata matrix $updatenrm(v_i)$ (see Algorithm 2).
end

for $\mathbf{k} = 1 \ldots n-1$

1.Choose as the next pivot node v_m in matrix A_{k-1} which has minimum $updatenrm(v_m)$ (break ties by choosing earlier node).

2.Update the decomposition,

$$A_k := B_k - M_k - F_k, \quad M_k = \alpha_k \beta_k^t / a_{kk}^{(k-1)}$$

where $F_k = \left[f_{ij}^{(k)} \right]$ is given by (20) and P_k is permutation matrix to exchange v_m to first position,

$$P_k A_{k-1} P_k^t = \begin{bmatrix} a_{kk}^{(k-1)} & \beta_k^t \\ \alpha_k & B_k \end{bmatrix}.$$

3.Update the fill-level of elements in A_k by (15).

for each v_i a neighbor of v_m, $(v_i, v_m) \in \mathcal{E}_{k-1}$
for each v_j a neighbor of v_m, $(v_m, v_j) \in \mathcal{E}_{k-1}$

$$\text{level}_{ij}^{(k)} := \min\left(\text{level}_{im}^{(k-1)} + \text{level}_{mj}^{(k-1)} + 1,\ \text{level}_{ij}^{(k-1)} \right)$$

end
end

4.Update the norms of update matrices of v_m's neighbors.

for each v_i a neighbor of v_m in $\mathcal{E}_{k-1}$

$$updatenrm(v_i) = \|\widetilde{M}_{k+1}\|_F, \quad P_{k+1} A_k P_{k+1}^t = \begin{bmatrix} d_{k+1} & \beta_{k+1}^t \\ \alpha_{k+1} & B_{k+1} \end{bmatrix}$$

where F_{k+1} is given by (20), $M_{k+1} = \alpha_{k+1} \beta_{k+1}^t / a_{k+1,k+1}^{(k)}$ and P_{k+1} is permutation matrix to exchange v_i to first position (see Algorithm 2).

end

end

Algorithm 1
Description of MUM algorithm.

Procedure update_nrm (..)
updatenrm$(v_i) := 0$
for each neighbor v_j of v_i in $\mathcal{E}_k$
for each p such that $\{\ a_{i,p}^{(k)} \neq 0,\ p \neq j\ \}$

$$updatenrm(v_i) = updatenrm(v_i) + \left(\frac{a_{i,p}^{(k)} a_{j,i}^{(k)}}{a_{i,i}^{(k)}} \right)^2$$

end
end
end

Algorithm 2
Description of procedure for calculating the norm of update matrix.

computation, based on the first Jacobian. However, the first Jacobian may not be characteristic of all the Jacobians arising during the solution of the equations. For example, if the initial velocity unknowns were set to zero, then the flow configuration at the start would resemble a Stokes flow, which is a Navier-Stokes flow without the convection (nonlinear) terms. The flow may not take on the characteristics of a Navier-Stokes flow until after the first few timesteps. Hence, the initial MUM ordering may not be the best ordering to use in the later stages of the solution process. Because the MUM ordering time is very small compared to the total solution time, it is often advantageous to find a new MUM ordering midway through the solution process.

4. Computational Details.

4.1. Timestepping strategy. The discretized equations are integrated to steady state, a dimensionless time of 10^4. Some tests were also carried out using a dimensionless stopping time of 10^5, for the Re = 1000 problems. The final results for (u, v, p) were unchanged to three significant figures compared to the runs with a stopping time of 10^4. Initially, velocities and pressures are set to zero, and a small timestep (10^{-1}) is chosen.

If the solution at each time is obtained successfully, the timestep is multiplied by 10, so steady state is reached after only six steps. At each step, the nonlinear equations are solved to convergence. We use an absolute tolerance of 10^{-3}; *i.e.* the solution is computed until there are no more changes in the first 3 decimal places. (All variables are scaled to be $O(1)$). If the changes in velocity after any timestep produce nonphysical results (*i.e.* velocities that exceed the maximum velocity on the boundary), then the changes are ignored and the step is repeated with a smaller increase in time. The step is also repeated if a maximum number of Newton iterations (20 for the tests described below) is surpassed.

A scheme that dynamically determines the size of timesteps may be more efficient. It is also probable that it is not necessary to solve the Newton iteration to convergence during the intermediate pseudo-timesteps. However, for the purpose of this paper, it was preferable to use a fixed series of times and to solve to convergence at each time to ensure a fair comparison of the various strategies studied.

4.2. Linear strategy. Each iteration of Newton's method requires a solution of the Jacobian. We have found that ILU(2) is an effective preconditioner for Jacobians arising from the Navier-Stokes equations, so this is used for all the tests described below. The linear equations are solved to an absolute tolerance of 10^{-6}. The maximum number of inner iterations allowed for each linear problem is 300. If convergence is not reached after the maximum number of inner iterations, the solution obtained at that point is returned regardless and used in the next iteration of Newton's method.

In the results section below, experiments with other aspects of the linear strategy

are described. The effects of three ordering strategies (uvp*, p-last and MUM) and two acceleration methods (CGS and CGSTAB) are compared.

For completeness, the precise CGSTAB algorithm is given below (the incomplete factorization of A is denoted by LU). Right preconditioning is used.

$$
\begin{aligned}
r_0 &= b - Ax_0; \\
\hat{\omega}_0 &= 1; \quad \beta = 1; \quad \alpha_0 = 1; \quad \overline{Aq}_0 = q_0 = 0; \\
\textbf{For} \quad & i = 1, 2, 3, ... \\
& \hat{\beta} = (r_0, r_{i-1}); \omega_i = (\hat{\beta}/\beta)(\hat{\omega}_{i-1}/\alpha_{i-1}); \\
& q_i = r_{i-1} + \omega_i(q_i - \alpha_{i-1}\overline{Aq}_{i-1}) \\
& \bar{q}_i = (LU)^{-1} q_i \\
& \overline{Aq}_i = A\bar{q}_i \\
& \hat{\omega}_i = \hat{\beta}/(r_0, \overline{Aq}_i) \\
& s = r_{i-1} - \hat{\omega}_i \overline{Aq}_i \\
& \bar{s} = (LU)^{-1} s \\
& t = A\bar{s} \\
& \alpha_i = (t, s)/(t, t) \\
& x_i = x_{i-1} + \hat{\omega}_i \bar{q}_i + \alpha_i \bar{s} \\
& \text{if converged, then quit} \\
& r_i = s - \alpha_i t \\
\textbf{End}
\end{aligned}
$$

Thus, the total number of floating-point multiplications needed for one CGSTAB iteration is

$$2NZP + 2NZ + 10N$$

where N is the number of unknowns, NZP is the number of nonzeros in the incomplete factorization and NZ is the number of nonzeros in the original matrix.

The MUM ordering is performed at the beginning of the computation; it is based on the Jacobian created on the first Newton iteration of the first timestep.

5. Test Problems. The performance of the computational procedure was tested using several problems. Results were obtained with Reynolds numbers of 100, 500, and 1000.

5.1. Problem 1: "cavity". The first problem is confined flow in a driven cavity (Figure 4). This is a common test problem which has been solved on grids as large as 320×320 [22]. The region is a unit square. The velocity is zero on all the boundaries except the top boundary, where the horizontal velocity is equal to 1. An equally-spaced grid is used, with $\Delta x = \Delta y$. This problem was solved on 20×20, 40×40 and 80×80 grids.

5.2. Problem 2: "NU cavity". The second problem is the driven cavity problem on a non-uniform grid. On an $n_x \times n_y$ grid, for each $j = 1$ to n_y,

$$\Delta x_{i,j} = \begin{cases} 0.1/n_{x1} & \text{for } i = 1 \text{ to } n_{x1} \\ 0.8/(n_{x2} - n_{x1}) & \text{for } i = n_{x1} + 1 \text{ to } n_{x2} \\ 0.1/(n_x - n_{x2}) & \text{for } i = n_{x2} + 1 \text{ to } n_x, \end{cases}$$

where $n_{x1} = n_x/3$ and $n_{x2} = 2n_x/3$. For each $i = 1$ to n_x,

$$\Delta y_{i,j} = \begin{cases} 0.1/n_{y1} & \text{for } j = 1 \text{ to } n_{y1} \\ 0.8/(n_{y2} - n_{y1}) & \text{for } j = n_{y1} + 1 \text{ to } n_{y2} \\ 0.1/(n_y - n_{y2}) & \text{for } j = n_{y2} + 1 \text{ to } n_y, \end{cases}$$

where $n_{y1} = n_y/3$ and $n_{y2} = 2n_y/3$. This problem was also solved on 20×20, 40×40 and 80×80 grids.

5.3. Problem 3: "step". The third problem is sudden-expansion flow in a channel, commonly called the "backward-facing step problem" (Figure 5). The dimensions of the region are taken from [23]. Fluid enters from the left, passes over a step and leaves at the right. At the left and right boundaries of the region, the velocity has no vertical component but has a parabolic profile for the horizontal component. There are no-slip conditions at the other boundaries. At the left boundary, the maximum horizontal velocity is equal to 1. At the right boundary, the velocities are set so that the amount of flow into the region is equal to the amount of flow out of the region. Recent work on the step problem has shown that, for certain geometries, the flow may not be completely developed at the end of the channel [24]. Therefore, it may be unrealistic to impose a parabolic outflow condition. Because we are primarily interested in solution techniques rather than the solutions themselves, the velocity profile at the end of the channel is not a great concern. For notational convenience, we will refer to the grid sizes used for this problem as 20×20, 40×40 and 80×80. It should be understood that these numbers refer to the smallest rectangle which overlays the problem domain of Figure 5. This grid has a constant spacing in the x-direction and a constant spacing

in the y-direction, with $\Delta x \neq \Delta y$. Some of the cells in this rectangular grid fall outside the problem domain (*i.e.* those in the lower left-hand corner of Figure 5) and hence are not included in the computation.

5.4. Problem 4: "channel". The fourth problem is the "convoluted channel problem" (Figure 6). Flow enters from the left with a maximum velocity of 1 and leaves at the right. The horizontal components of the velocities have a parabolic profile at these boundaries, and the velocity is zero at all other boundaries. At the right boundary, the velocities are set so that the amount of flow into the region is equal to the amount of flow out of the region. The parabolic outflow condition may be unrealistic for such a short channel. As stated above, this is not a concern as we are simply treating this problem as a test case for various solution strategies. As with problem 3, we refer to the various grids used for this problem as 20×20, 40×40 and 80×80. This refers to the smallest rectangle which contains the region of Figure 6. Constant spacing is used in the x- and y- directions. Cells which fall outside the region of Figure 6 are not included in the computation.

Table 1 shows the number of unknowns corresponding to the different grid sizes for the four test problems.

6. Results and discussion. Solutions to the test problems described above were computed using various strategies. The results given in this section are in terms of total inner iterations (for all the Newton iterations) and total execution time. In the tables, normalized execution times are provided to facilitate comparison. A different

normalization factor is used for each distinct test problem. The normalized total CPU time includes Jacobian construction, matrix solution and reordering costs (where applicable). Because the normalized CPU time includes time for system tasks, the execution times listed below may be in error by as much as 5%.

For some of the tests, the number of Newton iterations and the average number of inner iterations per Newton iteration are also given. When solving a particular problem at a given Reynolds number and grid size, we would expect a constant number of Newton iterations, regardless of the techniques (*e.g.* ordering, acceleration) used for the linear problems. In practice, when a test problem is solved with two different techniques, one case may require an extra Newton iteration due to small differences caused by round-off error. This extra Newton iteration usually requires only a few inner iterations. Occasionally, the matrix solver may fail to converge to a solution for a particular Newton iteration (*i.e.* the maximum number of inner iterations is reached before the required inner tolerance is achieved). In such a case, the number of total Newton iterations may be greatly affected. In the tables below, two asterisks (**) indicates that at least one such convergence failure occurred.

A computational process was aborted if too many repeat timesteps were required or if a maximum execution time was surpassed. In the tables given below, the entry "failed" indicates that a solution was not obtained for one of these reasons. For these cases, the maximum allowable repeat timesteps was set to three.

6.1. Comparison of ordering strategies. Results for the four problems were obtained with three Reynolds numbers on a 40×40 grid, using the uvp^*, p-last and

MUM orderings. MUM ordering was used in two cases. In the first case (denoted simply by "MUM" in the results), the MUM ordering was performed once at the start. In the second case (denoted by "MUM*"), a second MUM ordering was performed before the third timestep. Power-law differencing and CGSTAB acceleration were used for the tests described in this section.

Table 2 shows the number of Newton iterations required. Convergence problems were encountered in several test cases using *uvp** and *p*-last orderings. No solution was obtained for two cases with *uvp** ordering. Table 3 shows the total execution time and the total inner iterations required. For all the test cases except one, either MUM or MUM* ordering required the fewest inner iterations. MUM and MUM* orderings also showed the lowest execution times for most of the cases. For the few cases where *uvp** was fastest, *uvp** required about 15% to 25% less time than MUM*. However, for the NU cavity and step problems, MUM and MUM* showed significantly superior performance. The *p*-last ordering required, on average, twice as much time as MUM* ordering.

It is clear that the *p*-last ordering is a poor choice. The *uvp** ordering sometimes results in low execution times when it works, but it is prone to convergence failures. Although the MUM ordering does not always show the best performance, it is certainly the most robust ordering, as it did not fail in any of our tests. Thus, the MUM ordering may be a particularly good choice if a very difficult problem needs to be solved, or if the overall solution strategy is not very sophisticated. The MUM ordering can also be used when an *a priori* ordering like *uvp** cannot be easily determined (*e.g.* on a

finite-element mesh). Generally, the MUM ordering produces more fill than the *uvp** ordering. Even though MUM usually requires fewer inner iterations, it occasionally requires more execution time than *uvp** because the solution time for each inner iteration is proportional to the amount of fill. The execution time obtained with the MUM ordering can be significantly reduced if a drop tolerance were used to decrease the amount of fill [16].

Table 3 also shows an interesting trend. For the MUM orderings, the solution times increase monotonically as the Reynolds number increases (for a given problem). This is intuitively reasonable. However, the uvp^* ordering does not show this behaviour. In some cases uvp^* fails for low Reynold's numbers, and succeeds for high Reynolds numbers. Although the uvp^* approach ensures that the diagonal pivots of the incomplete factorization are not identically zero, this ordering may produce a poor preconditioner (some of the pivots could be numerically small). Clearly, the unpredictability of the uvp^* ordering is a disadvantage of this ordering compared to MUM ordering.

In forming the *uvp** and *p*-last orderings, the grid is traversed in an *x*-*y* fashion (*i.e.* *x*-direction first, then *y*-direction). The MUM reordering is also performed on an initial *x*-*y* ordering. It has been shown that a *y*-*x* ordering produces much faster convergence for anisotropic problems with strong coupling in the *x*-direction [15, 25]. The three orderings with *y*-*x* traversal of the grid were tested on the four problems (Re = 1000). In most of the cases, the execution time (Table 4) was not altered drastically when *y*-*x* ordering was used. However, *y*-*x* *uvp** ordering required 70% more time for the NU cavity problem and *y*-*x* *p*-last ordering required 38% more time for the step problem.

These two problems are highly anisotropic. The MUM ordering was the least sensitive to the change, which caused no more than 10% difference in execution time for each of the test problems. This again shows that MUM ordering is a robust technique.

6.2. Comparison of acceleration methods. The test problems (Re = 1000) were solved on 40×40 grids using CGS and CGSTAB acceleration methods. Power-law differencing and MUM ordering were used. For all the test problems, CGSTAB required fewer inner iterations and lower execution times (Table 5). The advantage of CGSTAB shows up even more clearly when the Jacobian is harder to solve. For example, when the step problem was solved with p-last ordering, an answer was obtained with CGSTAB acceleration, while with CGS acceleration, the computation failed to complete because of an excessive number of convergence failures. For any Newton iteration, if the maximum number of inner iterations is reached, then the answer obtained thus far is used for the next Newton iteration. If the residual has been reduced at all by the matrix solver after the maximum number of inner iterations, then the new solution is a better approximation than the previous guess. Because the CGSTAB method shows a more monotone decrease in the residual, the new approximation is usually better than the old approximation. With CGS, the residual is an erratic function of iteration number. Fluctuations in the residual may cause the matrix solver to return a worse approximation after a given number of inner iterations. If the new answer is extremely bad, this may cause further convergence problems.

A few tests with ORTHOMIN [25] acceleration showed that it performed very poorly compared to either CGS or CGSTAB. In most cases, it required at least twice

as much CPU time as CGS and in some cases, it failed to produce a solution. Hence, ORTHOMIN is not considered a viable alternative for these types of test problems.

6.3. Results for different weighting strategies. As anticipated, power-law weighting led to convergence problems with the matrix solver on an 80×80 grid. Thus, an alternate method was required to obtain solutions on fine grids. The three methods for obtaining the power-law solution discussed earlier were tested on the four problems (Re = 1000) on a 40×40 grid. CGSTAB acceleration and MUM ordering were used. Table 6 shows the execution time required by these methods, compared to times required with pure upstream or power-law weighting. Recall that power-law, PLR, PLL, and PLRL all converge to the power law solution, while the column in Table 6 labeled upstream refers to the converged upstream solution, which is observably different from the power law solution at these grid sizes. It can be seen that by using PLL (Power-Law on Last step), the power-law solution can be obtained at little extra cost compared to the cost of obtaining the upstream solution. Although PLRL (Power-Law Right-hand-side on Last step) shows slightly poorer performance than PLL, it is probably more robust, since an upstream Jacobian is used throughout the procedure. The PLRL method was used to obtain the power-law solution for the cavity, step and channel problems on an 80×80 grid at Re = 1000. The streamlines are shown in Figures 7, 8 and 9.

As mentioned above, the three methods PLR, PLL and PLRL all produced values that matched the power-law values to 3 digits on a number of test cases. Using an approximation to the Jacobian at any point in the solution procedure affects only intermediate values but does not affect the accuracy of the final solution. However, the

methods do vary in efficiency, as shown in Table 6. Given a particular discretization (e.g. power-law, hybrid, upstream, etc.) and grid size, all solution methods produce identical values. Naturally, the solutions will vary with *different* discretizations.

In order to further verify that our methods were producing valid solutions, the results were compared to previous work. The cavity problem, with Re = 1000, was solved on an 80×80 grid using a variation of PLRL (upstream Jacobian and right-hand-side throughout except hybrid right-hand-side on the last step) to obtain the hybrid solution. The maximum negative value for u on the vertical centerline was -0.338, which agrees to three figures with the value given in [22] for the identical test problem with hybrid weighting of the flux terms.

6.4. Effect of grid size. It is of interest to obtain an estimate of the effect of grid size on the number of inner iterations required to solve the Jacobian. In order to eliminate the effect of nonlinearities, a Stokes flow driven cavity problem was solved on various grids (with Re = 1000). CGSTAB acceleration and MUM ordering were used. The convergence tolerance for this linear problem was an absolute tolerance of 10^{-6} and an initial guess of $p = u = v = 0$ was used. The number of iterations (15, 28 and 54 for 20×20, 40×40 and 80×80 grids, respectively) approximately doubles as the number of unknowns increases fourfold. This indicates that the number of iterations is roughly proportional to $O(N^{1/2})$ which is what would be expected for a second-order operator. Note that for the special case of Stokes flow, a preconditioner has been developed which results in the number of iterations being independent of grid size [26, 27].

The full Navier-Stokes equations were also solved on 3 different grids. These runs

were carried out using upstream weighting, to avoid the complicating effect of using a combined upstream/power-law approach as discussed above. CGSTAB acceleration and MUM ordering were used. The execution times and inner iterations are shown in Table 7. The number of Newton iterations and the average number of inner iterations per Newton iteration are shown in Table 8. The number of inner iterations per Newton iteration shows an approximate $O(N^{1/2})$ behaviour for the full nonlinear problem. Thus, the total work is about $O(N^{1.5})$. For the 80×80 grids, the MUM ordering time was only about 2% of the total time. As the grid becomes finer, this cost will become insignificant compared to the matrix solution cost.

As expected with the PCG method, the required storage showed an $O(N)$ behaviour. With ILU(2) preconditioning, roughly 0.6 Kbytes of storage was required for each unknown (all variables are double precision). Of course, a direct solver generally requires more storage. For example, in [4], the YSMP code uses $O(N^{1.3})$ storage and requires $O(N^{1.8})$ work.

The overall computation time can usually be improved by first solving the problem on a coarse grid and then extrapolating to obtain an initial solution on the desired grid. This technique is effective in most cases; however, care must be taken when a complicated geometry producing a number of vortices (such as the convoluted channel problem) is being considered. Another possibility is to extrapolate in Reynolds number. Our preliminary tests have shown this to be less advantageous than using coarse grid approximations. However, more testing is required before we can present any conclusion on this subject.

Starting from all-zero initial values, as we have done here, provides a good test of the robustness of the solution strategies. In practice, however, one would normally start with an approximate solution as discussed above. In order to compute these initial approximations, it is still necessary to use a robust technique such as MUM. (Note that *uvp** can fail for low Re problems; see Table 3.) Consequently, using coarse grid or low Re initial approximations would not alter our conclusions.

7. Conclusions. PCG iterative methods can be used for effective solution of Jacobian matrices arising from finite-volume discretizations of the incompressible Navier-Stokes equations. However, the ordering of the unknowns has a large effect on the convergence behavior and on the efficiency of incomplete factorization preconditionings.

In the case of finite-volume discretizations on regular grids, it is possible to deduce several *a priori* orderings which ensure that zero pivots are not encountered during the incomplete factorization. However, a more robust method is the MUM (minimum updating matrix) ordering technique. Numerical tests have shown that MUM ordering results in generally faster convergence compared to the *a priori* orderings. The more efficient of the two *a priori* orderings was *uvp** (see Section 3.1). At best, *uvp** was 15-25% faster (in terms of total CPU cost) compared to MUM ordering. However, *uvp** sometimes required twice the CPU cost of MUM ordering and occasionally failed to converge within the allotted maximum CPU time. MUM ordering succeeded (completed the runs within the CPU time limit) in all cases. MUM was also fairly insensitive to the initial ordering, while the *uvp** ordering was very sensitive to the initial ordering.

Another advantage of MUM ordering is that it takes as input a general sparse matrix and uses no information about the underlying discretization method or the structure of the mesh. Consequently, we expect MUM ordering to be even more effective for finite-element discretizations on unstructured meshes. Some preliminary tests [16] support this conjecture.

The performance of ILU PCG methods is affected by the weighting used in the discretization of the momentum flux terms. Upstream weighting always produces a well-conditioned Jacobian. However, use of power-law or hybrid weighting schemes often caused poor convergence behavior on fine meshes. These methods (weighted average of upstream and downstream) ensure positive coefficients for the frozen coefficient equations. However, when a Jacobian is constructed, the derivative of the weighting factor may cause the off-diagonals of the Jacobian to change sign, and the Jacobian may become less diagonally dominant. Nevertheless, an efficient technique for obtaining the power-law solution is to use upstream weighting for the construction of the Jacobian, while evaluating the residual using the power-law method (on the last timestep only). The same idea can also be used for hybrid weighting. Tests show that this method obtains the hybrid or power-law solution at a cost of 20-40% more CPU time compared to full upstream weighting.

Our tests also indicate that CGSTAB acceleration seems to be more effective than CGS acceleration, especially in the context of a Newton iteration. ORTHOMIN did not appear to be competitive. We have also found that a level 2 ILU seems to be the most efficient in terms of total CPU cost for two-dimensional problems. Finally, the combi-

nation of an ordering method and use of a drop tolerance in the ILU preconditioning is a promising avenue of further work [16].

REFERENCES

[1] R. Schreiber and H. B. Keller. Driven cavity flows by efficient numerical techniques. *J. Comput. Phys.*, 49, 310–333, 1983.

[2] S. P. Vanka. Block-implicit calculation of steady turbulent recirculating flows. *Internat. J. Heat and Mass Transfer*, 28, 2093–2103, 1985.

[3] M. P. Robichaud and P. A. Tanguy. Finite element solution of three-dimensional incompressible fluid flow problems by a preconditioned conjugate residual method. *Internat. J. Numer. Methods Engrg.*, 24, 447–457, 1987.

[4] M. E. Braaten and S. V. Patankar. A block-corrected subdomain solution procedure for recirculating flow calculations. *Numer. Heat Transfer*, 15, 1–20, 1989.

[5] G. F. Carey, K. C. Wang, and W. D. Joubert. Performance of iterative methods for Newtonian and generalized Newtonian flows. *Internat. J. Numer. Methods Fluids*, 9, 127–150, 1989.

[6] D. S. Dandy and L. G. Leal. A Newton's method scheme for solving free-surface flow problems. *Internat. J. Numer. Methods Fluids*, 9, 1469–1486, 1989.

[7] J. W. MacArthur and S. V. Patankar. Robust semidirect finite difference methods for solving the Navier-Stokes and energy equations. *Internat. J. for Numer. Methods Fluids*, 9, 325–340, 1989.

[8] D. Howard, W. M. Connolley, and J. S. Rollett. Unsymmetric conjugate gradient methods and sparse direct methods in finite element flow simulation. *Internat. J. Numer. Methods Fluids*, 10, 925–945, 1990.

[9] K. C. Karki and H. C. Mongia. Evaluation of a coupled solution approach for fluid flow calculations in body-fitted co-ordinates. *Internat. J. Numer. Methods Fluids*, 11, 1–20, 1990.

[10] V. Venkatakrishnan. Viscous computations using a direct solver. *Computers and Fluids*, 18, 191–204, 1990.

[11] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow.* Hemisphere Publishing Corporation, 1980.

[12] S. V. Patankar. A calculation procedure for two-dimensional elliptic situations. *Numer. Heat Transfer*, 4, 409–425, 1981.

[13] D. P. Young, R. G. Melvin, F. T. Johnson, J. E. Bussoletti, L. B. Wigton, and S. S. Samant. Application of sparse matrix solvers as effective preconditioners. *SIAM J. Sci. Statist. Comput.*, 10, 1186–1199, 1989.

[14] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, pages 635–657, 1989.

[15] E. F. D'Azevedo, P. A. Forsyth, and Wei-Pai Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. Submitted to *SIAM J. Mat. Anal. and Appl.*, 1990.

[16] E. F. D'Azevedo, P. A. Forsyth, and Wei-Pai Tang. Towards a cost-effective high order ILU preconditioner. Submitted to *BIT*, 1990.

[17] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric systems. *SIAM J. Sci. Statist. Comput.*, 10, 36–52, 1989.

[18] H. A. Van De Worst and P. Sonneveld. CGSTAB, a more smoothly converging variant of CGS. Technical Report 90-50, Delft University of Technology, Delft, Netherlands, 1990.

[19] B. P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Comput. Methods Appl. Mech. Engrg.*, 19, 59–98, 1979.

[20] S. V. Parter. The use of linear graphs in Gaussian elimination. *SIAM Review*, 3, 364–369, 1961.

[21] D.J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

[22] S. P. Vanka. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *J. Comput. Phys.*, 65, 138–158, 1986.

[23] O. Pironneau. *Finite Element Methods for Fluids.* John Wiley and Sons, 1989.

[24] D. K. Gartling. A test problem for outflow boundary conditions–flow over a backward facing step. *Internat. J. Numer. Methods Fluids*, 11, 953–967, 1990.

[25] A. Behie and P. A. Forsyth. Comparison of fast iterative methods for symmetric systems. *IMA J. Numer. Anal.*, 3, 41–63, 1983.

[26] J. H. Bramble and J. E. Pasciak. A preconditioning technique for indefinite systems resulting from mixed approximations of elliptic problems. *Math. Comput.*, 50, 1–17, 1988.

[27] T. Rusten and R. Winther. A preconditioned iterative method for saddle point problems. *Proceedings of the Copper Mountain Conference on Iterative Methods*, 1990.