ELSEVIER

Comput. Methods Appl. Mech. Engrg. 190 (2001) 6533–6554

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics

Michele Benzi [a], Reijo Kouhia [b,*], Miroslav Tůma [c]

[a] *Department of Mathematics and Computer Science, Emory University, Atlanta, GA 30322, USA*
[b] *Laboratory of Structural Mechanics, Helsinki University of Technology, P.O. Box 2100, 02015 HUT, Finland*
[c] *Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic*

Received 8 November 2000; received in revised form 20 February 2001; accepted 20 February 2001

Abstract

The solution of linear systems arising in the finite element analysis of shells and solids by the preconditioned conjugate gradient method is considered. Stabilized and block versions of the AINV factorized approximate inverse preconditioner are presented and tested on a variety of difficult problems. Comparisons with other preconditioning methods are also included. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Preconditioning; Conjugate gradient; Factorized sparse approximate inverses; Block algorithms; Finite elements; Shells

1. Introduction

In this paper, we address the question of devising robust and efficient preconditioners for the conjugate gradient method applied to problems in solid and structural mechanics. Our main focus is on sparse linear systems of the type

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

arising from finite element models of thin shells, which are known to be especially challenging for iterative linear solvers. Problems from the modeling of 3-D solids will also be considered. The preconditioners examined in this paper are mainly sparse approximate inverses in factorized form; however, other techniques, like block Jacobi and point and block versions of incomplete Cholesky preconditioning, will also be considered.

A *sparse approximate inverse preconditioner* is a sparse matrix $\boldsymbol{M}$ that directly approximates the inverse of the coefficient matrix $\boldsymbol{A}$

$$\boldsymbol{M} \approx \boldsymbol{A}^{-1}.$$

The (left) preconditioned system, therefore, is of the form

$$\boldsymbol{M}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}\boldsymbol{b}.$$

[*] Corresponding author. Tel.: +9-451-3755; fax: +9-451-3826.
E-mail address: reijo.kouhia@hut.fi (R. Kouhia).

When $\boldsymbol{A}$ is symmetric positive definite and the preconditioned conjugate gradient (PCG) method is being used, the preconditioner must also be symmetric and positive definite. One way to insure this is to take $\boldsymbol{M}$ to be in factorized form, i.e.,

$$\boldsymbol{M} = \boldsymbol{Z}\boldsymbol{Z}^{\mathrm{T}}$$

for a nonsingular sparse matrix $\boldsymbol{Z}$. If $\boldsymbol{Z}$ is triangular, then nonsingularity is trivial to check, and the preconditioner can be regarded as an incomplete factorization of $\boldsymbol{A}^{-1}$. Thus, the preconditioned system takes the form

$$\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Z}\boldsymbol{u} = \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{b}, \tag{2}$$

and the solution to the original system (1) is given by $\boldsymbol{x} = \boldsymbol{Z}\boldsymbol{u}$. Note that the coefficient matrix in (2) is symmetric positive definite. In this paper, we consider exclusively approximate inverses in factorized form.

Several techniques have been proposed to compute factorized sparse approximate inverse preconditioners; see, e.g, [10,25,31,32,42] and the references therein. There are two major issues in computing a sparse approximate inverse: the first is the choice of an appropriate sparsity pattern for the approximate inverse factor $\boldsymbol{Z}$, and the second is the actual computation of the entries of $\boldsymbol{Z}$. Determining a good sparsity pattern for the approximate inverse can be difficult, especially for unstructured problems. Techniques for guessing a sparsity pattern a priori have been investigated, for instance, in [19]. Once a sparsity pattern has been found, least-squares techniques can be used to compute an approximate inverse with the given sparsity pattern [32]. Alternatively, the sparsity pattern can be computed dynamically, together with the nonzero entries, by applying a drop tolerance in the course of an inverse factorization algorithm. This is the approach taken in the AINV algorithm [10], and in this paper we will focus on variants of this technique.

Our interest in approximate inverse preconditioners stems in part from the fact that they are comparatively easy to implement on modern high-performance computers, particularly vector and parallel machines, since their application only involves matrix–vector products. This is in contrast with more traditional preconditioners, like incomplete factorization methods, which require highly sequential triangular solves and are difficult to implement efficiently on current parallel architectures, especially for unstructured problems. Another motivation for considering approximate inverse techniques instead of more traditional incomplete factorization preconditioners is related to the issue of numerical stability. In a previous paper [8], we ran into severe instability problems when trying to apply approximate inverse techniques to the stiffness matrices arising in thin shell analysis. Similar problems also plague the incomplete Cholesky methods, on occasions. In this paper we report on recent progress that was made towards ensuring the stable computation of factorized approximate inverse preconditioners for thin shell problems and other highly ill-conditioned matrices. We will argue that the new techniques are inherently more stable than standard incomplete factorization methods. Indeed, factorized approximate inverse preconditioners can be computed without the need for any kind of diagonal correction; this is not true, in general, for incomplete factorizations of $\boldsymbol{A}$.

Another contribution of this paper is to present block algorithms for both the basic and stabilized approximate inverse preconditioners, and to assess their performance on thin shell and solid mechanics problems. These problems have a natural block structure (stemming from the fact that there are several unknowns associated with each node), and exploiting this structure can result in increased robustness and improved convergence rates of the PCG iteration. Furthermore, dense blocks can be exploited to achieve reasonable performance on modern cache-based architectures.

The remainder of the paper is organized as follows. In Section 2 we review the basic AINV technique together with its stabilized variants. Block algorithms are described in Section 3. Section 4 is devoted to experimental results. We present some conclusions in Section 5.

2. A stable approximate inverse algorithm

In this section we recall the basic AINV technique [10] and its stabilized variants [7,30]. The AINV algorithm builds a factorized sparse approximate inverse of the form

$$\boldsymbol{M} = \boldsymbol{Z}\boldsymbol{D}^{-1}\boldsymbol{Z}^{\mathrm{T}} \approx \boldsymbol{A}^{-1}, \tag{3}$$

where $\boldsymbol{Z}$ is a sparse unit upper triangular matrix and $\boldsymbol{D}$ is diagonal.

The algorithm computes $\boldsymbol{Z}$ and $\boldsymbol{D}$ directly from $\boldsymbol{A}$ by means of an incomplete $\boldsymbol{A}$-orthogonalization process (that is, a Gram–Schmidt process with respect to the "energy" inner product $\langle \boldsymbol{x}, \boldsymbol{y}\rangle := \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{y}$) applied to the unit basis vectors $\{\boldsymbol{e}_i\}_{i=1}^{n}$. In this process, small elements are dropped to preserve sparsity. The resulting vectors $\{z_i\}_{i=1}^{n}$ are approximately conjugate with respect to $\boldsymbol{A}$, i.e., approximately $\boldsymbol{A}$-orthogonal. Letting $\boldsymbol{Z} = [z_1, z_2, \dots, z_n]$, then $\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Z}$ is approximately diagonal. Letting $\boldsymbol{D} = \mathrm{diag}(p_1, p_2, \dots, p_n)$ where $p_i = z_i^{\mathrm{T}}\boldsymbol{A}z_i$, it follows that $\boldsymbol{Z}\boldsymbol{D}^{-1}\boldsymbol{Z}^{\mathrm{T}} \approx \boldsymbol{A}^{-1}$. The approximate inverse factor $\boldsymbol{Z}$ is a sparse approximation of the inverse of the $\boldsymbol{L}^{\mathrm{T}}$ factor in the $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{T}}$ decomposition of $\boldsymbol{A}$. When the $\boldsymbol{A}$-orthogonalization process is performed exactly, the diagonal matrix $\boldsymbol{D}$ is the same in the two decompositions, and contains the *pivots* down the main diagonal.

The underlying assumption here is that many entries in $\boldsymbol{L}^{-1}$ are small in magnitude. This is true for many problems of practical interest, particularly for discretizations of boundary value problems for partial differential operators of elliptic type, the Green's function of which is known to exhibit fast decay in many cases. Sparsity can also be achieved by combining dropping of small entries with suitable sparse matrix orderings, such as nested dissection and minimum degree. These orderings are beneficial in that they result in smaller time and space requirements for forming and storing the preconditioner, while at the same time improving the quality of the preconditioner in a significant number of cases; see [11,14,25].

Denote by $\boldsymbol{a}_i^{\mathrm{T}}$ the ith row of $\boldsymbol{A}$. The $\boldsymbol{A}$-orthogonalization process for computing $\boldsymbol{Z}$ and $\boldsymbol{D}$ from $\boldsymbol{A}$ can be written in two mathematically equivalent ways. These are the *left-looking* and the *right-looking* algorithms, respectively. For brevity only the right-looking version, which is usually faster for matrices arising in finite element analysis, will be described here.

Algorithm 1 (*Right-looking A-orthogonalization algorithm*).

```
Let $z_i^{(0)} = e_i \ (1 \leqslant i \leqslant n)$
For $i = 1, 2, \dots, n$
   For $j = i, i+1, \dots, n$
      $p_j^{(i-1)} := a_i^{\mathrm{T}} z_j^{(i-1)}$
   End
   If $i < n$ Then
      For $j = i+1, \dots, n$
         $z_j^{(i)} := z_j^{(i-1)} - \left(\frac{p_j^{(i-1)}}{p_i^{(i-1)}}\right) z_i^{(i-1)}$
      End
   End If
End
```

This algorithm can be used to construct an AINV preconditioner by dropping entries below a prescribed drop tolerance in the z-vectors (corresponding to entries above the main diagonal of $\boldsymbol{Z}$) as they are computed. We distinguish between two kinds of dropping criteria, absolute and relative. In absolute dropping, a new entry in the $\boldsymbol{Z}$ matrix is dropped if it is less than a prescribed drop tolerance $\psi > 0$ times the largest entry in $\boldsymbol{A}$. With relative dropping, an entry is dropped if it is less than ψ times the infinity norm of the current row of $\boldsymbol{A}$. In either case, the dropping is applied after each update step for $z_j^{(i)}$ in Algorithm 1.

It is worth stressing that in order to have an efficient algorithm, the AINV process must be carefully implemented. Not only the sparsity in the rows of $\boldsymbol{A}$ and in the computed columns of $\boldsymbol{Z}$ must be exploited, but it is also important to make sure that of the inner products involving the rows of $\boldsymbol{A}$, only those that are structurally nonzero are actually computed; see, e.g., [15]. Then, for typical values of ψ and assuming an even distribution of nonzero entries in the incomplete inverse factor, all but a few of the innermost update loops in j can be skipped for each i, and the cost of computing the preconditioner scales linearly in the dimension n of the problem.

A crucial issue is that of the stability of the AINV process. For the preconditioner to be positive definite it is necessary that the pivots p_i be positive. In the absence of dropping, one has

$$p_i = \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{z}_i = \boldsymbol{z}_i^{\mathrm{T}} \boldsymbol{A} \boldsymbol{z}_i > 0,$$

since $\boldsymbol{A}$ is positive definite and $\boldsymbol{z}_i \neq \boldsymbol{0}$. However, when dropping is used the pivots may become nonpositive, forcing the algorithm to terminate. This is referred to as a *breakdown*. In [10], it was proved that the AINV process cannot suffer a breakdown if $\boldsymbol{A}$ is an H-matrix (see [5] for a definition of H-matrix), similar to incomplete Cholesky. If $\boldsymbol{A}$ is not an H-matrix, breakdowns are possible; unfortunately, stiffness matrices arising in finite element analysis are not H-matrices in general, and breakdowns are a common occurrence. For instance, in [8] we found that thin shell problems lead to massive breakdowns in the AINV algorithm. (Some incomplete factorization algorithms are also susceptible to break down on these problems.) Several diagonal compensation strategies have been proposed to deal with pivot breakdowns in the context of incomplete Cholesky preconditioning. Of these, the most popular in the structural engineering community is the Ajiz–Jennings variant of incomplete Cholesky [1]. These techniques can be readily extended to the case of factorized approximate inverses like AINV; however, the resulting preconditioner is often of poor quality. Fortunately, it is possible to prevent breakdowns without the need for any diagonal corrections, simply by formulating the $\boldsymbol{A}$-orthogonalization algorithm in a slightly different manner. This reformulation was recently developed, independently, in [7,30]; see also [13] for a somewhat different approach. We refer to this algorithm as the SAINV (for stabilized AINV) algorithm. In passing, we also note that there exist other algorithms for constructing factorized approximate inverse preconditioners for positive definite matrices which are guaranteed to be breakdown-free. These include the FSAI preconditioner [32] and the bordering algorithm described in [42].

SAINV is based on a reformulation of the $\boldsymbol{A}$-orthogonalization process, presented below, which in exact arithmetic and in the absence of dropping is mathematically equivalent to the previous ones. For brevity, we describe the right-looking variant only.

Algorithm 2 (*Right-looking* $\boldsymbol{A}$*-orthogonalization algorithm, version* 2).

```
Let $\boldsymbol{z}_i^{(0)} = \boldsymbol{e}_i \ (1 \leqslant i \leqslant n)$
For $i = 1, 2, \ldots, n$
    $\boldsymbol{v}_i = \boldsymbol{A}\boldsymbol{z}_i^{(i-1)}$
    For $j = i,\ i+1, \ldots, n$
        $p_j^{(i-1)} := \boldsymbol{v}_i^{\mathrm{T}} \boldsymbol{z}_j^{(i-1)}$
    End
    If $i < n$ Then
        For $j = i+1, \ldots, n$
            $\boldsymbol{z}_j^{(i)} := \boldsymbol{z}_j^{(i-1)} - \left( \frac{p_j^{(i-1)}}{p_i^{(i-1)}} \right) \boldsymbol{z}_i^{(i-1)}$
        End
    End If
End
```

It is easy to see [7] that when dropping in the $\boldsymbol{z}$-vectors is used in this algorithm, the pivots $p_i = p_i^{(i-1)}$ remain positive, and no breakdown is possible. In principle, breakdowns could still be possible because of round-off errors, but this is not likely to happen and we have never observed such an occurrence in actual computations. A more real danger is that of positive but very small pivots, which could potentially cause overflow and strong fill-in in the $\boldsymbol{z}$-vectors. In this case, some safeguarding might be necessary, such as pivot shifts and imposing a limit on the number of nonzeros allowed in a z-vector; however, we have not met any situation in actual computation where these safeguards were needed. We found SAINV to be a robust procedure in practice. With this preconditioner, we were able to solve a number of the problems that we could not solve with the standard AINV preconditioner; see the numerical tests in [7].

The price to pay for this robustness is a somewhat higher cost of computing the preconditioner. This increase, however, is usually modest, and the cost of computing the preconditioner remains reasonable – typically a linear function of the problem dimension n; see [7]. This is especially true if the preconditioner can be reused over different solves, for in this case the set-up cost would be easily amortized.

Although the stabilized approximate inverse algorithm is guaranteed to avoid breakdowns, the quality of the preconditioning as measured by the convergence rates of the conjugate gradient iteration may still be unsatisfactory, particularly for very ill-conditioned problems. The situation can sometimes be improved using block preconditioning, as described in the next section. Diagonal and block diagonal scalings are also recommended. If $\boldsymbol{D}$ denotes the main diagonal of $\boldsymbol{A}$, then the preconditioning is applied to the symmetrically (Jacobi) scaled matrix $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$. When the matrix has a natural block partitioning, a symmetric block Jacobi scaling is often beneficial. We will say more on scaling in Section 4.

3. Block algorithms

A standard technique to improve performance in dense matrix computations is to use blocking; see, e.g., [22]. By partitioning the matrices and vectors into blocks of suitable size (which usually depends on the target architecture) and by making such blocks the elementary entities on which the computations are performed, high-level BLAS can be used for cache efficiency on current architectures with a hierarchical memory structure. As a result of such fine-tuning, computational rates near the theoretical peak are possible for many dense linear algebra calculations.

In contrast, computations involving sparse matrices are much more difficult to optimize, particularly when the matrices are irregularly populated, and computational rates are typically only a small fraction of the theoretical peak. This is largely caused by the presence of indirect addressing in the innermost loops, e.g., in multiplying a sparse matrix by a dense vector. The situation can sometimes be improved by extending the use of blocking to sparse matrices.

Block iterative methods have been popular for many years in the solution of linear systems arising from the discretization of partial differential equations on structured grids; see, e.g., [5] or [38]. Here the blocks arise from some natural partitioning of the problem (grid lines, planes, or subdomains) and they are usually large and sparse. For structured grid problems in 2-D, efficient band solvers can be used to invert these blocks, leading to good performance in many cases.

A more interesting case for us is when the blocks are small and dense, as is the case when several variables are associated with a grid point, as with systems of partial differential equations. In this case high-level BLAS can be used as computational kernels and indirect addressing can be removed from the innermost loops, e.g., in the execution of matrix–vector products. For cache-based architectures, this leads to fairly good performance.

This block structure can either be naturally present in the matrix, or it must be imposed. As already mentioned, matrices with a natural block form often arise when the finite element method is used to discretize a partial differential equation or a system of them. In other cases there may be no natural block structure for us to exploit. However, we may still be able to use block algorithms by imposing a suitable blocking on the matrix, for instance by reordering the matrix using row and column permutations. A natural goal of the permutation should be to result in a block partitioned matrix with fairly dense blocks. To this end, any band/profile minimizing heuristic like reverse Cuthill–McKee (RCM) can be used; see [23] or [42]. Another possibility is to use the PABLO algorithm and its variants; see [41,18]. These schemes result in block partitioned forms in which the dense blocks tend to be clustered around the main diagonal of the matrix. In our case we opted for a block construction which provides a general distribution of the blocks in the matrix. The algorithm to construct the blocks is based on a graph compression procedure due to Ashcraft [4]. We will give now a brief overview of this approach.

The block construction is based on the properties of the underlying graph. The aim of the graph compression is to find *cliques* in the undirected graph of the matrix. Consider the graph $G = (V, E)$ of the symmetric matrix $\boldsymbol{A} = (a_{ij})$. $V = \{1, \dots, |V|\}$ is the set of vertices that correspond to rows and columns of the matrix. $E \subseteq V \times V$ is the set of its edges, where $(i, j) \in E$ if and only if the entry a_{ij} is nonzero. Then the adjacency set of a vertex v is

$$\mathrm{adj}(v) = \{u \mid (v, u) \in E\}.$$

Vertices $u \in V$ and $v \in V$ are adjacent if and only if $v \in \mathrm{adj}(u)$ (this is the same condition as $u \in \mathrm{adj}(v)$.) A *clique* in G is a set of vertices which are all mutually adjacent.

The task of finding blocks as large and dense as possible in a matrix $\boldsymbol{A}$ is equivalent to that of finding all cliques which are maximal with respect to inclusion in the graph G of the matrix $\boldsymbol{A}$. Of course, this task could be accomplished by brute force, comparing the adjacency sets of the vertices in G. However, this would result in an unnecessarily time-consuming procedure. In the paper [4], some basic enhancements of this procedure are described. Before actual computation and comparison of the adjacency sets take place, a preprocessing phase is performed. The candidates for inclusion into actual blocks are tested with respect to two easily computed additional quantities: vertex degree (size of an adjacency set) and vertex checksum (a simple function of adjacent vertices). These tests enable one to compute and compare adjacency sets in a few cases only. This contributes to the low cost of the graph compression routine.

The graph compression procedure was proposed as a tool for improving the performance of sparse direct solvers based on triangular factorizations of the system matrix. In this context, the compression contributes in a crucial way to reducing the symbolic overhead associated with the decomposition (reordering, elimination tree manipulation, symbolic factorization, etc.) As for the numeric phase of the factorization, it represents a useful complement to another important blocking strategy used in modern direct solvers, namely, the supernodal technique [35]. Note that the supernodal strategy is actually based on the structure of the triangular factors of $\boldsymbol{A}$, which are usually much less sparse than the system matrix itself. Because we are interested in block iterative algorithms, we focus on block strategies based on the sparsity of the original matrix only, like graph compression.

3.1. Block AINV preconditioning

Although the right-looking AINV algorithm is often faster, in this paper we restrict our attention to the left-looking block AINV scheme, which is easier to implement.

Assume the system matrix has been partitioned in the following block form:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \cdots & \boldsymbol{A}_{1N} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \cdots & \boldsymbol{A}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}_{N1} & \boldsymbol{A}_{N2} & \cdots & \boldsymbol{A}_{NN} \end{bmatrix}.$$

Here $\boldsymbol{A}_{ij}$ has order $n_i \times n_j$, where $1 \leqslant n_i \leqslant n$. N is the block dimension of the matrix, n is its dimension. Denote $\sum_{j<i} n_j$ by m_i (it is the offset of the ith block). Also, denote the block rows of $\boldsymbol{A}$ by $\boldsymbol{A}_i^{\mathrm{T}}$, $i = 1, \dots, N$. That is,

$$\boldsymbol{A}_i^{\mathrm{T}} = [\boldsymbol{A}_{i1}, \dots, \boldsymbol{A}_{iN}].$$

Note that the diagonal blocks $\boldsymbol{A}_{ii}$ are square symmetric positive definite matrices, and that $\boldsymbol{A}_{ji} = \boldsymbol{A}_{ij}^{\mathrm{T}}$ for $i \neq j$.

The block AINV algorithm computes block partitioned $\boldsymbol{Z}$ and $\boldsymbol{D}$ directly from $\boldsymbol{A}$ based on a block $\boldsymbol{A}$-orthogonalization process applied to blocks of columns of the identity matrix. Note that $\boldsymbol{D}$ is now block diagonal.

Let $\boldsymbol{E}_i$ denote the $n \times n_i$ matrix with all zero rows except for rows $m_i + 1$ through $m_i + n_i$ which correspond to the rows of the $n_i \times n_i$ identity matrix $\boldsymbol{I}_{n_i}$. The block $\boldsymbol{A}$-orthogonalization procedure can be written as follows.

Algorithm 3 (*Block $\boldsymbol{A}$-orthogonalization algorithm*).

```
Let $\boldsymbol{Z}_i^{(0)} = \boldsymbol{E}_i\ (1 \leqslant i \leqslant N)$; $\boldsymbol{P}_1^{(0)} = \boldsymbol{A}_{11}$
For $i = 2, \ldots, N$ do
  For $j = 1, \ldots, i-1$ do
    $\boldsymbol{P}_i^{(j-1)} := \boldsymbol{A}_j^{\mathrm{T}} \boldsymbol{Z}_i^{(j-1)}$
    $\boldsymbol{Z}_i^{(j)} = \boldsymbol{Z}_i^{(j-1)} - \boldsymbol{Z}_j^{(j-1)} [\boldsymbol{P}_j^{(j-1)}]^{-1} \boldsymbol{P}_i^{(j-1)}$
  End do
  $\boldsymbol{P}_i^{(i-1)} = \boldsymbol{A}_i^{\mathrm{T}} \boldsymbol{Z}_i^{(i-1)}$
End do
Let $\boldsymbol{Z}_i := \boldsymbol{Z}_i^{(i-1)}$ and $\boldsymbol{D}_i := \boldsymbol{P}_i^{(i-1)}$, for $1 \leqslant i \leqslant N$. Return
$\boldsymbol{Z} = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_N]$ and $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_N)$.
```

The block AINV preconditioner $\boldsymbol{M} = \boldsymbol{Z}\boldsymbol{D}^{-1}\boldsymbol{Z}^{\mathrm{T}} \approx \boldsymbol{A}^{-1}$ is computed by applying this block generalized Gram–Schmidt process incompletely. Blocks with small norms (corresponding to positions above the block diagonal part of the triangular matrix $\boldsymbol{Z}$) are dropped to preserve sparsity after the block updates for $\boldsymbol{Z}_i^{(j)}$. In our implementation, we used the infinity norm to gauge the size of the matrix blocks, but other choices are possible; see [15].

The inverses of the pivot blocks appearing in the update step for $\boldsymbol{Z}_i^{(j)}$ above are computed by means of a full triangular factorization. Notice that because of dropping, the pivot blocks in the incomplete process may not be positive definite, or even symmetric. In our implementation, we adopted the following safeguards. First, an $\boldsymbol{LU}$ factorization of $\boldsymbol{P}_j^{(j-1)}$ is computed (without pivoting). Then, the pivot block is symmetrized by replacing it, if necessary, with $\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$, where $\boldsymbol{L}$ is the lower triangular factor of $\boldsymbol{P}_j^{(j-1)}$. Pivot modifications may be used to enforce positive definiteness if needed. In practice, we found that such modifications are rarely needed, and the block AINV procedure is much more robust than the point AINV one.

Nevertheless, stabilization can and should be used to obtain a fully reliable algorithm. The block stabilized AINV algorithm is obtained from Algorithm 3 by replacing the expressions for $\boldsymbol{P}_i^{(j-1)}$ and $\boldsymbol{P}_i^{(i-1)}$ with $[\boldsymbol{Z}_j^{(j-1)}]^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Z}_i^{(j-1)}$ and $[\boldsymbol{Z}_i^{(i-1)}]^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Z}_i^{(i-1)}$, respectively. The pivot blocks are now guaranteed to be symmetric and positive definite, at the price of a somewhat higher construction cost.

Notice that the block AINV algorithms are rich in dense matrix–matrix operations, hence BLAS-3 kernels can be used to attain high performance.

In practice, we found that the performance of the block AINV schemes can be significantly enhanced by explicitly applying a symmetric block Jacobi scaling to the coefficient matrix $\boldsymbol{A}$ prior to performing the block AINV process. This preprocessing is based on the computation of the Cholesky factorizations $\boldsymbol{A}_{ii} = \boldsymbol{L}_i\boldsymbol{L}_i^{\mathrm{T}}$ for $1 \leqslant i \leqslant N$, followed by the explicit formation of the block scaled matrix

$$\hat{\boldsymbol{A}} = \boldsymbol{G}^{-1}\boldsymbol{A}\boldsymbol{G}^{-\mathrm{T}},$$

where $\boldsymbol{G} = \mathrm{diag}(\boldsymbol{L}_1, \ldots, \boldsymbol{L}_N)$. The additional costs incurred by this scaling are usually small compared to the total solution costs.

The use of blocking in the context of the AINV algorithm is not entirely new. Challacombe [16,17] has used a natural blocking of the problem to achieve high performance when applying the AINV transformation in quantum chemistry applications; however, the approximate inverse itself was computed using the point version of AINV (Algorithm 1). Bridson and Tang [15] developed a block variant of AINV which is also guaranteed not to break down; our approach differs from theirs in the blocking strategy, in the $\boldsymbol{A}$-orthogonalization scheme used and in the implementation, and appears to result in more robust preconditioning. Indeed, we are able to solve some problems for which they report failure in [15]. A different kind of block approximate inverse preconditioner, based on Frobenius norm minimization, has been recently described in [6]. Other recent references include [20,21,40] for the use of blocking within incomplete factorization preconditioning. It should be noticed that the effect of blocking is quite different for different preconditioners. For instance, for the case of the SPAI preconditioner, blocking has the effect of reducing the time to compute the preconditioner compared to the unblocked version, but at the cost of slower convergence rates [6]. As we will see in the next section, this is not usually the case for AINV. In all cases, however, blocking results in improved cache-efficiency on hierarchical memory computers.

4. Numerical experiments

In this section several numerical examples are presented. Special emphasis is given to problems arising in the modeling of thin shells, which are known to be hard for preconditioned iterative methods. Different parametric studies have been carried out. The effect of the regularizing parameter in the Hughes–Brezzi formulation on the quality of the preconditioner is investigated. The effects of point and block scaling and equation ordering are discussed. In addition, results for some 3-D solid mechanics problems are given.

The computations shown have been performed with a single processor on an SGI Origin 2000 (R12k processor) and on a Compaq AlphaServer GS140 (EV6 processor) at the Center for Scientific Computing, Espoo, Finland. In all the test runs, we used the zero vector as the initial guess and we stopped the iteration when the Euclidean norm of the initial residual had been reduced by five or ten orders of magnitude depending on whether realistic or artificial right-hand side vectors are used, respectively. Reported CPU timings are in seconds and correspond to the SGI Origin 2000 computations if not otherwise stated.

We also present results obtained with some incomplete Cholesky-type preconditioners, e.g., pointwise and block versions of symmetric ILUT [42] (denoted here by ICT) and of the robust Ajiz–Jennings incomplete factorization [1].

4.1. Shell examples

In this subsection, we report on results of computations for some commonly used shell test problems. In these models, the drilling rotation is accomplished by using the Hughes–Brezzi formulation [26] including an Allman-type displacement field [2]. The plate bending part of the element is based on the stabilized MITC theory [36] or the discrete-Kirchhoff concept. In the MITC formulation the stabilization parameter has been 0.4. The value of the regularizing penalty parameter γ used in the formulation of Hughes and Brezzi, unless otherwise stated, has been $\gamma = 10^{-3}G$, where G is the shear modulus. This value affects the condition number of the stiffness matrix and thus the convergence of the PCG iteration. The effect of the parameter γ on the spectral condition number as well as the convergence of the PCG iteration is demonstrated in [33] and will be further studied in the following examples.

4.1.1. On symmetric diagonal scaling and the effect of ordering

Diagonal scalings have a strong influence on the behavior of the approximate inverse preconditioning with (S)AINV. The reason is that the decay in the inverse factors can be significantly affected by symmetric Jacobi scaling. To illustrate this effect for the pointwise SAINV approach, we consider the analysis of a simple flat plate. For flat plates there is no coupling between the membrane and bending deformation and they depend on different nodal unknowns, thus by using a specific numbering of the unknowns, the global stiffness matrix can be assembled in a block form

$$\begin{bmatrix} \boldsymbol{A}_{\mathrm{m}} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{A}_{\mathrm{b}} \end{bmatrix},$$

where $\boldsymbol{A}_{\mathrm{m}}$ and $\boldsymbol{A}_{\mathrm{b}}$ are the global stiffness matrices associated with the membrane and the bending action, respectively.

The plate is discretized by a uniform quadrilateral 10×10 element mesh with four-node shell elements with drilling degrees of freedom (d.o.f.), resulting in 610 unknowns.

In Figs. 1 and 2 the sparsity patterns of the stabilized AINV preconditioner (incomplete inverse factor $\boldsymbol{Z}^{\mathrm{T}}$) with and without Jacobi scaling are shown. The drop tolerance is chosen in such a way that the density of the preconditioners in Figs. 1 and 2 will be approximately the same. It is clearly seen that without Jacobi scaling the dropping strategy will result in a preconditioner which almost totally ignores the bending deformational part of the system. In constructing these figures the absolute dropping criterion was used; however, similar patterns were also obtained with the relative criterion.

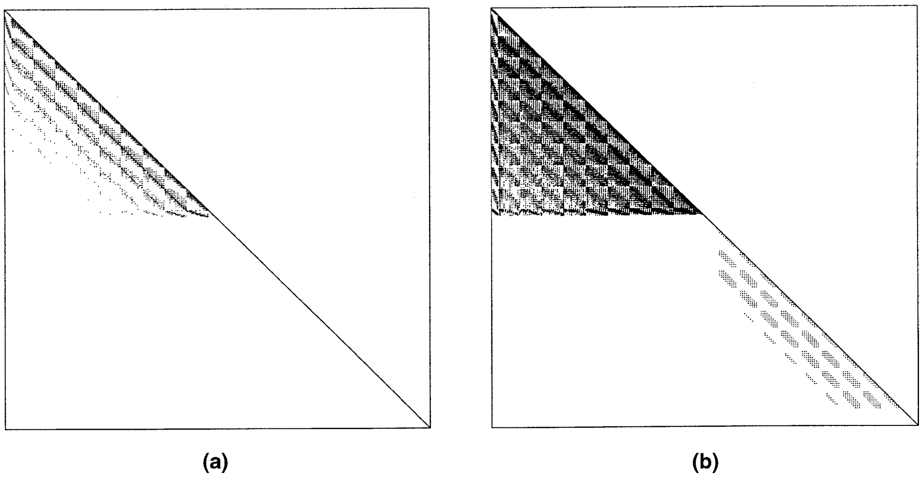


Fig. 1. Sparsity patterns of the SAINV preconditioner with two drop tolerances: (a) $\psi = 0.2 \max |a_{ij}|$, $nz(\boldsymbol{Z}^{\mathrm{T}}) = 3513$; (b) $\psi = 0.02 \max |a_{ij}|$, $nz(\boldsymbol{Z}^{\mathrm{T}}) = 20\,355$.

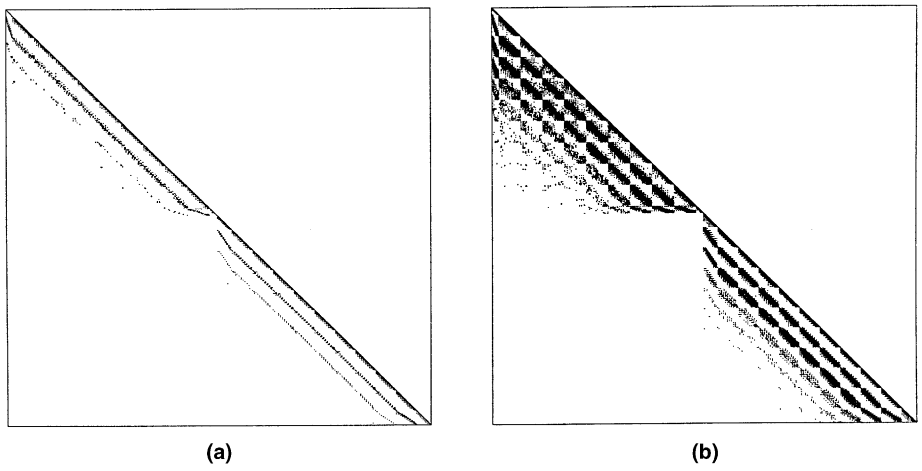


Fig. 2. Sparsity patterns of the SAINV preconditioner with two drop tolerances applied to the Jacobi scaled matrix: (a) $\psi = 0.2$, $nz(\boldsymbol{Z}^{\mathrm{T}}) = 3521$; (b) $\psi = 0.02$, $nz(\boldsymbol{Z}^{\mathrm{T}}) = 21\,623$.

In curved shells the coupling between membrane and bending deformations complicates the construction of optimal dropping strategies. However, the numerical experiments strongly suggest to apply Jacobi scaling when pointwise dropping is used in the preconditioner construction. [1]

To illustrate the effect of point and block scalings, we show some results from an analysis of a pinched cylindrical shell. In [8] the basic AINV strategy was tested for such matrices with modest results. As already mentioned, the main reason for the observed failures of the AINV-preconditioned conjugate gradient iteration in [8] was massive pivot breakdowns. If a static AINV preconditioner with the same sparsity pattern as that of $\boldsymbol{A}$, denoted by AINV(0), is applied to an a priori shifted matrix $\boldsymbol{A} + \alpha\,\mathrm{diag}(\boldsymbol{A})$, breakdowns can be avoided and the PCG iteration converges. However, the number of iterations can be rather high; see Table 1, where the matrix `S3RMQ4M1` is used as a test case with pinching load. This matrix can be obtained from the Matrix Market [39]. Furthermore, the use of a drop tolerance (instead of a static sparsity pattern) often resulted in instabilities and no PCG convergence [8].

Considerable improvements can be achieved with the stabilized pointwise algorithm SAINV with Jacobi scaling, as was shown in [7]. However, even better results are obtained with the block AINV method,

[1] The dropping strategy defined by Ajiz and Jennings [1] will implicitly do the same.

Table 1
Matrix S3RMQ4M1, standard AINV algorithm, MMD nodal ordering, $n = 5489$, $N = 961$, P-time = preconditioner set-up time, I-time = CG-iteration time, $P + I$ = total time for the solution of the linear system

Preconditioner	Scaling	ψ	ρ	Iterations	P-time	I-time	$P+I$	Notes
Static AINV(0)	–	–	1.00	1001	0.7	14.9	15.6	$\alpha = 0.01$
Block AINV	–	0.02	0.74	827	0.4	5.9	6.3	
	–	0.01	2.09	506	1.0	5.8	6.8	
	–	0.003	4.75	262	2.5	5.5	8.0	
Block AINV	Jacobi	0.25	1.06	672	0.6	5.0	5.6	
	Jacobi	0.15	2.53	370	1.2	4.8	6.0	
	Jacobi	0.075	4.53	286	2.4	6.7	9.1	
Block AINV	Block-Jacobi	0.1	1.21	408	0.7	3.3	4.0	
	Block-Jacobi	0.05	1.75	320	0.7	3.3	4.0	
	Block-Jacobi	0.02	3.73	189	1.8	3.1	4.9	

particularly with block Jacobi scaling. This is shown in Table 1, where ρ denotes the relative density of the preconditioner (i.e., the ratio of the number of nonzeros in the preconditioner and the number of nonzeros in the coefficient matrix) and ψ is the dropping tolerance. The point Jacobi scaling has little effect on the performance of the block AINV preconditioner, but the block scaling is very beneficial.

In Fig. 3 we show the effect of block Jacobi scaling on the performance of the block SAINV algorithm for another thin shell problem, S3RMT3M3, also available from the Matrix Market. This is an unstructured matrix corresponding to a graded mesh with 1666 triangles, 5357 d.o.f., and a spectral condition number $\mathrm{cond}_2(A) \approx 2.4 \times 10^{10}$. The block dimension is $N = 938$ resulting in the average block size of 5.7 ($= n/N$). Again, it is clear that block Jacobi scaling significantly improves the quality of the preconditioning.

The results above were obtained with a minimum degree ordering of the nodes. It is well known [24] that the numbering of unknowns may have significant influence on the performance of preconditioned iterations. For point versions of the AINV preconditioner, this has been investigated in [11,14]. Here we consider nodal orderings, corresponding to orderings of the block partitioned matrix or, equivalently, of the compressed graph. This is standard practice in structural mechanics. All the unknowns in a node are numbered consecutively before entering the next node. How to order the nodes depends on which kind of preconditioning will be used. Implicit incomplete factorization preconditioners tend to benefit from reorderings that try to minimize the root mean square (RMS) bandwidth, like RCM. On the other hand, factorized approximate inverse preconditioners do poorly with such orderings, and in [11,14] it was found that orderings like minimum degree or nested dissection, which result in short and wide elimination trees and sparse inverse factors, should be preferred.

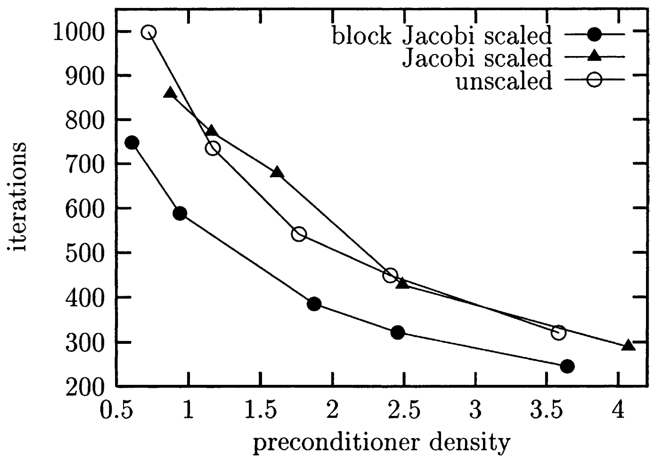


Fig. 3. Test case S3RMT3M3, effect of scaling for the block stabilized AINV.

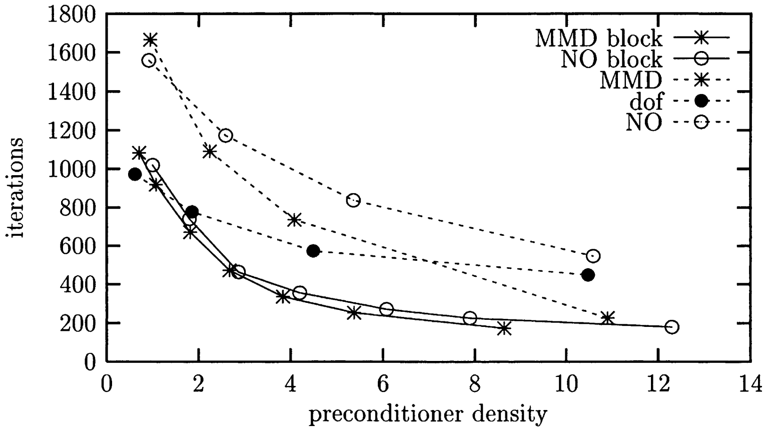


Fig. 4. Test case S3RMT3M3, comparison with pointwise stabilized AINV and the block stabilized AINV, different orderings.

Another possibility is to adopt an ordering by d.o.f. (d.o.f.-wise), where unknowns of similar type are ordered sequentially sweeping all the nodes in the mesh.

In Fig. 4 we show the iteration count as a function of the preconditioner density for two types of preconditioners (block SAINV and point SAINV) and different orderings for the shell problem S3RMT3M3. The ordering strategies are multiple minimum degree (MMD), generalized nested dissection (GND), multicoloring (MC) and quotient minimum degree (QMD), besides the original ordering (NO). GND, MC and QMD orderings give results which are almost identical to MMD ordering, thus they are not presented in Fig. 4. The block SAINV algorithm is scarcely sensitive to the ordering of the nodes. This is unlike point SAINV, for which we can see that the ordering makes a difference. Here, for reasonable preconditioner densities, the d.o.f.-wise ordering clearly outperforms the other orderings; however, the performance is not as good as with block SAINV. Experiments with other thin shell problems indicate that on average, MMD nodal ordering is slightly superior to the other orderings when block (S)AINV is being used.

4.1.2. On the value of the regularizing parameter in the Hughes–Brezzi formulation

In-plane rotational d.o.f., "drilling d.o.f.", are particularly convenient in the analysis of shells. Typical shell elements have three translational and two rotational d.o.f. at a node. This results in many difficulties of model construction and of programming and numerical ill-conditioning for certain types of element assemblages. Thus the presence of all three rotations at a node is advantageous from a practical point of view.

Early attempts to construct membrane elements with drill rotations were unsuccessful. Hughes and Brezzi [26] presented a simple variational formulation, which employs an independent rotation field and is also stable in the discrete case. They also proved that elements based on their formulation are convergent for all standard interpolations including equal order interpolation for displacements and rotation. Numerical experiments reported by Hughes et al. [27] confirm the a priori theoretical convergence estimates. They compared linear and quadratic triangular and quadrilateral elements and also bilinear elements with incompatible modes. Incompatible modes significantly improve the coarse mesh accuracy of the bilinear element. On the other hand, the static condensation needed to eliminate the nodeless generalized displacements is awkward, especially in nonlinear problems.

Ibrahimbegović et al. [29] amended the displacement interpolation of a four-node quadrilateral element by the Allman-type [2] quadratic modes in order to improve the coarse mesh accuracy of the element. This considerably improves the bending behavior of the element. They also added a hierarchical bubble interpolation mode to the displacement field.

The bilinear form of the variational equation of the Hughes–Brezzi formulation is

$$\int_{\Omega} \delta\varepsilon : \boldsymbol{C} : \varepsilon \, \mathrm{d}\Omega + \gamma \int_{\Omega} (\mathrm{skew}\nabla\delta\boldsymbol{u} - \delta\theta)(\mathrm{skew}\nabla\boldsymbol{u} - \theta) \, \mathrm{d}\Omega, \tag{4}$$

where $\boldsymbol{\varepsilon}$ is the strain tensor, i.e., the symmetric part of the displacement gradient, $\boldsymbol{\varepsilon} = \mathrm{symm}\nabla \boldsymbol{u} = \frac{1}{2}(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^{\mathrm{T}})$, θ a skew-symmetric tensor representing the in-plane rotation, $\mathrm{skew}\nabla \boldsymbol{u} = \frac{1}{2}(\nabla \boldsymbol{u} - (\nabla \boldsymbol{u})^{\mathrm{T}})$ and δ denotes the variation. The fourth order tensor $\boldsymbol{C}$ contains the material parameters.

An appropriate value of the regularizing penalty parameter γ is chosen in accordance with the ellipticity condition. For the isotropic case the value $\gamma = G$ (shear modulus) seems to balance the terms in the estimate and thus seems reasonable [28]. The method is insensitive to the choice of the penalty parameter in the region $0 < \gamma \leqslant G$ if discretization accuracy is considered. Since the above formulation is extensively studied in the papers [26–28], additional details are not repeated here.

However, little is known about the influence of the regularizing parameter on the conditioning of the stiffness matrix. In [33] the effect of the γ parameter on the condition number of a cylindrical shell modeled by a regular 30×30 quadrilateral element mesh (four-noded stabilized MITC type elements) was determined as well as the convergence of the IC(0)-PCG iteration. In the following, similar results are reported for a cylindrical shell with an irregular mesh modeled by three-node triangular elements. The effect of symmetric Jacobi scaling is also studied.

In Fig. 5 the effect of the γ parameter, symmetric Jacobi scaling and different drill-rotation formulations on the spectral condition number of the stiffness matrix are shown. The matrix corresponds to a pinched cylindrical shell discretized by means of an irregular mesh with 1666 triangular three-node stabilized MITC-type shell elements. Note that the unscaled case with $\gamma = 10^{-3}G$ and with Allman displacement field corresponds to the CYLSHELL matrix S3RMT3M3 in the Matrix Market.

When iterative methods are used, the recommended value for the γ parameter lies in the interval $10^{-6} < \gamma/G < 10^{-2}$, see Fig. 5 and Table 2, where the number of SAINV-PCG iterations is shown. Jacobi scaling gives an advantageous effect in lowering the spectral condition number by two orders of magnitude. Block Jacobi scaling gives consistently the best performance of the PCG iteration, although the difference in the condition number in comparison to the pointwise Jacobi scaling is not significant.

If an Allman-type displacement field is not used, it is usually recommended that the γ parameter should not be too small, owing to the fact that the condition number grows again if $\gamma < 10^{-6}G$. However, Jacobi or block Jacobi scaling seems to inhibit the growth of the condition number for small γ values.

4.1.3. Performance in shell problems

The performance of several preconditioning techniques in the case of the CYLSHELL matrix S3DKT3M2 [39] is shown in Table 3. It corresponds to a pinched cylindrical shell model ($t/R = 10^{-3}$) discretized by a uniform 150×100 mesh of triangular three-node discrete Kirchhoff-type elements (90 499

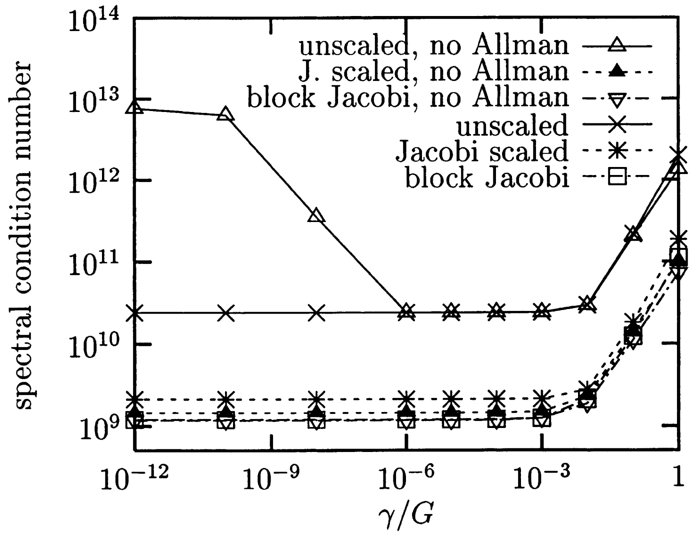


Fig. 5. Effect of the γ parameter, symmetric Jacobi scaling and different drill-rotation formulations on the spectral condition number. Pinched cylindrical shell, irregular mesh with 1666 triangular three-node stabilized MITC type shell elements, $n = 5357$, $N = 938$.

Table 2
Effect of the regularizing parameter on the convergence of block SAINV preconditioned iterations (MMD nodal ordering)[a]

γ/G	Block SAINV, unscaled			Block SAINV, J. scaled			Block SAINV, block J. scaled		
	ψ	ρ	Iterations	ψ	ρ	Iterations	ψ	ρ	Iterations
1	0.5	1.98	>10 000	0.5	1.95	5905	0.25	1.52	3969
10^{-1}	0.5	1.75	1892	0.5	1.47	1499	0.25	1.02	1535
10^{-3}	0.5	3.12	379	0.5	1.34	672	0.25	0.76	689
10^{-5}	0.5	2.76	500	0.5	1.33	688	0.25	0.76	673

[a] Irregular mesh with 1666 triangular three-node stabilized MITC type elements. The drill-formulation includes Allman-type displacement field.

Table 3
Cylindrical shell S3DKT3M2, $n = 90499$, $N = 15251$

	ψ	ρ	Iterations	P-time	I-time	$P+I$	Notes
Jacobi	–		>20 000	–	–	–	
Block Jacobi	–		8149	–	959	959	
IC(2)	–	1.83	1428	4	910	914	$\alpha = 2 \times 10^{-3}$
Block A–J	0.05	1.27	987	2	277	279	
	0.01	1.82	672	4	236	240	
	0.002	2.90	386	7	196	203	
Block ICT	0.1	0.75	3369	1	718	719	*lfil* = 10
	0.001	2.31	248	4	107	111	*lfil* = 10
Block SAINV	0.2	0.77	4338	5	911	915	
	0.15	1.17	3783	7	960	967	
	0.1	2.44	1903	15	782	797	
	0.05	4.01	1472	28	867	895	
	0.02	8.44	932	83	1050	1133	

unknowns). It is also the most ill-conditioned of the CYLSHELL matrices, having a spectral condition number $\approx 3.63 \times 10^{11}$.

In this particular problem, the incomplete Cholesky-type preconditioners perform rather well, partly due to the narrow band of the system matrix. RCM nodal ordering is used for IC-type methods (RMS bandwidth 497, maximum bandwidth 609) and MMD ordering for AINV. Although the block ICT preconditioner gives the fastest execution in this single-processor experiment, it is not easy to find a parameter pair (ψ, *lfil*) for which the decomposition exists without diagonal modifications, which usually destroy the convergence properties of the preconditioned iteration. (Here ψ denotes, as usual, the drop tolerance and *lfil* the maximum number of additional nonzero blocks allowed in each block column of the incomplete factor.) This is true for almost all shell problems we have tried to solve with the ICT preconditioner. Therefore this method cannot be considered robust for thin shell problems.

For this particular problem, block SAINV preconditioning is not competitive with the block version of the reliable Ajiz–Jennings IC preconditioner. For $\psi = 0.1$, however, it is faster than block Jacobi, so it may be useful in a parallel implementation.

The efficiency of the block methods is also studied by recording the Mflop rates for a sequence of cylindrical shell problems starting from a 30×30 element mesh (matrix S2RMT3M1) having 5489 unknowns to a fine discretization by 180×180 element mesh with 194 939 unknowns. Results are shown in Table 4. Here, B-A–J denotes the block Ajiz–Jennings incomplete Cholesky preconditioner. In all cases the number of nonzeros in the preconditioner is approximately the same as the number of nonzeros in the lower triangular part of the stiffness matrix, denoted by $nz(A)$.

The maximum performance of the SGI Origin 2000 R12k processor is 600 Mflops, thus for problems small enough to fit in the processor's external cache (8 Mb = one million double precision real numbers) we

Table 4
Cylindrical shell, Mflop rates

Mesh	n	$nz(\boldsymbol{A})$	Origin 2000[a]		AlphaServer GS140[b]	
			Block SAINV	Block A–J	Block SAINV	Block A–J
30×30[c]	5489	112 505	173	128	217	180
40×40	9719	201 605	160	128	134	110
50×50	15 149	316 505	139	109	101	90
60×60	21 779	457 205	101	92	81	78
70×70	29 609	623 705	83	80	76	75
80×80	38 639	816 005	77	73	75	70
90×90	49 136	1 034 105	73	67	75	65
100×100	60 299	1 278 005	71	67	74	62
140×140	118 019	2 511 605	65	57	68	53
180×180	194 939	4 158 005	65	62	67	53

[a] Compiled with -mips4 -64 -r12k -OPT: Olimit = 8000 -03 command.
[b] Compiled with -fkapargs = '-o = 5 -so = 3' command.
[c] Case `S2RMT3M1`.

can achieve over 25% of the processor's maximum performance. [2] For bigger problems we can see a rather rapid decay of performance with respect to flop rates. Interestingly, the flop rate for the preconditioner construction phase is scarcely affected by the dimension of the problem. For block SAINV on the origin, this was typically between 200 and 212 Mflops (between 260 and 310 Mflops on the AlphaServer), regardless of problem size.

A test case where block SAINV tends to outperform the incomplete Cholesky-type preconditioners, although it has a small bandwidth, is a simply supported T-beam problem from [34], where it has been used as a test example of interactive buckling phenomena. The structured mesh consists of 880 quadrilateral discrete-Kirchhoff type shell elements, resulting in 5563 unknowns. A strongly orthotropic layer is used to model the overlap between flange and web, having high bending rigidity in the transverse direction; see [34] for details. This leads to a highly ill-conditioned equation system, the spectral condition number of the stiffness matrix being $\approx 1.87 \times 10^{14}$.

Results are shown in Table 5. In this particular case the ICT preconditioner is completely useless. A stable preconditioner is obtained only with a very small drop tolerance and large *lfil*, resulting in nearly complete factorizations. The B-A–J preconditioner is stable, but it is clearly inferior to block SAINV.

4.2. Solid models

In this section some typical results with 3-D solid models are given. A standard displacement-based element formulation is used. In the standard FE models of 3-D solids there are three unknowns at each node, thus when the blocking is based only on the nodal unknowns, the maximum block size is three. However, the graph compression technique, explained in Section 3, results in larger blocks for quadratic and higher-order element meshes. In the subsequent solid model experiments the blocking is related only to a single node.

4.2.1. Surface mounted transistor

The first 3-D solid finite element model is from a thermal stress analysis of a surface mounted transistor, see Fig. 6. Only half of the transistor is modeled with a piece of a printing wiring board (PWB). The model consists of 1704 reduced triquadratic 20-node brick elements, resulting in 25 710 unknowns. The number of nonzeros in the lower triangular part of the matrix is $nz(\boldsymbol{A}) = 1\,889\,447$.

[2] For the Compaq AlphaServer GS140 EV6 processor, the external cache is 4 MB.

Table 5
T-beam, $n = 5563$, $N = 945$

ψ	ρ	Iterations
Block SAINV		
0.8	0.15	>10 000 = Block Jacobi
0.5	0.52	2851
0.4	0.76	>10 000
0.3	1.11	4943
0.2	1.66	699
0.1	3.11	948
0.05	4.57	550
0.02	6.89	191
Block A–J IC		
0.1	1.00	>10 000
0.01	2.10	5403
0.005	2.64	4405
0.001	4.99	1221
0.0005	5.46	840
0.0002	5.61	308

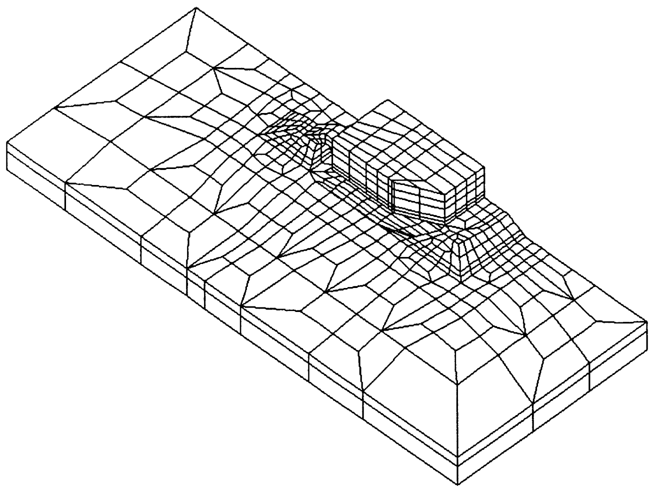


Fig. 6. Surface mounted transistor, 1704 triquadratic elements, 8744 nodes.

In this problem the standard IC and AINV approaches need shifting to behave well, and only block IC(0) succeeds without shifts. The ICT preconditioner was unsuccessful; without shifting the matrix, it is impossible to find a working pair of parameters ψ and *lfil*. Therefore only robust preconditioners, i.e., the Ajiz–Jennings IC and SAINV approaches are used in the comparison.

There is no big difference between nodal reorderings like MMD, QMD and GND, in terms of preconditioner density and number of iterations. The computing times are also comparable, thus only results with MMD reordering are shown in Table 6.

The Gibbs–King reordering gave the smallest RMS bandwidth (1231) and it was used with incomplete factorization-based preconditioners.

It should be mentioned that the iteration CPU-times for the pointwise SAINV can be reduced if the matrix is stored in the block format although the preconditioner is formed in a pointwise fashion. When using the block format one matrix–vector multiply takes 0.095 s while in the pointwise compressed storage by columns (CSC) format the time is 0.18 s, thus 25–36 s can be substracted from the iteration times of the pointwise SAINV results in Table 6. In this example the SAINV preconditioner results in similar convergence rates to the IC-based preconditioners although the preconditioner is only half as dense as the

Table 6
Surface mounted transistor, $n = 25\,710$, $nz = 1\,889\,447$, $N = 8744$

Preconditioner	ψ	ρ	Iterations	P-time	I-time	$P+I$
Block IC(0)	–	1.00	308	14	80	94
Block A–J	0.02	0.99	422	9	103	112
Block A–J	0.01	1.19	286	13	92	105
Block SAINV	0.2	0.20	917	34	111	145
Block SAINV	0.1	0.44	633	112	99	211
SAINV	0.03	0.43	379	26	99	125
SAINV	0.02	0.63	307	43	89	132
SAINV	0.01	1.17	218	96	85	181

IC-type ones. Hence, SAINV gives the fastest iteration time of the compared methods even in this single-processor experiment. This makes SAINV quite attractive, especially in a parallel environment.

4.2.2. Engine head

In Fig. 7 a finite element model of an engine head is shown. Unstructured discretization in 151 483 linear four-node tetrahedral elements results in 143 571 unknowns (number of nonzero elements in the lower triangular part of the stiffness matrix $nz(\boldsymbol{A}) = 2\,424\,822$). The loading used in the computations corresponds to a temperature increase by one degree centigrade.

This problem is an easy one to solve with preconditioned iterative methods. The standard AINV approach as well as IC preconditioners can be constructed without shifting. Nevertheless, the standard AINV performs worse than its stabilized variant, therefore such results are omitted. MMD reordering of nodes has been used in computations with the AINV algorithms, and the Gibbs–King profile reduction resulted in the smallest RMS bandwidth (2566), and it was used for the IC-type preconditioners.

It can be seen from the results in Table 7 that block SAINV gives slightly slower convergence rates in comparison to the pointwise version.

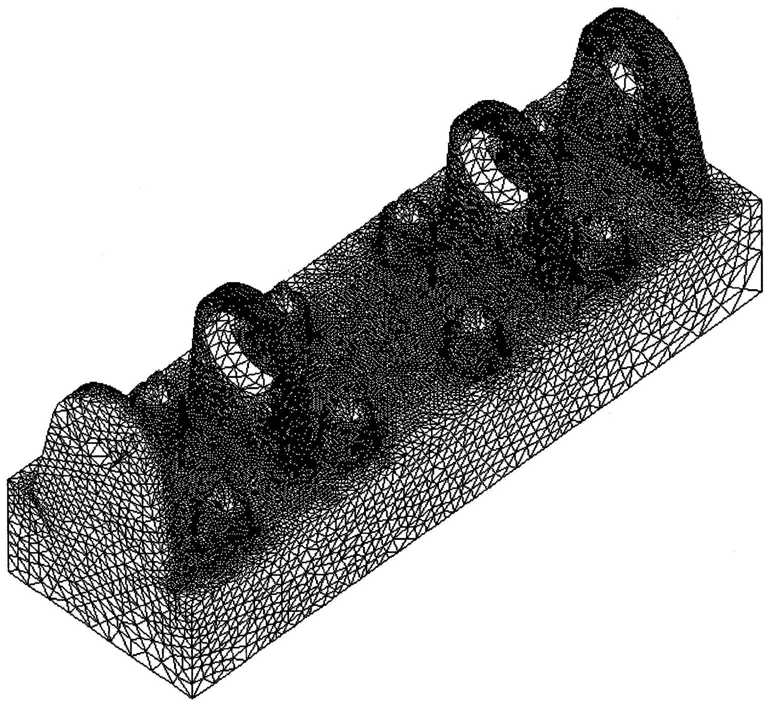


Fig. 7. Engine head, 151 483 linear tetrahedral elements, 48 989 nodes.

Table 7
Engine head, $n = 143\,571, nz = 2\,424\,822, N = 47\,857$

Preconditioner	ψ	ρ	Iterations	P-time	I-time	$P+I$	*lfil*
Right-looking SAINV preconditioners							
SAINV	0.1	0.29	568	15	226	241	
SAINV	0.05	0.63	427	26	209	235	
SAINV	0.025	1.24	375	51	236	287	
Left-looking SAINV preconditioners							
Block SAINV	0.2	0.48	695	14	218	232	
Block SAINV	0.1	1.02	485	33	202	235	
Incomplete factorization-based preconditioners							
IC(0)	–	1.00	312	1	162	163	
Block IC(0)	–	1.00	320	3	148	151	
Block A–J	0.02	1.38	275	7	129	136	
Block A–J	0.01	1.69	209	10	109	119	
Block ICT	0.05	0.81	560	3	206	209	10
Block ICT	0.01	1.34	211	6	90	96	10

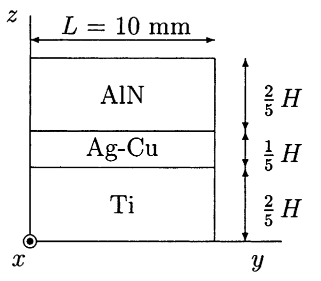
One matrix–vector multiply takes 0.25 or 0.14 s depending on whether the pointwise or block storage format is used, respectively. As explained in the previous example, 41–62 s could be saved from the pointwise SAINV iteration times if the matrix were stored in the block format.

4.2.3. Sensitivity to element aspect ratio

Current approaches for analyzing complex laminated structures with solid elements will usually result in element gometries where one dimension is small compared with the others. The convergence rate of the PCG iteration deteriorates with increasing aspect ratio.

An elastic block composed by three material layers and occupying the region (in cartesian coordinates) $0 < x,\ y < L$ and $0 < z < H$ is considered. The material interfaces are horizontal layers parallel to the xy-plane, and having positions $z = (2/5)H$ and $z = (3/5)H$. The stack models a ceramic (AlN) to metal (Ti) joint brazed together with Ag–Cu filler alloy. The constitutive parameters, Young's modulus E, Poisson's ratio ν and thermal elongation α, used for these materials are given in Fig. 8. A uniform $20 \times 20 \times 20$ element mesh with eight-node trilinear brick elements is used. Loading is temperature change defined by $\Delta T = \Delta T_0 xyz/L^3$. Minimal constraints which prevent the rigid body motion are imposed.

Computations are carried out with three different aspect ratios L/H (Table 8). As usual, the convergence of the preconditioned iterations deteriorates with increasing aspect ratio distortions. The block preconditioner suffers more severely from high aspect ratios than the pointwise one. Only if the element aspect ratios are in a feasible range, say $L/H < 10$, the performance of the block preconditioner is reasonable. This is exactly the opposite of what is observed in thin shell applications, where the block preconditioners are more robust also with respect to mesh distorsions. Hence, the relative behavior of block and pointwise versions of SAINV can be quite different depending on whether they are applied to shells or to solids.



	Ti	Ag-Cu	AlN
E [GPa]	118.6	94.4	315
ν	0.36	0.36	0.25
$10^6\alpha$	6.441	17.995	2.85

Fig. 8. Three-material solid block: geometry and material data.

Table 8
Trimetallic block

L/H	SAINV			Block SAINV		
	ψ	ρ	Iterations	ψ	ρ	Iterations
1	0.025	0.48	185	0.1	0.46	244
10	0.025	0.75	493	0.1	0.82	745
100	0.025	0.71	3036	0.1	1.03	9987

4.3. On the use of artificial right-hand sides

A frequent problem when testing iterative solvers is the difficulty in procuring physically realistic right-hand side vectors $\boldsymbol{b}$. Especially when using matrices from sparse matrix collections, such as the Matrix Market [39], corresponding right-hand sides may not be provided nor there are any hints for the construction of such vectors. As a consequence, developers of linear solvers are often forced to use artificially constructed right-hand sides, having no relation to the problem from which the coefficient matrix arises, and devoided of physical meaning. A typical choice is to compute $\boldsymbol{b}$ as the product of $\boldsymbol{A}$ times a given vector $\boldsymbol{x}$. In this way, the exact solution is known and the accuracy of the computed solution can be checked. Here we would like to emphasize the pitfalls involved in the use of artificial right-hand sides when testing iterative methods.

The convergence histories in Fig. 9 illustrate the fact that when using an artificial right-hand side, declaration of convergence based on the relative residual norm can be completely misleading. As it can be seen from the figure, the relative residual drops rapidly to the level of 10^{-4}–10^{-5}, even though there is practically no progress in the accuracy of the solution. However, if a physically realistic load vector is used, the relative residual behaves like the true solution error in Fig. 9. Nevertheless, residual tolerances can still lead to inaccurate results in some cases; more reliable stopping criteria have been proposed, e.g., in [3,37].

To understand this phenomenon, the stiffness matrix $\boldsymbol{A}$ and the displacement vector $\boldsymbol{x}$ are decomposed into parts describing membrane deformations $\boldsymbol{A}_{\mathrm{m}}, \boldsymbol{x}_{\mathrm{m}}$ and bending (and possibly transverse shear) $\boldsymbol{A}_{\mathrm{b}}, \boldsymbol{x}_{\mathrm{b}}$, which are orthogonal in the sense that $\boldsymbol{A}_{\mathrm{m}}\boldsymbol{x}_{\mathrm{b}} = \boldsymbol{A}_{\mathrm{b}}\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{0}$. The residual at a given iterate i is

$$\boldsymbol{r}_i = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_i) = \boldsymbol{A}\boldsymbol{e}_i,$$

where $\boldsymbol{x}$ is the true solution and $\boldsymbol{e}$ is the error, which also can be decomposed as $\boldsymbol{e}_{\mathrm{m}}, \boldsymbol{e}_{\mathrm{b}}$, corresponding to membrane and bending deformations, respectively. It is easy to see that the error components are also orthogonal in a way similar to the total displacement vector

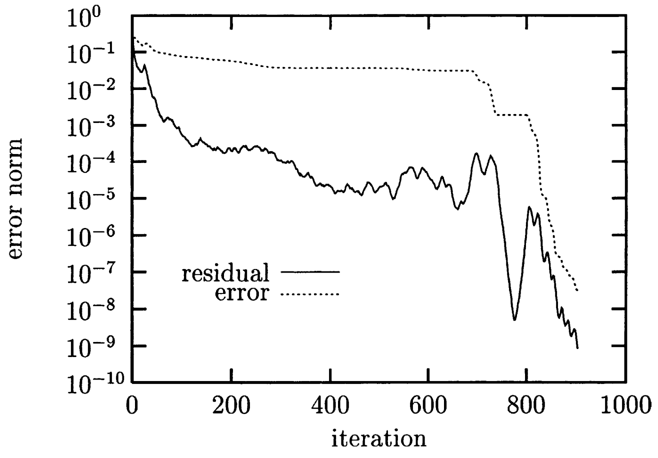


Fig. 9. Relative Euclidean norms for CG residual and the true solution error when artificial rhs-vector is used. Pinched cylindrical test case S3RMT3M3.

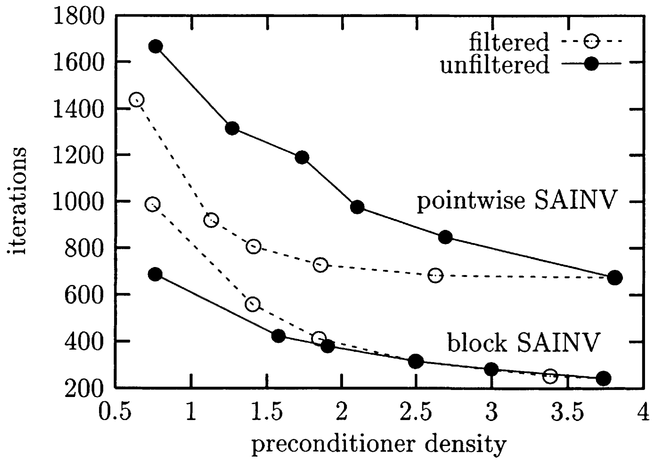


Fig. 10. Effect of a posteriori filtration on behavior of SAINV preconditioned CG. Test case S3RMT3M3.

$$r_i = (A_\mathrm{m} + A_\mathrm{b})(e_{\mathrm{m},i} + e_{\mathrm{b},i}) = A_\mathrm{m}e_{\mathrm{m},i} + A_\mathrm{b}e_{\mathrm{b},i} = r_{\mathrm{m},i} + r_{\mathrm{b},i},$$

thus the residual norm satisfies

$$\|r_i\| = \|r_{\mathrm{m},i} + r_{\mathrm{b},i}\| \leqslant \|A_\mathrm{m}e_{\mathrm{m},i}\| + \|A_\mathrm{b}e_{\mathrm{b},i}\| \leqslant \|A_\mathrm{m}\|\|e_{\mathrm{m},i}\| + \|A_\mathrm{b}\|\|e_{\mathrm{b},i}\| \leqslant C_1\left(\frac{t}{L}\right)\|e_{\mathrm{m},i}\| + C_2\left(\frac{t}{L}\right)^3\|e_{\mathrm{b},i}\|,$$

where C_1, C_2 are constants of the same order, depending on material parameters but not on the thickness t of the shell (L is some characteristic length of the shell). If the load vector is determined by assuming the components of the displacement solution vector to be all ones, it will result in a highly membrane dominated case ($x_\mathrm{m}^\mathrm{T}A_\mathrm{m}x_\mathrm{m}/x_\mathrm{b}^\mathrm{T}A_\mathrm{b}x_\mathrm{b} \gg 1$) and disproportionately large residuals in the membrane deformations, $\|r_{\mathrm{m},i}\|/\|r_{\mathrm{b},i}\| \gg 1$, even though the solution error norms initially are of the same order. Since the membrane solution converges much quicker, the residual norm can show reduction of order $(L/t)^2$ ($\sim 10^6$ in the example shown in Fig. 9) even though there is practically no improvement in the bending solution, which in many realistic cases is the dominant deformation mode.

4.4. On the use of post-filtration

Dropping selected entries in the computed approximate inverse factor Z, usually referred to as a posteriori filtration, may result in considerable savings and a more efficient solution strategy. The idea is to reject entries that contribute little to the quality of the preconditioner, thereby reducing the cost of applying the preconditioner without too much affecting the rate of convergence of the PCG iteration. A simple criterion is to drop entries below a prescribed drop tolerance. A posteriori filtration has been shown to improve the efficiency of the FSAI preconditioned iteration in [31]. Here we consider post-filtration for the SAINV preconditioner. Because SAINV, unlike FSAI, is already computed using a drop tolerance, the usefulness of applying post-filtration is not immediately obvious. [3]

Effects of post-filtration on point and block SAINV preconditioning are shown in Fig. 10 for the thin shell test example S3RMT3M3. The solid markers represent computations with unfiltered preconditioners while the open circles correspond to filtered cases obtained from the most dense preconditioners of the unfiltered case. Drop tolerances for the unfiltered case (solid markers) are 0.1, 0.05, 0.04, 0.03, 0.025, 0.02

[3] In the left-looking version two different dropping strategies can be utilized. In the standard approach small entries are dropped during the computation of $Z_i^{(j)}$, see Algorithm 3. Another approach delays the dropping until the whole column is computed, i.e., after the j-loop is completed in Algorithm 3. Rather limited testing showed the latter approach to result in slightly better quality preconditioners. However, the additional cost in the preconditioner construction was greater than the gains resulting from better convergence rates.

Table 9
Surface mounted transistor, effect of post-filtration

Preconditioner	ψ	ψ_{pf}[a]	ρ[b]	Iterations	P-time	I-time	$P+I$
SAINV	0.01	0.05	0.53	244	96	67	163
SAINV	0.02	0.07	0.33	341	43	82	125
SAINV	0.02	0.1	0.23	375	43	88	131

[a] Post filtration tolerance.
[b] Density of the preconditioner after post-filtration.

for block SAINV and 0.06, 0.035, 0.025, 0.02, 0.015, 0.01 for pointwise SAINV. The filtration tolerances are 0.05, 0.1, 0.15, 0.2, 0.3 for block SAINV and 0.06, 0.1, 0.15, 0.2 and 0.4 for pointwise SAINV. It can be seen that pointwise SAINV retains its convergence properties even though 50% of its elements are dropped. However, block SAINV is scarcely affected by filtering.

In 3-D solid examples the behavior appears to be somewhat different. For the engine head problem there is hardly any improvement in iteration times when post-filtration is used.

Slight improvements are obtained for the surface mounted transistor case. Here the increase in iteration counts is compensated by the reduced cost of each iteration; compare Tables 6 and 9.

5. Conclusions

While direct solution techniques, like skyline or multifrontal solvers, are still widely preferred for solid and structural mechanics problems, preconditioned iterative methods are rapidly gaining in popularity. As computer technology advances, the use of increasingly detailed and sophisticated models results in problems of ever growing size. While direct solvers are very effective for problems of moderate size, the use of iterative methods becomes unavoidable for very large systems. Indeed, the storage requirements for large problems make direct methods unfeasible.

Iterative methods require less storage than direct ones in most cases and can be more easily implemented on parallel architectures. Unfortunately, iterative solvers still lack the reliability of direct methods, and they can fail on challenging problems. Models of thin shells typically result in very ill-conditioned linear systems which are difficult to solve with iterative methods.

In order to increase the reliability of iterative solvers, more robust preconditioning techniques are needed. In this paper we have introduced some new preconditioning techniques for the conjugate gradient method and examined their performance on difficult shell and solid mechanics problems. Besides the SAINV preconditioner, which was first described in [7,30], we have developed new block variants of AINV and SAINV, as well as block versions of various incomplete Cholesky preconditioners, including ones based on drop tolerances and the stabilized variant of Ajiz and Jennings. Of these methods, the pointwise and block versions of SAINV are the only ones that do not require any diagonal modifications, and are always well defined.

For the block methods, the block structure naturally comes from the finite element formulation. Exploitation of this block structure improves robustness and performance on current cache-based architectures.

The importance of point and block diagonal scalings and the effect of different (block) orderings were investigated. In general, scalings and reorderings are found to significantly increase the robustness and performance of the preconditioning. For shell problems, we found that scalings also reduce the influence on the condition number of parameters like the regularizing parameter γ in the Hughes–Brezzi formulation.

Concerning the performance of approximate inverse techniques, we found that the block methods performed better overall than the pointwise ones on thin shell models, where the block size is typically six. On the other hand, for problems in solid mechanics, where the typical block size is only three, we found that the block methods, while reliable, were no better than the pointwise SAINV algorithm. We also found the block methods to be more sensitive to bad aspect ratios than the point ones, exactly the opposite than for the shell models.

Our experiments indicate that while incomplete Cholesky-type methods are often faster than approximate inverse techniques, it may be difficult to find parameter values that will result in a stable incomplete factorization. As for the Ajiz–Jennings robust IC factorization, the diagonal modifications needed to stabilize the decomposition can lead to slow convergence in some cases (as in the T-beam example). In contrast, robust approximate inverse preconditioners like SAINV and block SAINV cannot break down, do not need any diagonal modifications, and often result in convergence rates that are comparable with those for incomplete Cholesky-type preconditioners or even better. Thus, approximate inverse techniques can be a useful alternative to the popular incomplete Cholesky-type preconditioners, even on serial computers. However, they are expected to be especially useful on parallel computers, since their application only requires matrix–vector products.

Parallel implementations of the basic AINV algorithm have been described in [9,12], with good results on a range of scalar PDE problems from the modeling of diffusion and transport phenomena on both structured and unstructured grids. Future work will focus on developing parallel implementations of the SAINV and block SAINV preconditioners for large-scale problems in solid and structural mechanics.

Acknowledgements

The second author gratefully acknowledges the hospitality of the Institute of Computer Science of the Academy of Sciences of the Czech Republic where part of this work was performed. The work of the third author was supported by Grant Agency of the Academy of Sciences of the Czech Republic grant no. 2030801 and Grant Agency of the Czech Republic grant no. 201/00/0080. Part of this work was done during a visit to the Laboratory of Structural Mechanics at Helsinki University of Technology: the hospitality of the Laboratory is greatly appreciated.

References

[1] M.A. Ajiz, A. Jennings, A robust incomplete Cholesky conjugate gradient algorithm, Int. J. Numer. Methods Engrg. 20 (1984) 949–966.

[2] D.J. Allman, A compatible triangular element including vertex rotations for plane elasticity analysis, Comput. Struct. 19 (1984) 1–8.

[3] M. Arioli, I.S. Duff, D. Ruiz, Stopping criteria for iterative solvers, SIAM J. Matrix Anal. Appl. 13 (1992) 138–144.

[4] C. Ashcraft, Compressed graphs and the minimum degree algorithm, SIAM J. Sci. Comput. 16 (1995) 1404–1411.

[5] O. Axelsson, Iterative Solution Methods, Cambridge University Press, Cambridge, MA, 1994.

[6] S.T. Barnard, M.J. Grote, A block version of the SPAI preconditioner, in: B.A. Hendrickson et al. (Eds.), Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia, PA, 1999 [CD-ROM].

[7] M. Benzi, J.K. Cullum, M. Tůma, Robust approximate inverse preconditioning for the conjugate gradient method, SIAM J. Sci. Comput. 22 (2000) 1318–1332.

[8] M. Benzi, R. Kouhia, M. Tůma, An assessment of some preconditioning techniques in shell problems, Comm. Numer. Methods Engrg. 14 (1998) 897–906.

[9] M. Benzi, J. Marín, M. Tůma, A two-level parallel preconditioner based on sparse approximate inverses, in: D.R. Kincaid, A.C. Elster, (Eds.), Iterative Methods in Scientific Computation IV, IMACS Series in Computational and Applied Mathematics, vol. 5, IMACS, New Brunswick, NJ, 1999, pp. 167–178.

[10] M. Benzi, C.D. Meyer, M. Tůma, A sparse approximate inverse preconditioner for the conjugate gradient method, SIAM J. Sci. Comput. 17 (1996) 1135–1149.

[11] M. Benzi, M. Tůma, Orderings for factorized sparse approximate inverse preconditioners, SIAM J. Sci. Comput. 21 (2000) 1851–1868.

[12] L. Bergamaschi, G. Pini, F. Sartoretto, Approximate inverse preconditioning in the parallel solution of sparse eigenproblems, Numer. Linear Algebra Appl. 7 (2000) 99–116.

[13] R. Bridson, Multi-resolution approximate inverses, M.Sc. thesis, University of Waterloo, Ontario, Canada, 1999.

[14] R. Bridson, W.-P. Tang, Ordering, anisotropy, and factored sparse approximate inverses, SIAM J. Sci. Comput. 21 (1999) 867–882.

[15] R. Bridson, W.-P. Tang, Refining an approximate inverse, J. Comput. Appl. Math. 123 (2000) 293–306.

[16] M. Challacombe, A simplified density matrix minimization for linear scaling self-consistent field theory, J. Chem. Phys. 110 (1999) 2332–2342.

[17] M. Challacombe, A general parallel sparse-blocked matrix multiply for linear scaling SCF theory, Comp. Phys. Comm. 128 (2000) 93–107.

[18] H. Choi, D.B. Szyld, Application of threshold partitioning of sparse matrices to Markov chains, in: Proceedings of IEEE International Computer Performance and Dependability Symposium, IPDS'96, Urbana-Champaign, IN, September 4–6, 1996, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 158–165.

[19] E. Chow, A priori sparsity patterns for parallel approximate inverse preconditioners, SIAM J. Sci. Comput. 21 (2000) 1804–1822.

[20] E. Chow, M.A. Heroux, An object-oriented framework for block preconditioning, ACM Trans. Math. Software 24 (1998) 159–183.

[21] E. Chow, Y. Saad, Approximate inverse techniques for block-partitioned matrices, SIAM J. Sci. Comput. 18 (1997) 1657–1675.

[22] J.J. Dongarra, I.S. Duff, D.C. Sorensen, H.A. van der Vorst, Numerical Linear Algebra for High-Performance Computers, SIAM, Philadelphia, PA, 1998.

[23] I.S. Duff, A.M. Erisman, J.K. Reid, Direct Methods for Sparse Matrices, Oxford University Press, Oxford, 1986.

[24] I.S. Duff, G. Meurant, The effect of ordering on preconditioned conjugate gradients, BIT 29 (1989) 635–657.

[25] M.R. Field, Improving the performance of factorised sparse approximate inverse preconditioner, Technical Report HDL-TR-98-199, Hitachi Dublin Laboratory, Dublin, Ireland, 1998.

[26] T.J.R. Hughes, F. Brezzi, On drilling degrees of freedom, Comput. Methods Appl. Mech. Engrg. 72 (1989) 105–121.

[27] T.J.R. Hughes, F. Brezzi, A. Masud, I. Harari, Finite elements with drilling degrees of freedom: theory and numerical evaluations, in: R. Gruber, J. Periaux, R.P. Shaw (Eds.), Proceedings of the Fifth International Symposium on Numerical Methods in Engineering, Springer, Berlin, 1989, pp. 3–17.

[28] T.J.R. Hughes, A. Masud, I. Harari, Numerical assessment of some membrane elements with drilling degrees of freedom, Comput. Struct. 55 (1995) 297–314.

[29] A. Ibrahimbegović, R.L. Taylor, E.L. Wilson, A robust quadrilateral membrane finite element with drilling degrees of freedom, Int. J. Numer. Methods Engrg. 30 (1990) 445–457.

[30] S.A. Kharchenko, L.Yu. Kolotilina, A.A. Nikishin, A.Yu. Yeremin, A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form, Numer. Linear Algebra Appl. 8 (2001) 165–179.

[31] L.Yu. Kolotilina, A.A. Nikishin, A.Yu. Yeremin, Factorized sparse approximate inverse preconditioning IV: simple approaches to rising efficiency, Numer. Linear Algebra Appl. 6 (1999) 515–531.

[32] L.Yu. Kolotilina, A.Yu. Yeremin, Factorized sparse approximate inverse preconditioning I: Theory, SIAM J. Matrix Anal. Appl. 14 (1993) 45–58.

[33] R. Kouhia, Using iterative solvers in non-linear continuation algorithm, in: J. Aalto, T. Salmi (Eds.), Proceedings of the 6th Finnish Mechanics Days, 1997, pp. 253–267.

[34] R. Kouhia, M. Mikkola, Tracing the equilibrium path beyond compound critical points, Int. J. Numer. Methods Engrg. 46 (1999) 1049–1074.

[35] J.W.H. Liu, E.G. Ng, B.W. Peyton, On finding supernodes for sparse matrix computations, SIAM J. Matrix Anal. Appl. 14 (1993) 242–252.

[36] M. Lyly, R. Stenberg, T. Vihinen, A stable bilinear element for the Reissner–Mindlin plate model, Comput. Methods Appl. Mech. Engrg. 110 (1993) 343–357.

[37] G. Meurant, Numerical experiments in computing bounds for the norm of the error in the preconditioned conjugate gradient method, Numer. Algorithms 22 (1999) 353–365.

[38] G. Meurant, Computer Solution of Large Linear Systems, North-Holland, Amsterdam, 1999.

[39] National Institute of Standards, Matrix Market, available online at http://math.nist.gov/MatrixMarket.

[40] E. Ng, B.W. Peyton, P. Raghavan, A blocked incomplete Cholesky preconditioner for hierarchical-memory computers, in: D.R. Kincaid, A.C. Elster (Eds.), Iterative Methods in Scientific Computation IV, IMACS Series in Computational and Applied Mathematics, vol. 5, 1999, pp. 211–221.

[41] J. O'Neil, D.B. Szyld, A block ordering method for sparse matrices, SIAM J. Sci. Stat. Comput. 11 (1990) 811–823.

[42] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing Co, Boston, MA, 1996.