INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 9, 213–233 (1989)

CONJUGATE GRADIENT METHODS AND ILU PRECONDITIONING OF NON-SYMMETRIC MATRIX SYSTEMS WITH ARBITRARY SPARSITY PATTERNS

HANS PETTER LANGTANGEN

Department of Mathematics, University of Oslo, PO Box 1053, Blindern, N-0316 Oslo 3, Norway

SUMMARY

Preconditioning techniques based on incomplete Gaussian elimination for large, sparse, non-symmetric matrix systems are described. A certain level of fill-in may be specified in the incomplete factorizations. All methods considered may be applied to matrices with arbitrary sparsity patterns, for instance those associated with the general preprocessor algorithms or adaptive mesh techniques. The preconditioners have been combined with five conjugate gradient-like methods and tested on finite element discretized scalar convection–diffusion equations in 2D and 3D. It is found from numerical experiments that an amount of fill-in corresponding to about 50% of the number of original non-zero matrix entries is the optimal choice for this class of preconditioners. The preconditioners show almost no sensitivity to grid distortion. In problems with significantly variable coefficients or anisotropy the preconditioners stabilize the basic iterative schemes in addition to reducing the computational work substantially, mostly by more than 90%. The modified preconditioning technique, where fill-in is added on the main diagonal, performs in general better than the standard incomplete LU factorization, but is inferior to the latter in 3D problems and for matrix systems with complicated sparsity patterns.

KEY WORDS Preconditioning Conjugate gradients Non-symmetric matrices Finite elements Convective transport

1. INTRODUCTION

When a large, sparse, linear system of equations $\mathbf{Ax}=\mathbf{b}$, $\mathbf{A}\in\mathbf{R}^{n,n}$ and $\mathbf{b}, \mathbf{x}\in\mathbf{R}^{n}$, is solved by iterative methods, the convergence rate can mostly be greatly improved by applying the iterative method to an equivalent preconditioned system

$$\mathbf{M}^{-1}\mathbf{Ax}=\mathbf{M}^{-1}\mathbf{b}. \tag{1}$$

The non-singular preconditioning matrix $\mathbf{M}$ should be a good approximation to $\mathbf{A}$ and easy to compute, and equation systems of the form $\mathbf{Mc}=\mathbf{d}$ should be effectively solved in $O(n)$ operations. In addition, $\mathbf{M}$ should have low storage requirements. A popular choice for $\mathbf{M}$ is $\mathbf{M}=\mathbf{LU}$, where $\mathbf{LU}$ is an incomplete LU(ILU) factorization of $\mathbf{A}$; see, e.g. Reference 1.

In this paper the attention is focused on ILU preconditioning of non-symmetric equation systems which arises from finite element discretization of non-self-adjoint differential operators. Such operators occur frequently in hydrodynamical applications. If the physical problem involves complicated geometries, a preprocessor is usually required to partition the domain into finite. elements. General preprocessor techniques often produce an irregular node numbering which leads to a stiffness matrix with a complicated sparsity pattern. Similar sparsity patterns are also produced by adaptive mesh techniques. We will therefore be concerned with incomplete LU

0271-2091/89/020213-21$10.50

Received 22 April 1988
Revised 15 August 1988

factorization preconditioners that handle matrices with arbitrary sparsity patterns. Thus no restrictions are made by the method in this paper with respect to the matrix structure, the nodal numbering or the number of unknowns per node. However, **A** must be sparse in order to gain efficiency. The requirement of generality forces us to consider only pointwise preconditioners, in contrast to the probably more efficient block preconditioners.[2, 3]

A preconditioner is usually combined with an acceleration method. Five conjugate gradient-like methods are applied as accelerators in this paper: orthomin(k)[4] (OM(k)), truncated orthominres(k)[5] (T-OMR(k)), biconjugate gradients[6] (BiCG), conjugate gradients for the normal equations[7] (CG-$\mathbf{A}^{\mathrm{T}}\mathbf{A}$) and conjugate gradients squared[8] (CGS). In these iterative methods the coefficient matrix is used only in matrix–vector products. BiCG and CG-$\mathbf{A}^{\mathrm{T}}\mathbf{A}$ also involve a product of $\mathbf{A}^{\mathrm{T}}$ and an n-vector in each iteration, which gives rise to equation systems of the form $\mathbf{M}^{\mathrm{T}}\mathbf{c}=\mathbf{d}$ in the preconditioned version. Conjugate gradient-like methods are simple to implement and have low storage requirements.

ILU preconditioning of general non-symmetric linear systems has previously been treated in, for example, References 4 and 9. Most of the previous work on incomplete factorization preconditioners has been restricted to matrices with special regular sparsity patterns, for example those arising from natural, red–black, D2- or D4-diagonal orderings[10] on rectangular or box-shaped domains. The major part of the contributions is also devoted to finite difference methods and symmetric positive definite matrix systems.[2, 3, 11–13] Practical hydrodynamical problems often involve convection operators, complicated geometries, adaptive meshes and implicit time integration. As a result, large, sparse, non-symmetric matrix systems with arbitrary sparsity patterns must be solved, and a large amount of the total computational work is spent on this solution procedure. The need for fast iterative methods with low storage requirements is therefore substantial.

This work focuses attention on the dependence of incomplete LU factorization preconditioners on the sparsity pattern of **A** (i.e. the nodal numbering), the sparsity pattern of **M** (i.e. the amount of allowable fill-in), the element distortion and the nature of the test problem. All test problems are special cases of the stationary convection–diffusion equation that models the convective and diffusive transport of a scalar quantity (temperature, pollution, etc.) in a flow field. The convection–diffusion equation is also often used as a simple model equation for testing numerical techniques, such as equation solvers, which are intended to be applied to more complicated systems of non-linear hydrodynamical conservation laws.

Section 2 treats incomplete LU factorization when the sparsity of **M** is known, and choices of the sparsity pattern of **M** are presented. Some specific examples of sparsity patterns are shown in Section 3. Results from numerical experiments are presented in Section 4.

2. ILU PRECONDITIONING

The preconditioning matrix **M** is chosen to be an incomplete LU decomposition of **A**. Let P be a set of pairs of indices (i, j), $1 \leqslant i, j \leqslant n$, representing the desired sparsity pattern (non-zeros) of **M**. An incomplete LU factorization may then be obtained by performing naive Gaussian elimination on **A**, rejecting all fill-in entries (k, l) if $(k, l) \notin P$. A common choices is $P=\{(i,j) \mid A_{i,j} \not\equiv 0\}$ (by $A_{i,j} \not\equiv 0$ we mean the entries in **A** which are not trivially equal to zero on the basis of the discretization method and the nodal numbering).This choice leads to an ILU preconditioner where all fill-in entries are rejected. Gustafsson[14] has proposed a modification to this preconditioning where the fill-in entries are added to the main diagonal. This version of the ILU preconditioner is called *modified* ILU (MILU) preconditioning.

Meijerink and van der Vorst[12] have shown existence and uniqueness of ILU factorization when $\mathbf{A}$ is an M-matrix. The MILU factorization is stable if $\mathbf{A}$ is diagonally dominant.[15] Stability of the ILU factorization for non-symmetric systems has been discussed by Elman.[16] These theoretical results are based on model problems such as Poisson's equation or the convection–diffusion equation, mostly with constant coefficients.

If we let P contain all pairs of indices within the band or skyline of $\mathbf{A}$, the LU factorization becomes exact. The approximation $\mathbf{M}$ to $\mathbf{A}$ is improved by increasing the number of pair of indices in P. On the other hand, the cost of calculating $\mathbf{M}$ and solving systems with $\mathbf{M}$ as coefficient matrix is increasing with increasing number of elements in P. The storage required by $\mathbf{M}$ is also proportional to the number of elements in the set P.

By allowing a certain amount of fill-in in $\mathbf{M}$, it is possible to improve the incomplete LU factorization. This will of course increase both the storage requirements and the computational labour in each iteration of the iterative equation solver. Nevertheless, it is possible to reduce the number of iterations so that the *total* computational work is lowered.

Two different improvements have been popular in the literature. The first approach consists in determining the set P on the basis of the couplings between the unknowns (i.e. on the basis of the element topology). In References 13 and 14 this method is applied with success (they considered mainly the five-star finite difference approximation to the Laplacian on a regularly numbered grid). The second approach determines the elements in P during the elimination. P will then contain the index pair (k, l) if the corresponding fill-in entry has magnitude larger than a given threshold. This technique is described in Reference 17 (symmetric problems) and in Reference 4 (non-symmetric problems). It is expected that this second approach will yield a better incomplete LU factorization than the first, although no comparison between the two methods seems to have appeared in the literature. However, in the second approach each fill-in entry must be calculated before the rejection criterion can be employed. This fact makes the elimination procedure more cost-expensive than in the first approach. Also, rejecting the fill-in entries on the basis of their magnitudes means that P may change significantly from problem to problem (or from time step to time step) even when the spatial discretization remains the same. The memory available may then suddenly be too small, causing unexpected program abortion. Because of this practical disadvantage, we will base the improved ILU factorization on the element or cell topology only. We must mention, however, that Axelsson and Munksgaard[18] have devised an approach which adjusts the rejection threshold according to the currently available computer memory.

The work of Gustafsson[14] on constructing improvements to the common choice of the sparsity of $\mathbf{M}$, $P=\{(i,j)|A_{i,j}\neq 0\}$, will here be extended to matrices with arbitrary sparsity patterns. We call the resulting preconditioning ILU(l) factorization (or correspondingly, MILU(l) factorization if fill-in entries are added on the main diagonal). l is a parameter describing the level of allowed fill-in. $l=0$ corresponds to the standard ILU factorization, that is, $P=\{(i,j)|A_{i,j}\neq 0\}$. ILU($l$) is defined in terms of ILU($l-1$) as follows. Let P_{l-1} correspond to the index set at level $l-1$, that is, the set used for computing ILU($l-1$). Set $P_l=P_{l-1}$ and form the product of $\mathbf{L}$ and $\mathbf{U}$. The sparsity of $\mathbf{L}$ and $\mathbf{U}$ is given by P_{l-1}. However, the product $\mathbf{LU}$ will in general contain non-zero terms not represented in P_{l-1}. If entry (l, m) in the product $\mathbf{LU}$ is not trivially equal to zero, then the index pair (l, m) is added to P_l. One should note that no multiplications are needed to determine P_l and that the non-zeros of $\mathbf{LU}$ are usually distributed symmetrically: $(l, m)\in P_l \Leftrightarrow (m, l)\in P_l$.

Algorithms for the methods sketched in this section are given in the Appendix.

3. EXAMPLES OF SPARSITY PATTERNS

To demonstrate how the sparsity pattern P_l grows with the parameter l, a few examples are shown next. The two-dimensional examples represent discretization of the unit square shown in Figure 1.

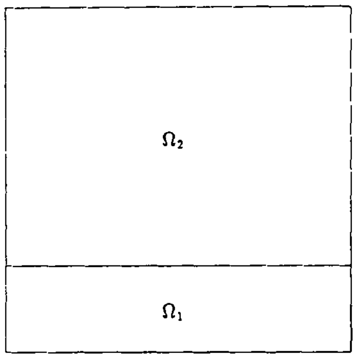


Figure 1. Sketch of a two-dimensional domain $\Omega=\Omega_1\cup\Omega_2$

The unit square is divided into two super elements, Ω_1 and Ω_2. The number of bilinear finite elements in the x- and y-direction is denoted n_x and n_y^i respectively, where $i=1, 2$ corresponds to Ω_1, Ω_2. A regular node numbering is obtained by numbering the nodes row by row, starting at $y=0$.

Figure 2 shows the sparsity pattern P_0 for a mesh with 20 x 20 bilinear elements. There are 3 x 3 diagonals with non-zeros. The sparsity pattern P_1 is similar to those given in References 13 and 14, that is, four additional diagonals are included. There are 19 diagonals in P_2.

Figure 3 shows the sparsity pattern P_0 for an element partitioning similar to the case in Figure 2 but with an irregular node numbering. One of the main advantages of the finite element method is its easy handling of complex geometries. Often a general preprocessor technique, which mostly produces an irregular nodal numbering, is required for the partitioning of complicated domains. Such a general technique is applied here to a simple geometry for comparison of matrix sparsity patterns corresponding to regular and irregular node numberings. When the general preprocessor algorithm is used, the interior nodes in each super element are first numbered regularly (row by row), then the nodes on the boundaries of the super elements are numbered.

Figures 4 and 5 show the P_1 and P_2 sparsity patterns for the irregular node numbering. In the part of the matrix where the sparsity pattern is regular, the additional elements in P_l are situated

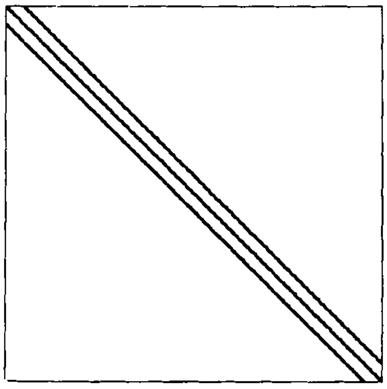


Figure 2. Sparsity pattern P_0 corresponding to a bilinear finite element partitioning of the domain in Figure 1; $n_x=20$, $n_y^1=4$ and $n_y^2=16$; regular numbering; 3721 non-zeros

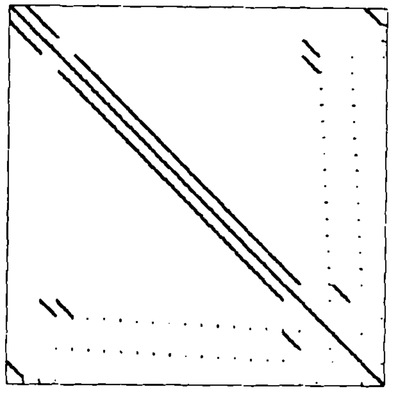


Figure 3. Sparsity pattern P_0 of a mesh similar to that in Figure 2, except that the node numbering is irregular; 3721 non-zeros

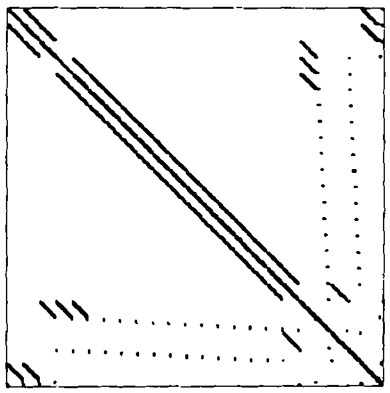


Figure 4. Sparsity pattern P_1 corresponding to the same mesh and node numbering as used in Figure 3; 5241 non-zeros

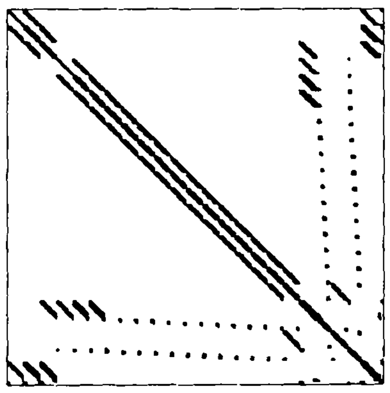


Figure 5. Sparsity pattern P_2 corresponding to the same mesh and node numbering as used in Figure 3; 7327 non-zeros

along diagonals as previously outlined. In the remaining parts the elements are to a large extent clustered around the original elements in P_0. The total amount of allowable fill-in, for a given l, is larger here than for a regular sparsity pattern.

We have also calculated the sparsity of some smaller 3D problems. Figure 6 shows the sparsity pattern P_0 for a regularly numbered box. The elements were of the trilinear type. Figure 7 displays the corresponding P_1 sparsity pattern. In Figure 8 we have plotted the non-zeros in the coefficient matrix that arose from an irregular numbering of the 3D grid used in Figure 6. The domain consisted of two super elements, and the node numbering algorithm was similar to that in the 2D case. That is, all internal nodes within each super element are numbered regularly (row by row, plane by plane), and finally the nodes on the surfaces of the super elements are given numbers. Figure 9 shows the P_1 sparsity pattern that corresponds to Figure 8. From Figures 7–9 we see that the number of non-zeros in **M** increases much more rapidly in 3D than in 2D problems. Moreover, the irregular node numbering leads to a larger increase than the regular numbering.

In adaptive mesh algorithms it is common to have a coarse, regularly numbered 'parent' mesh with local refinements. The nodes in the refined regions usually have higher numbers than the nodes in the 'parent' mesh. The corresponding sparsity patterns thus become very similar to the

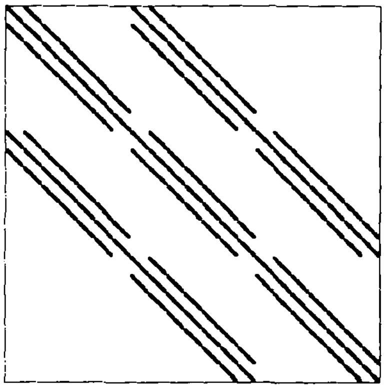


Figure 6. Sparsity pattern P_0 corresponding to a three-dimensional trilinear finite element mesh with 273 nodes and regular node numbering; 4921 elements in P_0

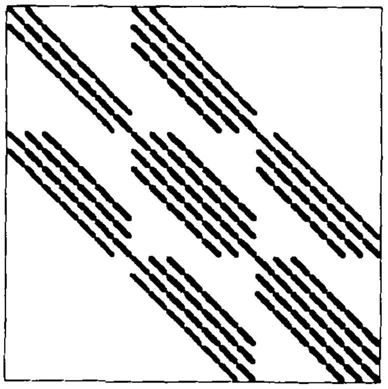


Figure 7. Sparsity pattern P_1 corresponding to a three-dimensional finite element mesh with 273 nodes

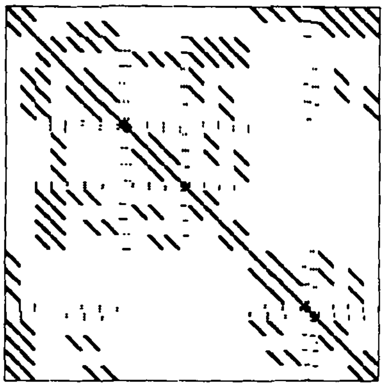


Figure 8. Sparsity pattern P_0 corresponding to an irregular numbering of nodes in the mesh used for Figure 6; 4921 non-zeros

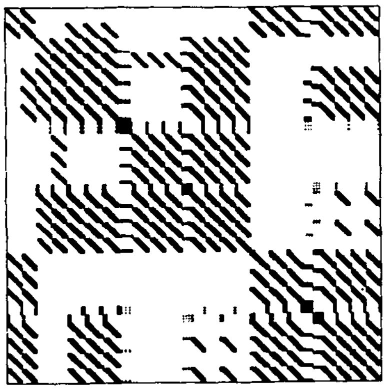


Figure 9. Sparsity pattern P_1 corresponding to the mesh and node numbering in Figure 8; 13833 non-zeros

sparsity patterns associated with the general preprocessor algorithm outlined here. Preconditioners that handle the arbitrary sparsity patterns exemplified in the figures are therefore of importance not only in problems with complicated geometries but also in situations where simple meshes are locally refined to resolve, for example, shock fronts.

It should be observed that trilinear elements with one unknown per node result in 27 diagonals in **A**. The matrix–vector product $\mathbf{M}^{-1}\mathbf{Ac}$ then dominates the cost in each iteration of almost any conjugate gradient-like method. Thus the overall efficiency is crucially dependent on the implementation of the matrix–vector product. Use of higher-order elements decreases the sparsity, and this leads to more work per iteration without reducing the number of iterations. For example, the 27-node 3D brick element leads to 125 non-zero diagonals in **A**. Each iteration is then roughly four times as expensive as an iteration with an **A** corresponding to the trilinear brick element, assuming n fixed. Although higher-order elements allow less unknowns, this cannot compensate for the decreased sparsity. The same argument also favours finite difference methods in 3D where **A** has only seven non-zero diagonals.

4. NUMERICAL EXPERIMENTS

4.1. Description of test problems

All test problems considered herein are special cases of the linear, stationary convection–diffusion equation governing the transport of a scalar quantity in fluid flow.

As start vector for the iterative methods we have employed the null vector with essential boundary conditions inserted. The computational labour of an equation solver applied to a specific problem is measured in *work units.* One work unit is defined as one multiplication (division) per unknown.[10] Work units are given to three significant digits. An iterative equation solver was considered to be converged when the initial residual was reduced by a factor of 10^{-4}. We also stopped the iteration when the number of iterations exceeded a given level.

4.1.1. Problem 1: Thermal shear layer, regular node numbering. Let

$$\Omega=[0,1]\times[0,1],$$
$$\Gamma_1=\{x=0,\ 0\leqslant y\leqslant\tfrac{1}{4}\}\cup\{0\leqslant x\leqslant 1,\ y=0\},$$
$$\Gamma_2=\{x=0,\ \tfrac{1}{4}<y\leqslant 1\}.$$

Test problem 1 reads

$$v\left(\cos\alpha\frac{\partial u}{\partial x}+\sin\alpha\frac{\partial u}{\partial y}\right)=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\quad\text{in}\quad\Omega,\tag{2}$$
$$u=0\quad\text{on}\quad\Gamma_1,$$
$$u=1\quad\text{on}\quad\Gamma_2,$$
$$\partial u/\partial n=0\quad\text{on}\quad\partial\Omega\backslash(\Gamma_1\cup\Gamma_2),$$

where v and α are given constants: $0<v<\infty$ and $0\leqslant\alpha\leqslant\pi/2$.

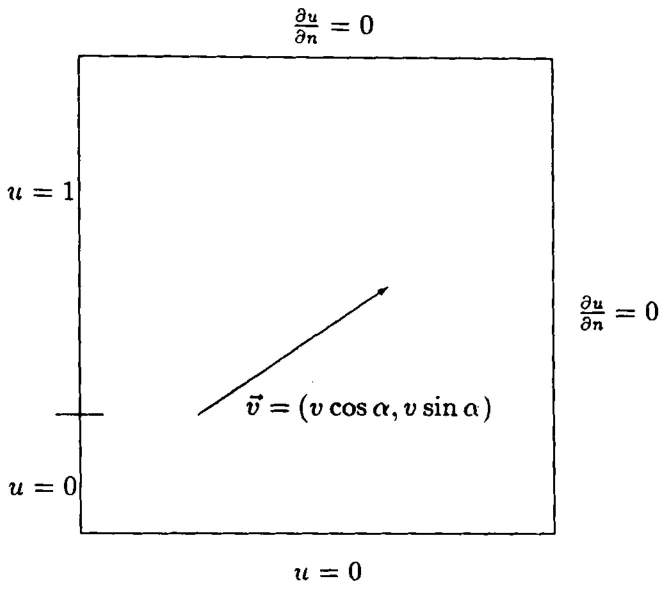
Physically, this problem may model the temperature field when two fluids with equal velocities but different temperatures meet at $x=0$. Figure 10 gives a sketch of the domain and the boundary conditions, and Figure 11 displays the solution u for two values of v. The discontinuous Dirichlet condition at $(0,\frac{1}{4})$ leads to a steep front in the solution for large values of v. α represents the angle between the front and the x-axis.

Because of possible numerical instabilities, equation (2) was discretized with a Petrov–Galerkin method with weighting functions of the form

$$\hat{N}_i=N_i+\tilde{k}\left(v\cos\alpha\frac{\partial N_i}{\partial x}+v\sin\alpha\frac{\partial N_i}{\partial y}\right),\tag{3}$$

where N_i is the trial function for u. Square-shaped, equal-sized bilinear elements with a regular node numbering were used. The value of $\tilde{k}$ in our Petrov–Galerkin formulation may be found in Reference 19. The Petrov–Galerkin formulation increases the convergence rate at high Peclet numbers even when the standard Bubnov–Galerkin formulation ($\tilde{k}=0$) leads to physically acceptable results.

4.1.2. Problem 2: Thermal shear layer, irregular node numbering. This test problem is identical to problem 1 except that the node numbering is irregular. The irregular node numbering was produced by a general finite element preprocessor algorithm as described in Section 3. The associated sparsity patterns are displayed in Figures 2–5.



$\frac{\partial u}{\partial n} = 0$
$u = 1$
$u = 0$
$\frac{\partial u}{\partial n} = 0$
$\vec{v} = (v \cos\alpha, v \sin\alpha)$
$u = 0$

Figure 10. Sketch of the domain and boundary conditions in test problem 1

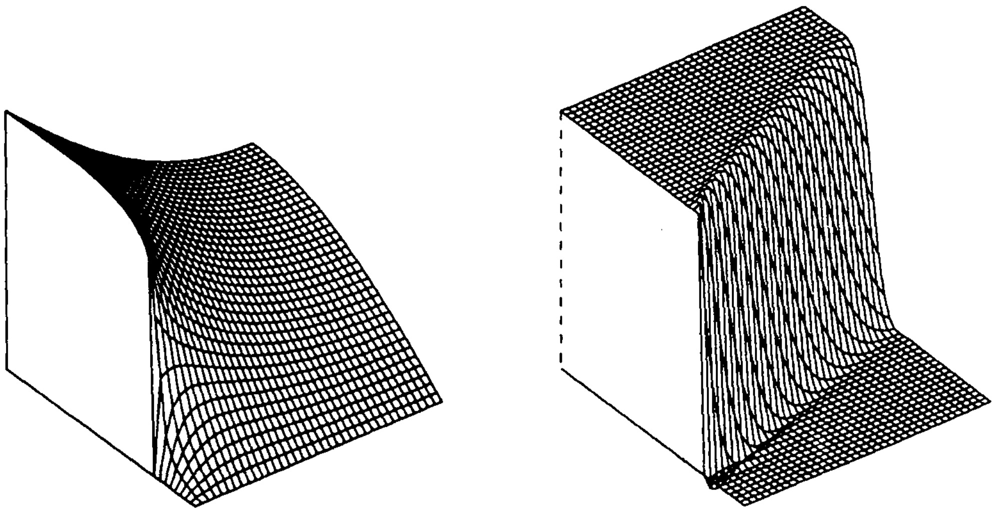


Figure 11. Solution $u(x, y)$ in test problem 1; $\alpha=\pi/4$. Left: $v=0{\cdot}1$; Right: $v=1000$

4.1.3 Problem 3: 3D thermal transport in porous media. Let Ω be a two-layered porous cube:

$$\Omega=[0, 1]\times[0, 1]\times[0, 1].$$

Four injection wells are situated at the bottom of the cube, while one production well is placed at the top. The stationary temperature field in Ω is governed by

$$u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}+w\frac{\partial T}{\partial z}=\frac{\partial}{\partial x}\left(\kappa_x\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\kappa_y\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\kappa_z\frac{\partial T}{\partial z}\right), \tag{4}$$

where κ_x, κ_y and κ_z are the thermal diffusivities in the x-, y- and z-direction respectively. In layer 1, $0<z\leqslant\frac{1}{2}$, the diffusivities are denoted as $\kappa_x^{(1)}$, $\kappa_y^{(1)}$ and $\kappa_z^{(1)}$. The diffusivities in layer 2, $\frac{1}{2}<z<1$, are similarly denoted as $\kappa_x^{(2)}$, $\kappa_y^{(2)}$ and $\kappa_z^{(2)}$.

At $\partial\Omega$ the no-flux condition is imposed, and at the five wells we set

$$T(0,0,0)=T(1,0,0)=T(0,1,0)=T(1,1,0)=0, \qquad T(1,1,1)=1.$$

The fluid velocity $(u, v, w)^{\mathrm{T}}$ is given in terms of the pressure field p by

$$u(x,y,z)=-k_x\frac{\partial p}{\partial x}, \qquad v(x,y,z)=-k_y\frac{\partial p}{\partial y}, \qquad w(x,y,z)=-k_z\frac{\partial p}{\partial z},$$

where k_x, k_y and k_z are the permeabilities in the x-, y- and z-direction respectively. The permeabilities used in our numerical experiments were $k_x=k_y=10^4$ and $k_z=10^5$ in layer 1 and $k_x=k_y=10^2$ and $k_z=1$ in layer 2. It is assumed that the effects of gravity and thermal convection are negligible in comparison with the pressure gradient. The pressure fulfils

$$\frac{\partial}{\partial x}\left(k_x\frac{\partial p}{\partial x}\right)+\frac{\partial}{\partial y}\left(k_y\frac{\partial p}{\partial y}\right)+\frac{\partial}{\partial z}\left(k_z\frac{\partial p}{\partial z}\right)=0 \tag{5}$$

in Ω if the fluid is incompressible. The boundary conditions for p are in this example equal to those for $(-T)$.

The equations for T and p were discretized by a finite element method employing equal-sized trilinear elements. A Petrov–Galerkin formulation of type used for test problem 1 was applied for the temperature equation. Because of limited computer resources, we tested the 3D problem with only one mesh. The mesh consisted of $10\times10\times12=1200$ elements and 1573 nodes. An ILU(0) preconditioned standard conjugate gradient method was used for solving the symmetric, positive definite matrix system arising from a Bubnov–Galerkin formulation of equation (5). This iterative method required 23 iterations, corresponding to 1330 work units.

The temperature equation is of hyperbolic nature in the well regions. Nevertheless, the mean Peclet number in Ω is moderate. Figure 12 shows equidistant contour lines of the temperature field on the surface of the reservoir. The diffusivities in Figure 12 correspond to $\kappa_x^{(1)}=1$, $k_y^{(1)}=\kappa_z^{(1)}=2$, $\kappa_z^{(1)}=2$, $\kappa_x^{(2)}=\kappa_y^{(2)}=\kappa_z^{(2)}=20$.

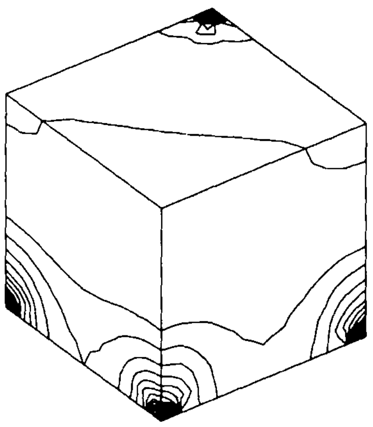


Figure 12. Test problem 3: equidistant contour lines of the temperature field on the facing surface of the two-layered, anisotropic porous reservoir with fluid flow

4.1.4. Problem 4: 2D version of test problem 3. A 2D version of test problem 3 is constructed in order to test differences between the preconditioners in 2D and 3D. Consider the fluid flow and temperature distribution in the plane $y=1$ in test problem 3. We set

$$\frac{\partial}{\partial y}=0 \qquad \kappa_y^{(i)}=k_y=0, \qquad i=1,2$$

and apply the same boundary conditions and the same values of permeabilities and diffusivities as in test problem 3. The mesh was identical to that for $y=1$ in problem 3, i.e. 10×12 bilinear elements with 143 nodes.

4.1.5. Problem 5: Temperature distribution in a flow field around a body. Consider a cold body moving with constant velocity in a hot, incompressible, viscous fluid. We introduce a co-ordinate system where the body is at rest so that stationary conditions can be assumed. For simplicity, only two-dimensional variations are taken into account. The temperature field T is governed by the convection–diffusion equation

$$u(x,y)\frac{\partial T}{\partial x}+v(x,y)\frac{\partial T}{\partial y}=\kappa\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right) \quad \text{in} \quad \Omega. \tag{6}$$

The diffusivity κ is regarded as constant, whereas the velocity field (u and v) must be found by solving the Navier–Stokes equations and the equation of continuity in Ω. We assume that the internal heat source due to dissipation can be neglected in the equation governing the temperature distribution.

Equation (6) was discretized by a Petrov–Galerkin finite element method, as in test problem 1, using isoparametric bilinear elements. The incompressible viscous flow field was solved by a Navier–Stokes code employing an implicit penalty function method with selective reduced integration on bilinear elements.

The boundary conditions for T are

$$\begin{aligned} T=1 &\quad \text{on} \quad \Gamma_1,\\ T=0 &\quad \text{on} \quad \Gamma_2\\ \partial T/\partial n=0 &\quad \text{on} \quad \partial\Omega\backslash(\Gamma_1\cup\Gamma_2). \end{aligned}$$

Figure 13 shows a bilinear finite element mesh with 1011 nodes and an indication of the boundaries Γ_1 and Γ_2. In contrast to the grids in problems 1–4, this mesh is slightly distorted and unequally spaced. The velocity field used in the numerical experiments herein corresponds to $Re=6$ and is displayed in Figure 14 (Re is the flow Reynolds number based on the diameter of the

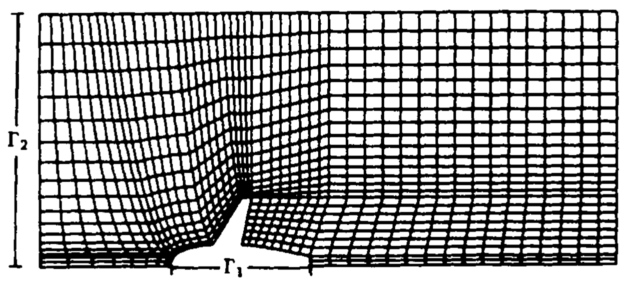


Figure 13. Test problem 5: bilinear finite element mesh around the body; 1011 nodes

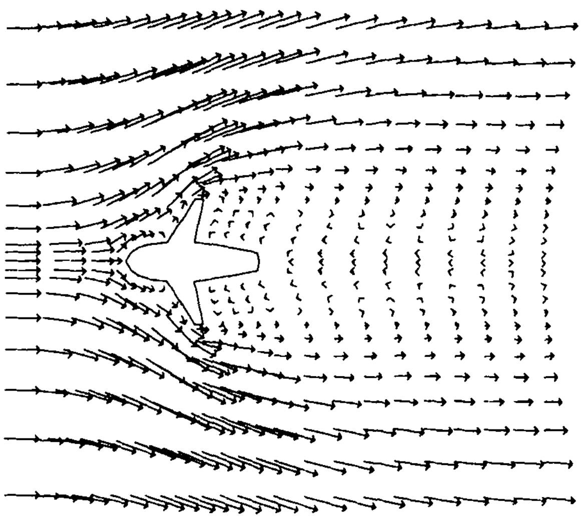


Figure 14. Test problem 5: velocity field; $Re=6$

body). The most pronounced features are the high velocity gradients upstream and the vortex downstream of the body. The equation for T will thus have dominating hyperbolic terms upstream of the body, while the local nature of (6) in the recirculating region will be elliptic. Figure 15 displays contour lines of the temperature field for $\kappa=0{\cdot}2$ and (u, v) as in Figure 14. There is a sharp front close to the stagnation point of the body. The temperature field is smooth in the vortex region.

4.2. *The effect of ILU/MILU preconditioning*

In this subsection we consider T-OMR(5) with ILU(0), MILU(0) and no preconditioning in order to demonstrate the effect of incomplete LU factorization preconditioners. The four other equation solvers have also been tested and the results were in qualitative accordance with those of T-OMR(5).

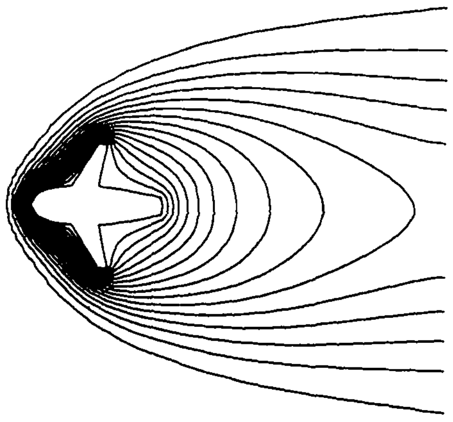


Figure 15. Test problem 5: the temperature field visualized by equidistant contour lines; $\kappa=0{\cdot}2$

Test problem 1 has been run for different values of v, α and n, where n is the number of nodes (unknowns). At small v values ($v \leqslant 10$) the convergence of the preconditioned versions was considerably slower than the convergence rate at larger v values ($v \geqslant 100$). Table I shows the number of iterations and work units required by T-OMR(5) without preconditioning and with ILU(0) and MILU(0) preconditioning for different values of n and v. The convergence rate and also the total amount of computational work were greatly improved by the incomplete LU factorization preconditioners, especially for $v \geqslant 100$. In this test problem MILU preconditioning performed more efficiently than the standard ILU preconditioner.

The convergence of conjugate gradient-like methods depends on the eigenvalue spectrum of $\mathbf{M}^{-1}\mathbf{A}$, which should be as clustered as possible. We have computed the complex eigenvalue distribution for $\alpha = \pi/4$ and different values of v. The ILU(0) and ILU(1) preconditioners cluster the spectrum considerably, especially when $v \geqslant 100$. The spectrum associated with MILU(l) was almost identical to the ILU(l) spectrum. The clustering effect explains to some extent why incomplete LU factorization preconditioners reduce the number of iterations. However, no information from the eigenvalue spectrum seemed to be relevant for explaining the superior behaviour of MILU compared with ILU.

In test problems 3–5 the effect of preconditioning was even greater than in test problem 1. Table II displays the behaviour of T-OMR(5) in test problems 3 and 4. The number of work units in problems 3 and 4 differs considerably, but this is because n in problem 3 is more than ten times larger than n in problem 4. Fast convergence was obtained for $\kappa \ll 1$, otherwise the convergence rate for the preconditioned methods was not very sensitive to variations in the diffusivities. As the anisotropy increased, the non-preconditioned equation solvers converged considerably slower.

Table I. Performance of T-OMR(5) with and without preconditioning in test problem 1 for different values of n and v. The results show the number of work units with the number of iterations in parentheses

Preconditioner	v	$n=441$	$n=1369$	$n=4761$	$n=9409$
None	0·1	1180 (26)	2640 (58)	3020 (66)	4890 (106)
ILU(0)	0·1	463 (7)	972 (15)	1490 (23)	1960 (31)
MILU(0)	0·1	401 (6)	658 (10)	1170 (18)	1170 (18)
None	100	955 (21)	1460 (32)	2470 (54)	3330 (72)
ILU(0)	100	153 (2)	281 (4)	537 (8)	654 (10)
MILU(0)	100	153 (2)	218 (3)	410 (6)	468 (7)

Table II. Performance of T-OMR(5) with and without preconditioning in test problems 3 and 4. The results show the number of work units with the number of iterations in parentheses. κ-set 1 corresponds to $\kappa_x^{(1)} = \kappa_y^{(1)} = \kappa_z^{(1)} = \kappa_x^{(2)} = \kappa_y^{(2)} = \kappa_z^{(2)} = 2$. κ-set 2 involves anisotropy: $\kappa_x^{(1)} = 1$, $\kappa_y^{(1)} = \kappa_z^{(1)} = 2$, $\kappa_x^{(2)} = \kappa_y^{(2)} = \kappa_z^{(2)} = 20$

Preconditioner	κ-set	Test problem 3 3D, $n=1573$	Test problem 4 2D, $n=143$
None	1	5590 (76)	1080 (24)
ILU(0)	1	1100 (8)	392 (6)
MILU(0)	1	2410 (19)	453 (7)
None	2	21900 (299)	3970 (89)
ILU(0)	2	1580 (12)	332 (5)
MILU(0)	2	6200 (51)	392 (6)

From Table II it is also evident that MILU(0) is considerably inferior to ILU(0) in 3D. The behaviour of ILU(0) and MILU(0) is quite similar in the 2D version of test problem 3. MILU(0) was in fact better than ILU(0) when combined with BiCG or CGS. The results from test problem 5, which are given in Table III, showed little dependence on the diffusivity. We notice that without preconditioning all equation solvers converged very slowly in these three test problems.

It may be mentioned that other preconditioners such as the SOR and SSOR schemes have also been tested for comparison. These preconditioners were seldom competitive with and clearly less reliable than the ILU/MILU preconditioners.

4.3. Effect of irregularly numbered grids and distorted elements

In this subsection we will compare the performance of the incomplete LU factorization preconditioners when the elements are distorted and when the node numbering is irregular with a corresponding complex sparsity pattern of **A** and **M**. We consider the same test problem as in the previous subsection, except that $v = 10$ and only $n = 1369$ is considered. By grid 1 we will mean the mesh used in the previous subsection (regular numbering and uniform element partitioning). An irregular node numbering (test problem 2) is applied for grid 2, and grid 3 refers to a mesh with distorted elements and regular node numbering. The maximum angle at the corners of the most distorted bilinear elements was 120°, and the length of the sides of two adjacent elements differed at most by a factor of 20. Grid 3 is displayed in Figure 16. The high distortion should not be

Table III. Performance of T-OMR(5) with and without preconditioning in test problem 5. The results show the number of work units with the number of iterations in parentheses. $\kappa = 0{\cdot}2$ and $n = 1011$

Preconditioner	Test problem 5
None	5700 (126)
ILU(0)	528 (8)
MILU(0)	404 (6)

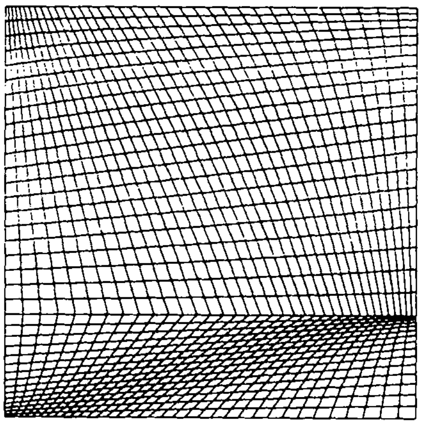


Figure 16. Distorted bilinear finite element mesh corresponding to grid 3

accepted in a practical problem, but may serve as a test case. Tables IV and V compare the performance of T-OMR(5) and BiCG for these three grids.

The distorted mesh (grid 3) had a significantly negative influence on the convergence rate when no preconditioning was used. Almost no negative influence of the distorted grid was observed when the equation solver was preconditioned.

The irregularly numbered mesh (grid 2) had no influence on the non-preconditioned iterative methods. The ILU(0) preconditioner required slightly more work for grid 2 than for grids 1 and 3, while MILU(0) performed poorly. Several other values of v and α were also run for gird 2, and all results indicate that the *modified* incomplete LU factorization preconditioner is not suited for sparsity patterns associated with the irregular node numbering where the amount of fill-in is larger than for the regularly numbered grid.

We have also briefly studied test examples where n is kept fixed and the bandwidth is varied. For small bandwidths MILU preconditioning was clearly more effective than ILU. As the bandwidth was increased, the superior performance of MILU relative to ILU decreased, and ILU was better than MILU for large bandwidths. In this context it must be mentioned that ILU also became slightly less effective as the bandwidth was increased. Our experience indicates that the efficiency of MILU versus ILU depends on the ratio between the number of fill-in entries and n. This assertion is in accordance with the observed inefficiency of MILU in 3D problems. However, other factors also influence the behaviour of MILU, and no rigorous conclusions can presently be drawn for predicting when MILU should be applied in preference to ILU.

4.4. *Test of ILU(l)/MILU(l) preconditioning*

The ILU(l) and MILU(l) preconditioners usually reduce the number of iterations as l increases. However, the computational work associated with the factorization of $\mathbf{M}$ and the inversion of $\mathbf{M}$ increases very rapidly with l. It is therefore of interest to find an optimal value of l.

Three parameter sets in problem 1 with $n=1369$ and $w=0{\cdot}1$, 10, 100 were tested for $l=0, 1, 2, 3, 4$. The average results are shown in Table VI for ILU(l) and in Table VII for MILU(l). The optimal choice seemed to be $l=2$ for ILU and $l=1$ for MILU. CG-$\mathbf{A}^T\mathbf{A}$ had the best performance

Table IV. Performance of T-OMR(5) applied to three different grids

Preconditioner	Grid 1 ('regular')	Grid 2 ('irregular')	Grid 3 ('distorted')
None	1600 (35)	1600 (35)	2420 (53)
ILU(0)	595 (9)	658 (10)	595 (9)
MILU(0)	469 (7)	divergence	407 (6)

Table V. Performance of BiCG applied to three different grids

Preconditioner	Grid 1 ('regular')	Grid 2 ('irregular')	Grid 3 ('distorted')
None	1620 (66)	1620 (66)	2350 (96)
ILU(0)	824 (19)	907 (21)	865 (20)
MILU(0)	532 (12)	1700 (40)	490 (11)

Table VI. Average results in work units based on three parameter sets in test problem 1; ILU(l) preconditioning

Method	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$
OM(5)	678	507	433	472	658
T-OMR(5)	616	453	444	472	693
BiCG	699	600	594	640	804
CGS	476	384	373	461	638
CG-A^TA	1810	1110	837	762	939

Table VII. Average results in work units based on three parameter sets in test problem 1; MILU(l) preconditioning

Method	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$
OM(5)	516	438	402	453	652
T-OMR(5)	448	382	389	472	636
BiCG	518	501	514	640	770
CGS	365	335	353	512	638
CG-A^TA	1190	870	760	787	939

for $l=3$ (ILU) and $l=2$ (MILU). BiCG was less sensitive to variations in l than the other equation solvers.

A serious drawback with the ILU(l) and MILU(l) preconditioners is the large storage requirements. Let $n_z^M(l)$ be the number of elements in the sparsity pattern of **M**. In this test we have $n_z^M(0)=11\,881$, $n_z^M(1)=16\,921$, $n_z^M(2)=24\,127$, $n_z^M(3)=35\,319$ and $n_z^M(4)=51\,147$. In general the relation $n_z^M(l)\approx n_z^M(0)\times 1{\cdot}44^l$ holds for regularly numbered grids consisting of bilinear elements. Notice that the choice ILU(2) requires more than twice as many memory locations as ILU(0).

If the nodes are reordered so that fill-in is minimized, one should expect that the incomplete LU factorization is improved. Simon[9] tested several such node numbering algorithms with the ILU(0) preconditioner and found only slightly increased efficiency of orthomin. From the figures in Section 5 it is seen that the irregular node numbering produced by a general preprocessor technique results in a more rapid growth of $n_z^M(l)$ with l than does the regular node numbering. This indicates that the number of fill-in entries which are rejected in the ILU(0) preconditioner is larger for an irregular node numbering than for a regular one. We may thus expect that an irregular node numbering decreases the efficiency of an incomplete LU factorization preconditioner. The same test as shown in Table VI has been performed with an irregularly numbered grid, and the average results are reported in Table VIII. Since MILU(0) preconditioning performed less well than ILU(0) on the irregularly numbered grid in test problem 2, no results for MILU(l) are presented. A comparison of Tables VI and VIII shows that the irregular node numbering leads to an ILU(0) preconditioner which is inferior to the ILU(0) preconditioner associated with a regularly numbered grid. Nevertheless, the differences are surprisingly small, and we may conclude that there is no *essential* loss in efficiency when employing an irregular node numbering. A further step in the argument may lead us to expect that a numbering which is optimal with respect to the amount of fill-in probably gives only minor gain with respect to the efficiency of an ILU preconditioner.

Table VIII. Average results in work units based on three parameter sets in test problem 2; ILU(l) preconditioning

Method	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$
OM(5)	715	579	563	711	1130
T-OMR(5)	574	518	523	704	1170
BiCG	782	693	732	744	1460
CGS	462	437	492	680	1130
CG-A^TA	2380	1840	1800	2210	3010

In the test problem associated with Table VIII we had $n_z^M(0)=11\,881$, $n_z^M(1)=18\,055$, $n_z^M(2)=27\,683$ and $n_z^M(3)=43\,439$. $n_z^M(l)$ grows faster with l in this case than for the regularly numbered grid: $n_z^M(l)\approx n_z^M(0)\times 1{\cdot}6^l$.

From Tables VI and VIII we also see that the preconditioned CG-A^TA was far more sensitive to the node numbering than the other equation solvers. Our conclusion is that the node numbering has a minor influence on the behaviour of the tested ILU(l) preconditioned equation solvers, with the exception of CG-A^TA.

In test problem 3 the MILU preconditioner could not compete with the standard ILU preconditioner. Convergence results for ILU(l) with $l=0$, 1, 2 are displayed in Table IX. We see that, except for CG-A^TA, there is no gain in using $l>0$. $n_z^M(l)$ grows much faster with l in 3D than in 2D. In this test problem we had $n_z^M(0)=35\,557$, $n_z^M(1)=74\,797$ and $n_z^M(2)=156\,893$. In general, $n_z^M(l)=n_z^M(0)\times 2{\cdot}1^l$ for regularly numbered grids consisting of trilinear bricks. Most of the computational work per iteration in 3D problems is spent on the matrix–vector product $\mathbf{M}^{-1}\mathbf{A}\mathbf{z}$. The cost of the matrix–vector product in 3D is $3\times 1{\cdot}46^l$ times the cost of the product in 2D. This explains why $l>0$ is mostly ineffective in 3D problems.

An irregular node numbering as in test problem 2 has also been applied to the 3D problem. The results confirm the conclusions drawn from problem 2. However, as expected, the difference in efficiency of ILU(0) associated with the two numberings was larger in 3D than in 2D. For our irregularly numbered grid, $n_z^M(l)=n_z^M(0)\times 2{\cdot}7^l$. In the 2D version of test problem 3 the optimal l was as in test problem 1.

The ILU(l)/MILU(l) results for test problem 5 ($n=1011$, $\kappa=0{\cdot}2$, 5, 50) are reported in Tables X and XI. The optimal l values are in accordance with the results of the other two-dimensional test problems. Since l should be kept low because of storage requirements, it is felt that $l=1$ is the recommended value in this test example.

Table IX. Results in work units with the number of iterations shown in parentheses for test problem 3; ILU(l) preconditioning

Method	$l=0$	$l=1$	$l=2$
OM(5)	1530 (21)	1640 (11)	3550 (7)
T-OMR(5)	1580 (12)	1670 (6)	3930 (5)
BiCG	1910 (18)	2480 (12)	4350 (7)
CGS	1420 (13)	2010 (9)	4090 (6)
CG-A^TA	4300 (43)	3910 (21)	5760 (12)

Table X. Average results in work units based on three parameter sets in test problem 5; ILU(l) preconditioning

Method	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$
OM(5)	685	531	450	472	529
T-OMR(5)	611	496	429	467	616
BiCG	773	642	554	593	674
CGS	498	462	437	447	554
CG-A^TA	2060	1350	962	803	854

Table XI. Average results in work units based on three parameter sets in test problem 5; MILU(l) preconditioning

Method	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$
OM(5)	426	394	358	384	529
T-OMR(5)	404	380	375	436	542
BiCG	484	462	476	520	644
CGS	333	315	319	398	524
CG-A^TA	1550	1100	810	803	854

5. CONCLUSIONS

Incomplete LU factorization preconditioners for non-symmetric matrices with arbitrary sparsity patterns have been described and tested on 2D and 3D convection–diffusion problems. The MILU preconditioners were in general more efficient than the standard ILU preconditioners for regularly numbered grids in 2D, but the latter methods were superior to the former on irregularly numbered grids where the sparsity pattern was complex. Also in 3D problems ILU performed better than MILU. The dependence of the ILU preconditioner on the node numbering (or the matrix sparsity pattern) was found to be small. The convergence rate of ILU/MILU preconditioned conjugate gradient-like methods showed almost no sensitivity to grids with highly distorted elements. It was never observed that the ILU preconditioners had any destabilizing effect on iterative equation solvers. Very fast convergence of preconditioned conjugate gradient-like methods seems to be attainable in problems of hyperbolic nature in the whole domain.

For the ILU(l)/MILU(l) preconditioners in 2D the most efficient performance from a computational point of view was obtained with $l=1$ for MILU and with $l=2$ for ILU. Since the differences between different l values are relatively small and since the storage requirements of the preconditioning matrix increase with increasing l, the value $l=1$ or even $l=0$ may be recommended in practice. In 3D the decrease in sparsity with l led to an increase in the cost of the matrix–vector products which was too large to be compensated by the reduction in the total number of iterations. Therefore $l=0$ was the optimal choice in 3D problems.

It is observed that our simplest test problem, which is of the type often used in the literature, is readily handled by most iterative methods whether they are preconditioned or not. In the more complicated and physically relevant problems tested herein it seems that preconditioning is required for conjugate gradient-like methods to be of practical use. In contrast to methods without preconditioning, the preconditioned methods handle anisotropy with little reduction in efficiency. Our MILU(0) preconditioner led to a reduction of the total computational work of about 90% in the physically most relevant test problems. When the number of unknowns increases, the saving will be even greater as the preconditioned methods have more beneficial asymptotic work estimates than the non-preconditioned methods.

Since five different conjugate gradient-like methods were used in the tests, it is interesting to report their relative performance. None of the tested schemes diverged for any test problem. However, the non-preconditioned CG-A^TA sometimes converged extremely slowly. The preconditioned CG-A^TA required about the same amount of work as the *non*-preconditioned CGS. CGS turned out to be the best method, with competition from orthominres (k) and orthomin(k). The latter two methods were tested with the standard choice $k=5$, but lower k values increase their efficiency considerably. However, small values of k occasionally lead to divergence.

APPENDIX: ALGORITHMS

In this appendix we give detailed algorithms for the decomposition of **M** and the solution of equation systems with **LU** as coefficient matrix.

A common way to store a sparse matrix is store the non-zero entries in a one-dimensional array, say RA. To locate a special entry, two address arrays IR and IC are needed. $IR(k)$ gives the address in RA of the first non-zero entry in row number k in **A**. $IC(l)$ gives the column number (in **A**) of the entry $RA(l)$. IR has dimension $n+1$, while RA and IC have dimension $IR(n+1)-1$. In the algorithms below, **M** is assumed to be stored in RA (with length equal to the number of elements in P_l), with IC and IR as address arrays. The function $idx(r, s)$ gives the entry number in RA corresponding to the matrix entry $M_{r,s}$ if $M_{r,s} \neq 0$, otherwise the function value equals zero. The ILU factorization overwrites the array RA which initially contains the non-zero entries in **A** and possibly additional zeros according to the set P. After the elimination **L** will occupy the lower half part of **A**, while **U** will occupy the upper part. **L** has unit diagonal entries. The following algorithm computes the LU factorization of **M**:

```
for $r=1$ step 1 until $n-1$ do
begin
  $r_r := idx(r, r)$
  $d := RA(r_r)$
  for $q_1 = r_r + 1$ step 1 until $IR(r+1)-1$ do
  begin
    $i := IC(q_1)$
    $i_r := idx(i, r)$
    if $RA(i_r) \neq 0$ then
    begin
      $e := RA(i_r)/d$
      $RA(i_r) := e$
      $i_i := idx(i, i)$
      for $q_2 = r_r + 1$ step 1 until $IR(r+1)-1$ do
      begin
        $j := IC(q_2)$
        if $RA(q_2) \neq 0$ then
        begin
          $i_j := idx(i, j)$
          if $i_j \neq 0$ then
            $RA(i_j) := RA(i_j) - e \cdot RA(q_2)$
```

```
                else if $MILU$ then
                    $RA(i_i) := RA(i_i) - e \cdot RA(q_2)$
            end
        end
    end
    else $RA(i_r) := 0$
  end
end
```

The Boolean variable $MILU$ is false for standard incomplete LU factorization (ILU) and true for Gustafsson's[14] modified incomplete LU factorization (MILU).

The solution of $\mathbf{LUc} = \mathbf{d}$ is calculated by the following algorithm ($\mathbf{c}$ is stored in $\mathbf{d}$ at return):

```
for $i = 1$ step 1 until $n$ do
begin
  $s = 0$
  $i_i := IR(i)$
  while $IC(i_i) < i$ do
  begin
    $s := s + RA(i_i) \cdot \mathbf{d}(IC(i_i))$
    $i_i := i_i + 1$
  end
  $\mathbf{d}(i) := \mathbf{d}(i) - s$
end
$\mathbf{d}(n) := \mathbf{d}(n)/RA(idx(n, n))$
for $i = n - 1$ step $-1$ until 1 do
begin
  $s := 0$
  $i_i := idx(i, i) + 1$
  $i_d := i_i - 1$
  while $i_i \leqslant IR(i+1) - 1$ do
  begin
    $s := s + RA(i_i) \cdot \mathbf{d}(IC(i_i))$
    $i_i := i_i + 1$
  end
  $\mathbf{d}(i) := (\mathbf{d}(i) - s)/RA(i_d)$
end
```

Some preconditioned conjugate gradient-like methods require the solution of $\mathbf{M}^{\mathrm{T}}\mathbf{c} = \mathbf{d}$. The next algorithm overwrites $\mathbf{d}$ with $\mathbf{M}^{-\mathrm{T}}\mathbf{d}$. The algorithm is only valid if the relation $(l, m) \in P \Leftrightarrow (m, l) \in P$ holds, but this is fulfilled in most finite element and finite difference methods.

```
for $i = 1$ step 1 until $n$ do
begin
  $s := 0$
  $i_i := IR(i)$
  $i_d := i_i$
  while $IC(i_i) < i$ do
  begin
    $j := IC(i_i)$
    $j_i := idx(j, i)$
```

```
        $s := s + RA(j_i) \cdot \mathbf{d}(IC(i_i))$
        $i_i := i_i + 1$
    end
    $\mathbf{d}(i) := (\mathbf{d}(i) - s)/RA(i_d)$
end
for $i = n-1$ step $-1$ until 1 do
begin
    $s := 0$
    $i_i := idx(i, i) + 1$
    $i_d := i_i - 1$
    while $i_i \leqslant IR(i+1) - 1$ do
    begin
        $j := IC(i_i)$
        $j_i := idx(j, i)$
        $s := s + RA(j_i) \cdot \mathbf{d}(IC(i_i))$
        $i_i := i_i + 1$
        end
        $\mathbf{d}(i) := \mathbf{d}(i) - s$
end
```

REFERENCES

1. O. Axelsson and V. A. Barker, *Finite Element Solutions of Boundary Value Problems*, Academic Press, New York, 1984.
2. O. Axelsson, 'A survey of preconditioned iterative methods for linear systems of algebraic equations', *BIT*, **25**, 166–187 (1985).
3. P. Concus, G. H. Golub and G. Meurant, 'Block preconditioning for the conjugate gradient method', *SIAM J. Sci. Stat. Comput.*, **6**, 220–252 (1985).
4. J. R. Wallis, 'Incomplete gaussian elimination as a preconditioning for generalized conjugate gradient acceleration', in *Society of Petroleum Engineers of AIME, Proc. Seventh Symp. on Reservoir Simulation*, San Francisco, 1983.
5. H. P. Langtangen, T. Rusten, A. Tveito and S. Ø. Wille, 'An element by element preconditioner for iterative equation solvers', in *Proc. VI Int. Conf. on Finite Elements in Water Resources*, Lisboa, Portugal, 1986.
6. R. Fletcher, 'Conjugate gradient methods for indefinite systems', in G. A. Watson (ed.), *Dundee Biennal Conf. of Numerical Analysis*, Springer, New York, 1975, pp. 73–89.
7. M. R. Hestenes and E. Stiefel, 'Methods of conjugate gradients for solving linear systems', *J. Res. NBS*, **49**, 409–436 (1952).
8. P. Sonneveld, 'CGS, a fast Lanczos-type solver for nonsymmetric linear systems', *Report 84-16*, Department of Mathematics and Informatics, Delft University of Technology, 1984.
9. H. D. Simon, 'Incomplete LU preconditioners for conjugate-gradient-type iterative methods', in *Society of Petroleum Engineers of AIME, Proc. Eighth Symp on Reservoir Simulation*, Dallas, 1985.
10. G. A. Behie and P. A. Forsyth, 'Incomplete factorization methods for fully implicit simulation of enchanced oil recovery', *SIAM J. Sci. Stat. Comput.*, **5**, 543–561 (1984).
11. D. S. Kershaw, 'The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations', *J. Comput. Phys.*, **26**, 43–65 (1978).
12. J. A. Meijerink and H. A. van der Vorst, 'An iterative solution method for linear systems of which the coefficient matrix is a symmetric *M*-matrix', *Math. Comput.*, **31**, 148–162 (1977).
13. J. A. Meijerink, 'Guidelines for the usage of incomplete decomposition in solving sets of linear equations as they occur in practical problems', *J. Comput. Phys.*, **44**, 134–155 (1981).
14. I. Gustafsson, 'A class of first order factorizations', *BIT*, **18**, 142–156 (1978).
15. I. Gustafsson, 'Stability and rate of convergence of modified incomplete Cholesky factorization methods', *Computer Sciences 79.02 R*, Chalmers University of Technology, Gøteborg, Sweden, 1979.
16. H. C. Elman, 'A stability analysis of incomplete LU factorizations', *Math. Comput.*, **47**, 191–217 (1986).
17. M. A. Ajiz and A. Jennings, 'A robust incomplete Cholesky-conjugate gradient algorithm', *Int. j. numer. methods eng.*, **20**, 949–966 (1984).
18. O. Axelsson and N. Munksgaard, 'Analysis of incomplete factorizations with fixed storage allocation', in D. J. Evans (ed.), *Preconditioning Methods: Analysis and Applications*, 1983.
19. A. Brooks and T. Hughes, 'Streamline upwind/Petrov–Galerkin formulation for convection-dominated flows with particular emphasis on the incompressible Navier–Stokes equations', *Comput. Methods Appl. Mech. Eng.*, **32**, 199–259 (1982).