ELSEVIER

Applied Numerical Mathematics 30 (1999) 305–340

APPLIED NUMERICAL MATHEMATICS

A comparative study of sparse approximate inverse preconditioners

Michele Benzi [a,*,1], Miroslav Tůma [b,2]

[a] *Los Alamos National Laboratory, MS B256, Los Alamos, NM 87545, USA*
[b] *Institute of Computer Science, Czech Academy of Sciences, 182 07 Prague 8, Czech Republic*

Abstract

A number of recently proposed preconditioning techniques based on sparse approximate inverses are considered. A description of the preconditioners is given, and the results of an experimental comparison performed on one processor of a Cray C98 vector computer using sparse matrices from a variety of applications are presented. A comparison with more standard preconditioning techniques, such as incomplete factorizations, is also included. Robustness, convergence rates, and implementation issues are discussed.

Keywords: Sparse linear systems; Sparse matrices; Preconditioned Krylov subspace methods; Incomplete factorizations; Factorized approximate inverses; SPAI; FSAI; Incomplete biconjugation

1. Introduction

One of the most important problems in scientific computing is the development of efficient parallel iterative solvers for large, sparse systems of linear equations $Ax = b$. Krylov subspace methods, which require at each iteration matrix–vector products and a few vector operations (dot products, vector updates), can be efficiently implemented on high-performance computers, but they necessitate preconditioning in order to be effective. Many of the most popular general-purpose preconditioners, such as those based on incomplete factorizations of A, are fairly robust and result in good convergence rates, but are highly sequential and it is difficult to implement them efficiently on parallel computers, especially for unstructured problems. Thus, preconditioning is currently the main stumbling block precluding high performance in the solution of large, sparse linear systems.

* Corresponding author. E-mail: benzi@lanl.gov.

[1] Work supported in part by the Department of Energy through grant W-7405-ENG-36 with Los Alamos National Laboratory.

[2] E-mail: tuma@uivt.cas.cz. Work supported in part by the Grant Agency of the Czech Academy of Sciences through grants No. 2030706 and 205/96/0921.

0168-9274/99/$20.00
PII: S0168-9274(98)00118-4

Since the early days of vector and parallel processing, a great amount of work has been devoted to the problem of extracting as much inherent parallelism as possible from the best serial preconditioners, such as SSOR and incomplete factorization methods. As a result, good performance is possible for certain classes of problems (e.g., highly structured matrices arising from the discretization of PDEs on regular grids). On the other hand, it is still very difficult to solve general linear systems with an irregular sparsity pattern efficiently on vector and parallel computers.

Another line of research consists in developing alternative preconditioning methods which have natural parallelism. Among the first techniques of this type we mention polynomial preconditioners, which are based on approximating the inverse of the coefficient matrix A with a low-degree polynomial in the matrix. These methods have a long history (see, e.g., [20,24]), but came into vogue only after the first vector processors had become available [38,58]. Polynomial preconditioners only require matrix–vector products with A and therefore have excellent potential for parallelization, but they are not as effective as incomplete factorization methods at reducing the number of iterations. With polynomial preconditioning, the reduction in the number of iterations tends to be compensated by the additional matrix–vector products to be performed at each iteration. More precisely, it was shown by Axelsson [4] that the cost per iteration increases linearly with the number $m+1$ of terms in the polynomial, whereas the number of iterations decreases more slowly than $\mathrm{O}(1/(m+1))$. Therefore, polynomial preconditioners cannot be very effective, especially on serial and shared memory computers. For distributed memory parallel machines they are slightly more attractive, due to the reduced number of dot products and matrix accesses. Another attractive feature is that polynomial preconditioners require almost no additional storage besides that needed for the coefficient matrix A and have very small set-up times. Also, they can be used in a matrix-free context, and they are easily combined with other preconditioners. On the other hand, polynomial preconditioners have serious difficulties handling the case of matrices with general complex spectra (i.e., with eigenvalues on both sides of the imaginary axis). In summary, it is fair to say that polynomial preconditioning is unlikely to yield a satisfactory solution to the problem of high-performance preconditioning for general sparse matrices.

In recent years there has been a growing interest in yet another class of algebraic preconditioners—sparse approximate inverses. These methods are based on approximating the inverse matrix directly, as in polynomial preconditioning. However, with polynomial preconditioning the approximate inverse is available only implicitly in the form of a polynomial in the coefficient matrix A; in contrast, with sparse approximate inverse preconditioning a matrix $M \approx A^{-1}$ is explicitly computed and stored. The preconditioning operation reduces to a matrix–vector product with M. Methods of this kind were first proposed in the early 1970s (see [10,48]), but they received little attention, due to the lack of effective strategies for automatically determining a good nonzero pattern for the sparse approximate inverse. Several such strategies have recently been developed, thus spurring renewed interest in this class of preconditioners.

While parallel processing has been the main motivation driving the development of approximate inverse techniques, there has been at least one other influence. It is well known that incomplete factorization techniques can fail on matrices which are strongly nonsymmetric and/or indefinite. The failure is usually due to some form of instability, either in the incomplete factorization itself (zero or very small pivots), or in the back substitution phase, or both; see [30]. Most approximate inverse techniques are largely immune from these problems, and therefore constitute an important complement to more standard preconditioning methods even on serial computers.

We remark that approximate inverse techniques rely on the tacit assumption that for a given sparse matrix A, it is possible to find a sparse matrix M which is a good approximation of A^{-1}. However, this is not at all obvious, since the inverse of a sparse matrix is usually dense. More precisely, it can be proved that the inverse of an irreducible sparse matrix is structurally dense. This means that for a given irreducible sparsity pattern, it is always possible to assign numerical values to the nonzeros in such a way that all entries of the inverse will be nonzero; see [40]. Nevertheless, it is often the case that many of the entries in the inverse of a sparse matrix are small in absolute value, thus making the approximation of A^{-1} with a sparse matrix possible. Recently, much research has been devoted to the difficult problem of capturing the "important" entries of A^{-1} automatically.

There exist currently several alternative proposals for constructing sparse approximate inverse preconditioners, a few of which have been compared with standard incomplete factorization methods. However, a direct comparison between different approximate inverse techniques on a broad range of problems is still lacking. The present paper attempts to fill this gap. Our main contribution consists in a systematic computational study aimed at assessing the effectiveness of the various methods for different types of problems. While we refrain from ranking the various methods in some linear ordering, which would make little sense in a subject like preconditioning for general sparse matrices, we are able to draw some tentative conclusions about the various methods and to provide some guidelines for their use. In addition, we compare the approximate inverse techniques with various incomplete factorization strategies, showing that under appropriate circumstances approximate inverse preconditioning can indeed be superior to more established methods.

The paper is organized as follows. In Section 2 we give an overview of sparse approximate inverse techniques. In Section 3 we provide some information on how these techniques were implemented for our experiments. The results of numerical experiments, carried out on a Cray C98 vector computer, are presented and discussed in Section 4. The main results of this study are summarized in Section 5. An extensive list of references completes the paper.

2. Overview of approximate inverse methods

In this section we give an overview of approximate inverse techniques and some of their features. Because the emphasis is on the practical use of these methods, we do not give a detailed description of their theoretical properties, for which the interested reader is referred to the original papers. See also the overviews in [5,29,71]. It is convenient to group the different methods into three categories. First, we consider approximate inverse methods based on Frobenius norm minimization. Second, we describe factorized sparse approximate inverses. Finally, we consider preconditioning methods which consist of an incomplete factorization followed by an approximate inversion of the incomplete factors. The approximate inversion can be achieved in many ways, each leading to a different preconditioner. Throughout this section we shall be concerned with linear systems of the form $Ax = b$ where A is a real $n \times n$ matrix and b is a given real n-vector. The extension to the complex case is straightforward.

2.1. Methods based on Frobenius norm minimization

We begin with this class of approximate inverse techniques because they were historically the first to be proposed, and because they are among the best known approximate inverse methods. Also, the methods

in this subsection have the highest potential for parallelism. The basic idea is to compute a sparse matrix $M \approx A^{-1}$ as the solution of the following constrained minimization problem:

$$\min_{M \in \mathcal{S}} \|I - AM\|_F,$$

where $\mathcal{S}$ is a set of sparse matrices and $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. Since

$$\|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|_2^2,$$

where e_j denotes the jth column of the identity matrix, the computation of M reduces to solving n independent linear least squares problems, subject to sparsity constraints. This approach was first proposed by Benson [10]. Other early papers include [11,12], and [48].

Notice that the above approach produces a right approximate inverse. A left approximate inverse can be computed by solving a constrained minimization problem for $\|I - MA\|_F = \|I - A^{\mathrm{T}}M^{\mathrm{T}}\|_F$. This amounts to computing a right approximate inverse for A^{T}, and taking the transpose of the resulting matrix. In the case of nonsymmetric matrices, the distinction between left and right approximate inverses can be important. Indeed, there are situations where it is difficult to compute a good right approximate inverse, but it is easy to find a good left approximate inverse. Furthermore, when A is nonsymmetric and ill-conditioned, a matrix $M \approx A^{-1}$ may be a poor right approximate inverse, but a good left approximate inverse; see [65,66]. In the following discussion, we shall assume that a right approximate inverse is being computed.

In early papers, the constraint set $\mathcal{S}$, consisting of matrices with a given sparsity pattern, was prescribed at the outset. Once $\mathcal{S}$ is given, the computation of M is straightforward, and it is possible to implement such computation efficiently on a parallel computer. In a distributed memory environment, the coefficient matrix A can be distributed among the processors before the computation begins, and the construction of M is a local process which can be done with little communication among processors (such communication could be completely eliminated at the price of duplicating some of the columns of A).

When the sparsity pattern is fixed in advance, the construction of the preconditioner can be accomplished as follows. The nonzero pattern is a subset $\mathcal{G} \subseteq \{(i, j) \mid 1 \leqslant i, j \leqslant n\}$ such that $m_{ij} = 0$ if $(i, j) \notin \mathcal{G}$. Thus, the constraint set $\mathcal{S}$ is simply the set of all real $n \times n$ matrices with nonzero pattern contained in $\mathcal{G}$. Denote by m_j the jth column of M $(1 \leqslant j \leqslant n)$. For a fixed j, consider the set $\mathcal{J} = \{i \mid (i, j) \in \mathcal{G}\}$, which specifies the nonzero pattern of m_j. Clearly, the only columns of A that enter the definition of m_j are those whose index is in $\mathcal{J}$. Let $A(:, \mathcal{J})$ be the submatrix of A formed from such columns, and let $\mathcal{I}$ be the set of indices of nonzero rows of $A(:, \mathcal{J})$. Then we can restrict our attention to the matrix $\widehat{A} = A(\mathcal{I}, \mathcal{J})$, to the unknown vector $\widehat{m}_j = m_j(\mathcal{J})$ and to the right-hand side $\widehat{e}_j = e_j(\mathcal{I})$. The nonzero entries in m_j can be computed by solving the (small) unconstrained least squares problem

$$\|\widehat{e}_j - \widehat{A}\widehat{m}_j\|_2 = \min.$$

This least squares problem can be solved, for instance, by means of the QR factorization of $\widehat{A}$. Clearly, each column m_j can be computed, at least in principle, independently of the other columns of M. Note that due to the sparsity of A, the submatrix $\widehat{A}$ will contain only a few nonzero rows and columns, so each least squares problem has small size and can be solved efficiently by dense matrix techniques.

The role of the constraint set $\mathcal{S}$ is to preserve sparsity by somehow filtering out those entries of A^{-1} which contribute little to the quality of the preconditioner. For instance, one might want to ignore those

entries that are small in absolute value, while retaining the large ones.[3] Unfortunately, for a general sparse matrix it is not usually known where the large entries of the inverse are located, and this makes the *a priori* choice of a nonzero sparsity pattern for the approximate inverse very difficult. A possible exception is the case where A is a banded symmetric positive definite (SPD) matrix. In this case, the entries of A^{-1} are bounded in an exponentially decaying manner along each row or column; see [36]. Specifically, there exist $0 < \rho < 1$ and a constant C such that for all i, j

$$\left|(A^{-1})_{ij}\right| \leqslant C\rho^{|i-j|}.$$

The numbers ρ and C depend on the bandwidth and on the spectral condition number of A. For matrices having a large bandwidth and/or a high condition number, C can be very large and ρ very close to one, so that the decay could actually be so slow to be virtually imperceptible. On the other hand, if the entries of A^{-1} can be shown to decay rapidly, then a banded M would be a good approximation to A^{-1}. In this case, $\mathcal{S}$ would just be the set of matrices with a prescribed bandwidth.

For matrices with a general sparsity pattern, the situation is far more difficult. A common choice is to choose $\mathcal{S}$ to be the set of matrices with the same sparsity structure as the coefficient matrix A. This choice is motivated by the empirical observation that entries in the inverse of a sparse matrix $A = (a_{ij})$ which are located at positions (i, j) for which $a_{ij} \neq 0$ tend to be relatively large. However, this simple approach is not robust for general sparse problems, as there may be large entries of A^{-1} in positions outside the nonzero pattern of A. Another common approach consists in taking the sparsity pattern of the approximate inverse to be that of A^k where k is a positive integer, $k \geqslant 2$. This approach can be justified in terms of the Neumann series expansion of A^{-1}. While the approximate inverses corresponding to higher powers of A are often better than the one corresponding to $k = 1$, there is still no guarantee that they will result in satisfactory preconditioners. Furthermore, the costs for computing, storing and applying the preconditioner grow rapidly with k. Slightly more sophisticated strategies have been recently examined by Huckle [57], but more evidence is needed before these techniques can be considered effective. We mention here that some simple heuristics for prescribing a sparsity pattern have been successfully used for constructing sparse approximate inverse preconditioners for dense linear systems arising in the numerical solution of integral equations [1, 60].

Because for general sparse matrices it is difficult to prescribe a good nonzero pattern for M, several authors have developed adaptive strategies which start with a simple initial guess (for example, a diagonal matrix) and successively augment this pattern until a criterion of the type $\|e_j - Am_j\|_2 < \varepsilon$ is satisfied for a given $\varepsilon > 0$ (for each j), or a maximum number of nonzeros in m_j has been reached. Such an approach was first proposed by Cosgrove et al. [33]. Slightly different strategies were also considered by Grote and Huckle [54] and by Gould and Scott [52].

The most successful of these approaches is the one proposed by Grote and Huckle, hereafter referred to as the SPAI preconditioner [54]. The algorithm runs as follows.

Algorithm 1. SPAI algorithm

For every column m_j of M:

(1) Choose an initial sparsity $\mathcal{J}$.

[3] Note, however, that there is no guarantee that this will result in a good preconditioner; see [7].

(2) Determine the row indices $\mathcal{I}$ of the corresponding nonzero entries and the QR decomposition of $\widehat{A} = A(\mathcal{I}, \mathcal{J})$. Then compute the solution $\widehat{m}_j$ of the least squares problem $\|\widehat{e}_j - \widehat{A}\widehat{m}_j\|_2 = \min$ and its residual $r = \widehat{e}_j - \widehat{A}\widehat{m}_j$.

While $\|r\|_2 > \varepsilon$:

(3) Set $\mathcal{L}$ equal to the set of indices ℓ for which $r(\ell) \neq 0$.

(4) Set $\widetilde{\mathcal{J}}$ equal to the set of all new column indices of A that appear in all $\mathcal{L}$ rows but not in $\mathcal{J}$.

(5) For each $k \in \widetilde{\mathcal{J}}$ compute the norm ρ_k of the new residual via the formula

$$\rho_k^2 = \|r\|_2^2 - (r^{\mathrm{T}} A e_k)^2 / \|A e_k\|_2^2$$

and delete from $\widetilde{\mathcal{J}}$ all but the most profitable indices.

(6) Determine the new indices $\widetilde{\mathcal{I}}$ and update the QR factorization of the submatrix $A(\mathcal{I} \cup \widetilde{\mathcal{I}}, \mathcal{J} \cup \widetilde{\mathcal{J}})$. Then solve the new least squares problem, compute the new residual $r = e_j - Am_j$, and set $\mathcal{I} = \mathcal{I} \cup \widetilde{\mathcal{I}}$ and $\mathcal{J} = \mathcal{J} \cup \widetilde{\mathcal{J}}$.

This algorithm requires the user to provide an initial sparsity pattern, and several parameters. These parameters are: the tolerance ε on the residuals, the maximum number of new nonzero entries actually retained at each iteration (that is, how many of the most profitable indices are kept at step (5)), and also the maximum number of iterations for the loop (3)–(6). The latter two parameters together determine the maximum number of nonzeros allowed in each column of the approximate inverse. More detailed information on the implementation of this algorithm is given in Section 3.

As we shall see, the serial cost of computing the SPAI preconditioner can be very high, and the storage requirements rather stringent. In an attempt to alleviate these problems, Chow and Saad [27] proposed to use a few steps of an iterative method to reduce the residuals corresponding to each column of the approximate inverse. In other words, starting from a sparse initial guess, the n independent linear subproblems

$$Am_j = e_j, \quad j = 1, 2, \ldots, n,$$

are approximately solved with a few steps of a minimal residual-type method. For a small number of iterations, the approximate inverse columns m_j will remain sparse; elements that contribute little to the quality of the preconditioner can be removed. Simple dropping based on a drop tolerance is known to give poor results; a more effective strategy for removing the less profitable nonzeros is described in [27].

The basic algorithm, taken from [27], is called the Minimal Residual (MR) algorithm and runs as follows.

Algorithm 2. MR algorithm

(1) Choose an initial guess $M = M_0 = [m_1, m_2, \ldots, m_n]$.
(2) For each column m_j, $j = 1, 2, \ldots, n$, do
(3) For $i = 1, 2, \ldots, n_{\mathrm{i}}$ do
(4) $r_j = e_j - Am_j$
(5) $\alpha_j = r_j^{\mathrm{T}} A r_j / ((Ar_j)^{\mathrm{T}}(Ar_j))$
(6) $m_j = m_j + \alpha_j r_j$
(7) Apply numerical dropping to m_j
(8) End do
(9) End do

Here n_i denotes the number of iterations. At each step, this algorithm computes the current residual r_j and then performs a one-dimensional minimization of the residual norm $\|e_j - Am_j\|_2$ in the direction r_j with respect to α. In Algorithm 2, dropping is performed in m_j after the update, but other options are possible; see [27]. For efficiency, it is imperative that all the matrix–vector products, saxpy, and dot product kernels be performed in sparse–sparse mode. Besides the number of iterations n_i and the dropping criterion, the user is required to supply an upper bound on the number of nonzeros to be retained in each column of M. Another user-defined parameter is the initial guess M_0. Possible choices are $M_0 = 0$ or else a multiple of the identity matrix, $M_0 = \alpha I$, where α is chosen so as to minimize $\|I - \alpha A\|_F$. In some cases it may be useful to let M_0 be a multiple of A^T; see [27].

The iterative solution of the linear systems $Am_j = e_j$ could be improved by the use of a preconditioner. A natural idea is to use the already computed columns to precondition each linear system. This is referred to as *self-preconditioning* in [27]. There are at least two ways to perform self-preconditioning. The first is to run Algorithm 2 for a given number n_i of "inner" steps, then use the resulting approximate inverse M as a preconditioner for another sweep of n_i iterations, update M, and repeat this for a given number n_o of "outer" iterations. The preconditioner M is not updated until a whole sweep of all the columns has been completed. This is similar in spirit to the Jacobi iteration. The second way is to update the jth column of M right away, similar to a Gauss–Seidel iteration. Sometimes the second strategy results in a better approximate inverse, but this is not always the case, and, furthermore, the construction of the approximate inverse looses its inherent parallelism. As suggested in [27], a reasonable compromise is to process blocks of columns simultaneously, i.e., in a block Gauss–Seidel fashion.

Algorithm 3 below implements the MR iteration with self-preconditioning.

Algorithm 3. Self-preconditioned MR algorithm

(1) Choose an initial guess $M = M_0 = [m_1, m_2, \ldots, m_n]$.
(2) For $i = 1, 2, \ldots, n_o$ do
(3) For each column m_j, $j = 1, 2, \ldots, n$, do
(4) Let $s = m_j$
(5) For $i = 1, 2, \ldots, n_i$ do
(6) $r = e_j - As$
(7) $z = Mr$
(8) $q = Az$
(9) $\alpha_j = r^T q/(q^T q)$
(10) $s = s + \alpha_j z$
(11) Apply numerical dropping to s
(12) End do
(13) Update jth column of M: $m_j = s$
(14) End do
(15) End do

Notice that the user is required to provide the following parameters: the initial guess M_0, the number of inner and outer iterations n_i and n_o, the dropping criterion, and the bound on the number of nonzeros allowed in each column of M.

Because of the high number of user-defined parameters needed in input, the SPAI and MR algorithms are not easy to use, especially in an automated setting. Another drawback is that it is generally not

possible to ensure the nonsingularity of M, although in practice it is unlikely that an approximate inverse computed with these algorithms will be exactly singular. A more serious limitation of this type of preconditioners is that they cannot be used with the conjugate gradient method to solve SPD problems, since in general M will neither be symmetric nor positive definite, even if A is. Furthermore, the serial cost of MR is still relatively high, especially when self-preconditioning is used. Finally, because of the adaptive and irregular nature of the computations, the parallelism inherent in the construction of these preconditioners is not easily exploited in practice.

On the positive side, these methods have a good deal of flexibility. For instance, one can attempt to improve their quality simply by rerunning Algorithms 1–3 with M as an initial guess. This is especially true of the MR preconditioners, for which the only storage needed is that for M. Additional advantages of these approaches are their possible use to improve a given preconditioner and, for the MR methods, the fact that the coefficient matrix A is needed only in the form of matrix–vector products, which can be advantageous in some cases [27]. Also, MR may be used with iterative methods which can accommodate variations in the preconditioner, like FGMRES [71].

Another issue that deserves to be mentioned is the sensitivity of these preconditioners to reorderings. It is well known that incomplete factorization preconditioners are very sensitive to reorderings; see [16,43]. On the other hand, the SPAI and MR preconditioners are scarcely sensitive to reorderings. This is, at the same time, good and bad. The advantage is that A can be partitioned and reordered in whichever way is more convenient in practice, for instance to better suite the needs of a distributed implementation (e.g., load balancing) without having to worry about the impact on the convergence rate. The disadvantage is that reorderings cannot be used to reduce fill-in and/or improve the quality of the approximate inverse. For instance, if A^{-1} has no small entries, methods like SPAI will face serious difficulties, and no permutation of A will change this, since the inverse of a permutation of A is just a permutation of A^{-1}. This is not the case for the preconditioners described in the next two subsections.

Practical experience has shown that SPAI and MR preconditioners can solve very hard problems for which more standard techniques, like ILU, fail (see, e.g., [70]). In this sense, they provide a useful complement to more established techniques. Furthermore, these methods are potentially very useful for solving large problems on distributed memory machines, since both A and M can be distributed across the memories local to processors. Summarizing, in spite of some disadvantages, the techniques discussed in this section have several attractive features, and deserve serious consideration.

2.2. Factorized sparse approximate inverses

In this subsection we consider preconditioners based on incomplete inverse factorizations, that is, on incomplete factorizations of A^{-1}. If A admits the factorization $A = LDU$ where L is unit lower triangular,[4] D is diagonal, and U is unit upper triangular, then A^{-1} can be factorized as $A^{-1} = U^{-1}D^{-1}L^{-1} = ZD^{-1}W^{\mathrm{T}}$ where $Z = U^{-1}$ and $W = L^{-\mathrm{T}}$ are unit upper triangular matrices. Note that in general, the inverse factors Z and W will be rather dense. For instance, if A is an irreducible band matrix, they will be completely filled above the main diagonal. Factorized sparse approximate inverse preconditioners can be constructed by computing sparse approximations $\overline{Z} \approx Z$ and $\overline{W} \approx W$. The factorized approximate inverse is then

$$M = \overline{Z}\,\overline{D}^{-1}\overline{W}^{\mathrm{T}} \approx A^{-1},$$

[4] A *unit* triangular matrix is a triangular matrix with ones on the diagonal.

where $\overline{D}$ is a nonsingular diagonal matrix, $\overline{D} \approx D$.

There are several approaches available for computing approximate inverse factors of a nonsingular matrix A. A first class of methods, described in this subsection, does not require any information about the triangular factors of A: the factorized approximate inverse preconditioner is constructed directly from A. Methods in this class include the FSAI preconditioner introduced by Kolotilina and Yeremin [61], a related method due to Kaporin [59], incomplete (bi)conjugation schemes [15,18], and bordering strategies [71]. Another class of methods first compute an incomplete triangular factorization of A using standard techniques, and then obtain a factorized sparse approximate inverse by computing sparse approximations to the inverses of the incomplete triangular factors of A. We shall discuss such two-stage methods in the next subsection.

The FSAI method, proposed by Kolotilina and Yeremin [61], can be briefly described as follows. Assume that A is SPD, and let $\mathcal{S}_L$ be a prescribed lower triangular sparsity pattern which includes the main diagonal. Then a lower triangular matrix $\widehat{G}_L$ is computed by solving the matrix equation

$$\left(A\widehat{G}_L\right)_{ij} = \delta_{ij}, \quad (i,j) \in \mathcal{S}_L.$$

Here $\widehat{G}_L$ is computed by columns: each column requires the solution of a small "local" SPD linear system, the size of which is equal to the number of nonzeros allowed in that column. The diagonal entries of $\widehat{G}_L$ are all positive. Define $\widehat{D} = (\mathrm{diag}(\widehat{G}_L))^{-1}$ and $G_L = \widehat{D}^{1/2}\widehat{G}_L$, then the preconditioned matrix $G_L A G_L^{\mathrm{T}}$ is SPD and has diagonal entries all equal to 1. A common choice for the sparsity pattern is to allow nonzeros in G_L only in positions corresponding to nonzeros in the lower triangular part of A. A slightly more sophisticated (but also more costly) choice is to consider the sparsity pattern of the lower triangle of A^k where k is a small positive integer, e.g., $k = 2$ or $k = 3$; see [59].

The approximate inverse factor computed by the FSAI method can be shown to minimize $\|I - XL\|_F$ where L is the Cholesky factor of A, subject to the sparsity constraint $X \in \mathcal{S}_L$. Note, however, that it is not necessary to know L in order to compute G_L. The matrix G_L also minimizes, w.r.t. $X \in \mathcal{S}_L$, the Kaporin condition number of XAX^{T}:

$$\frac{1}{n}\mathrm{tr}\left(XAX^{\mathrm{T}}\right)/\det\left(XAX^{\mathrm{T}}\right)^{1/n},$$

where n is the problem size and $\mathrm{tr}(B)$ denotes the trace of matrix B; see [59].

The extension of FSAI to the nonsymmetric case is straightforward; however, the solvability of the local linear systems and the nonsingularity of the approximate inverse is no longer guaranteed unless all leading principal minors of A are nonzero, and some kind of safeguarding may be required. The FSAI preconditioner can be efficiently implemented in parallel, and has been successfully applied to the solution of difficult problems in the finite element analysis of structures; see [45,62]. Its main disadvantage is the need to prescribe the sparsity pattern of the approximate inverse factors in advance. Using the sparsity pattern of the lower triangular part of A is a simple solution, but it is often ineffective when solving general sparse problems; see the results in Section 4.2. Different choices of the sparsity pattern for SPD matrices arising in finite element analyses are described in [62]; see also the experiments in [82].

Another method of computing a factorized approximate inverse is the one based on incomplete (bi)conjugation, first proposed in [13]. This approach, hereafter referred to as the AINV method, is described in detail in [15] and [18]. The AINV method does not require that the sparsity pattern be known in advance, and is applicable to matrices with general sparsity patterns. The construction of the

AINV preconditioner is based on an algorithm which computes two sets of vectors $\{z_i\}_{i=1}^n$, $\{w_i\}_{i=1}^n$, which are A-biconjugate, i.e., such that $w_i^{\mathrm{T}} A z_j = 0$ if and only if $i \neq j$. Given a nonsingular matrix $A \in \mathbb{R}^{n\times n}$, there is a close relationship between the problem of inverting A and that of computing two sets of A-biconjugate vectors $\{z_i\}_{i=1}^n$ and $\{w_i\}_{i=1}^n$. If

$$Z = [z_1,\ z_2, \ldots, z_n]$$

is the matrix whose ith column is z_i and

$$W = [w_1,\ w_2, \ldots, w_n]$$

is the matrix whose ith column is w_i, then

$$W^{\mathrm{T}} A Z = D = \begin{pmatrix} p_1 & 0 & \ldots & 0 \\ 0 & p_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & p_n \end{pmatrix},$$

where $p_i = w_i^{\mathrm{T}} A z_i \neq 0$. It follows that W and Z are necessarily nonsingular and

$$A^{-1} = Z D^{-1} W^{\mathrm{T}} = \sum_{i=1}^{n} \frac{z_i w_i^{\mathrm{T}}}{p_i}.$$

Hence, the inverse of A is known if two complete sets of A-biconjugate vectors are known. Note that there are infinitely many such sets. Matrices W and Z whose columns are A-biconjugate can be explicitly computed by means of a biconjugation process applied to the columns of any two nonsingular matrices $W^{(0)}, Z^{(0)} \in \mathbb{R}^{n\times n}$. A computationally convenient choice is to let $W^{(0)} = Z^{(0)} = I$: the biconjugation process is applied to the unit basis vectors. In order to describe the procedure, let a_i^{T} and c_i^{T} denote the ith row of A and A^{T}, respectively (i.e., c_i is the ith column of A). The basic A-biconjugation procedure can be written as follows.

Algorithm 4. Biconjugation algorithm

(1) Let $w_i^{(0)} = z_i^{(0)} = e_i$ $(1 \leqslant i \leqslant n)$.
(2) For $i = 1, 2, \ldots, n$ do
(3) For $j = i, i+1, \ldots, n$ do
(4) $p_j^{(i-1)} := a_i^{\mathrm{T}} z_j^{(i-1)}$; $q_j^{(i-1)} := c_i^{\mathrm{T}} w_j^{(i-1)}$
(5) End do
(6) If $i = n$ go to (11)
(7) For $j = i+1, \ldots, n$ do
(8) $z_j^{(i)} := z_j^{(i-1)} - (p_j^{(i-1)}/p_i^{(i-1)}) z_i^{(i-1)}$; $w_j^{(i)} := w_j^{(i-1)} - (q_j^{(i-1)}/q_i^{(i-1)}) w_i^{(i-1)}$
(9) End do
(10) End do
(11) Let $z_i := z_i^{(i-1)}$, $w_i := w_i^{(i-1)}$ and $p_i := p_i^{(i-1)}$, for $1 \leqslant i \leqslant n$. Return $Z = [z_1,\ z_2, \ldots, z_n]$, $W = [w_1,\ w_2, \ldots, w_n]$ and $D = \mathrm{diag}(p_1, p_2, \ldots, p_n)$.

This algorithm can be interpreted as a (two-sided) generalized Gram–Schmidt orthogonalization process with respect to the bilinear form associated with A. Some references on this kind of algorithm are [21,31,46,47]. If A is SPD, only the process for Z need to be carried out (since in this case $W = Z$),

and the algorithm is just a conjugate Gram–Schmidt process, i.e., orthogonalization of the unit vectors with respect to the "energy" inner product $\langle x,\ y\rangle := x^{\mathrm{T}}Ay$.

It is easy to see that, in exact arithmetic, the above process can be completed without divisions by zero if and only if all the leading principal minors of A are nonzeros or, equivalently, if and only if A has an LU factorization [18]. In this case, if $A = LDU$ is the decomposition of A as a product of a unit lower triangular matrix L, a diagonal matrix D, and a unit upper triangular matrix U, it is easily checked that $Z = U^{-1}$ and $W = L^{-\mathrm{T}}$ (D being exactly the same matrix in both factorizations). Because of the initialization chosen (step (1) in Algorithm 4), the z- and w-vectors are initially very sparse; however, the updates in step (8) cause them to fill-in rapidly. See [18] for a graph-theoretical characterization of fill-in in Algorithm 4. Sparsity in the inverse factors is preserved by carrying out the biconjugation process incompletely, similar to ILU-type methods. Incompleteness can be enforced either on the basis of position, allowing the vectors $\overline{z}_i$ and $\overline{w}_i$ to have nonzero entries only in prescribed locations, or on the basis of a drop tolerance, whereby new fill-ins are removed if their magnitude is less than a prescribed threshold $Tol > 0$. Unsurprisingly, the second strategy is much more robust and effective, particularly for unstructured problems. Because of incompleteness, the question arises of whether the preconditioner construction can be performed without breakdowns (divisions by zero): in [15] it is proved that a sufficient condition is that A be an H-matrix, similar to ILU [63,64]. In the general case, diagonal modifications may be necessary.

A third approach that can be used to compute a factorized sparse approximate inverse preconditioner directly from the input matrix A is the one based on bordering. Actually, several schemes are possible. The method we describe here is a modification of one first proposed by Saad in [71, p. 308]. Let A_k denote the principal leading $k \times k$ submatrix of A. Consider the following bordering scheme:

$$\begin{pmatrix} W_k^{\mathrm{T}} & 0 \\ w_k^{\mathrm{T}} & 1 \end{pmatrix} \begin{pmatrix} A_k & v_k \\ y_k^{\mathrm{T}} & \alpha_{k+1} \end{pmatrix} \begin{pmatrix} Z_k & z_k \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} D_k & 0 \\ 0 & \delta_{k+1} \end{pmatrix},$$

where $z_k = -Z_k D_k^{-1} W_k^{\mathrm{T}} v_k$, $w_k = -W_k D_k^{-1} Z_k^{\mathrm{T}} y_k$ and $\delta_{k+1} = \alpha_{k+1} + w_k^{\mathrm{T}} A_k z_k + y_k^{\mathrm{T}} z_k + w_k^{\mathrm{T}} v_k$. Here W_k, Z_k are unit upper triangular matrices of order k, w_k, v_k, y_k and z_k are k-vectors and α_{k+1} and δ_{k+1} are scalars, with $\alpha_{k+1} \equiv a_{k+1,k+1}$. Starting from $k = 1$, this scheme suggests an obvious algorithm for computing the inverse factors of A (assuming A has an LU factorization). When this scheme is carried out incompletely, an approximate factorization of A^{-1} is obtained. Sparsity can be preserved by dropping elements in the vectors w_k and z_k after they are computed, for instance by means of a drop tolerance, as in the AINV process. The resulting factorized sparse approximate inverse preconditioner will be referred to as AIB, for Approximate Inverse via Bordering. Besides a matrix–vector product with A_k, the construction of the AIB preconditioner requires four sparse matrix–vector products involving W_k, Z_k and their transposes at each step k, which account for most of the work. It is important that these operations be performed in sparse–sparse mode.

Note that the computations for the inverse factors Z and W are tightly coupled, in contrast to the biconjugation Algorithm 4, where the two factors can be computed independently. As usual, if A is symmetric, $W = Z$ and work is halved. Furthermore, if A is SPD, then it can be shown [71] that, in exact arithmetic, $\delta_k > 0$ for all k. Therefore, the AIB preconditioner is always well-defined in the SPD case. In the general case, diagonal modifications may be required in order to complete the process.

Factorized sparse approximate inverses are free from some of the problems that limit the effectiveness of other approaches. Unlike the methods described in the previous subsection, they can be used as preconditioners for the conjugate gradient method for solving SPD problems. Indeed, if A is SPD, then

$Z = W$ and the preconditioner $M = \overline{Z}\,\overline{D}^{-1}\overline{Z}^{\mathrm{T}}$ is SPD as long as the diagonal entries of $\overline{D}$ are all positive. It is also clear that the nonsingularity of M is trivial to check when M is expressed in factorized form. Following [26] (and contrary to what stated in [23, p. 109]), it can be argued that factorized forms provide better approximations to A^{-1}, for the same amount of storage, than nonfactorized ones, because they can express denser matrices than the total number of nonzeros in their factors. As our experiments show, this intuitive observation almost always translates in better convergence rates for the same number of nonzeros. In addition, factorized forms are less expensive to compute, and in most cases they require less user-defined parameters than the techniques in the previous subsection. Finally, factorized forms are sensitive to reorderings of the coefficient matrix, a fact which can be exploited to reduce fill-in in the inverse factors and/or to improve the rate of convergence; see [18,19,22].

On the other hand, factorized forms have problems of their own. Being incomplete (inverse) factorization methods, they can fail due to breakdowns during the incomplete factorization process, like ILU. While diagonal shifts [63] or diagonal compensation strategies [6] can be used to safeguard the computation, there is no guarantee that the resulting preconditioner will be effective, especially when large and/or numerous shifts are needed. The FSAI method requires to prescribe the nonzero structure of the factors in advance, which makes it difficult to use on problems with general sparsity patterns. Other methods, like AINV, offer limited opportunity for parallelization of the preconditioner construction phase. In its current formulation, AINV appears to be difficult to implement on distributed memory machines. Also, the parallelism in the application of factorized approximate inverses is somewhat less than that for nonfactorized forms, since it is now necessary to perform two matrix–vector multiplies with the factors, in sequence.

2.3. Inverse ILU techniques

Several authors have proposed to construct factorized sparse approximate inverse preconditioners based on the following two-stage process: first an incomplete LU factorization $A \approx \overline{L}\,\overline{U}$ is computed using standard techniques, and then the incomplete factors $\overline{L}$ and $\overline{U}$ are approximately inverted; see [2,34, 75,76,80]. There are various possibilities for computing approximate inverses of $\overline{L}$ and $\overline{U}$, each leading to a different preconditioner. Here we give only a very brief description, referring the interested reader to the original papers for more information (see also Section 3.3).

Assuming that incomplete factors $\overline{L}$ and $\overline{U}$ are available, approximate inverse factors can be computed by inexactly solving the $2n$ triangular linear systems

$$\overline{L}x_i = e_i, \quad \overline{U}y_i = e_i \quad (1 \leqslant i \leqslant n).$$

Notice that these linear systems can all be solved independently; hence, a good deal of parallelism is available, at least in principle. The linear systems are solved approximately to reduce costs and, of course, because sparsity must be preserved in the columns of the approximate inverse factors.

One possibility is to prescribe sparsity patterns $\mathcal{S}_L$ and $\mathcal{S}_U$ for the approximate inverses of $\overline{L}$ and $\overline{U}$. Sparse approximations can then be computed using a Frobenius norm-type approach. In alternative, the adaptive SPAI method could be used to approximately invert $\overline{L}$ and $\overline{U}$, without the need to prescribe the sparsity patterns. Some tests were conducted in [80], where it was concluded that this approach is not recommended.

A better approach consists in solving the $2n$ triangular systems for the columns of $\overline{L}^{-1}$ and $\overline{U}^{-1}$ by forward and back substitution, respectively. Sparsity is preserved by dropping in the solution vectors,

either on the basis of position (more generally, using a level-of-fill scheme) or on the basis of a drop tolerance. Several such schemes have been described in detail in [34,76,80]. Some authors have proposed to drop in $x_i = \overline{L}^{-1} e_i$ and $y_i = \overline{U}^{-1} e_i$ once the exact solution has been computed, but a more practical scheme is to drop during the substitution process, rather than after [80].

The preconditioners in this class share some of the advantages of factorized approximate inverse methods, but suffer from certain disadvantages that the preconditioners described in the previous subsections do not have. These disadvantages all stem from the assumption that an ILU factorization has already been computed. This implies that these methods are not even applicable if an ILU factorization does not exist, or if it is unstable, as is sometimes the case for highly nonsymmetric, indefinite problems [30,44]. Clearly, this assumption also limits the parallel efficiency of this class of methods, since the preconditioner construction phase is not entirely parallelizable (computing an ILU factorization is a highly sequential process).

Another disadvantage is that the computation of the preconditioner involves two levels of incompleteness, rather than just one, as is the case for the other approximate inverse methods considered in this paper. For some problems this could lead to a significant degradation in the quality of the preconditioner. Perhaps more importantly, the presence of two levels of incompleteness makes these methods difficult to use in practice, owing to the necessity to choose a large number of user-defined parameters. For instance, if an ILUT-like dual threshold approach is used [69], then the user is required to choose the values of four parameters, two for the ILUT factorization and other two for the approximate inversion of the ILUT factors. Notice that the values of the parameters will generally be very different for the two phases, in particular the drop tolerance is typically much larger in the approximate inverse phase than in the ILUT phase. In our experience, inverse ILU methods are much more difficult to use than the factorized preconditioners described in Section 2.2.

We conclude this overview of approximate inverse preconditioners with a brief discussion of inverse ILU methods based on truncated Neumann expansions. Strictly speaking, these are not approximate inverse methods; rather, they can be regarded as a hybrid of ILU and polynomial preconditioning techniques. However, they are similar to the other methods in this subsection in that they are based on an ILU factorization in which the incomplete factors are inverted inexactly by applying some kind of truncation. In particular, the forward and back substitutions are replaced by matrix–vector products with sparse triangular matrices. This idea goes back to van der Vorst [78], and has been recently applied to the SSOR preconditioner, which can be seen as a kind of incomplete factorization, by Gustafsson and Lindskog [55]. The truncated Neumann SSOR preconditioner for a symmetric matrix A is defined as follows. Let $A = E + D + E^{\mathrm{T}}$ be the splitting of A into its strictly lower triangular, diagonal, and strictly upper triangular part. Consider the SSOR preconditioner

$$M = (\omega^{-1}D + E)(\omega^{-1}D)^{-1}(\omega^{-1}D + E^{\mathrm{T}}),$$

where $0 < \omega < 2$ is a relaxation parameter. Let $L = \omega E D^{-1}$ and $\widehat{D} = \omega^{-1}D$, then

$$M = (I + L)\widehat{D}(I + L)^{\mathrm{T}}$$

so that

$$M^{-1} = (I + L)^{-\mathrm{T}}\widehat{D}^{-1}(I + L)^{-1}.$$

Notice that

$$(I+L)^{-1} = \sum_{k=0}^{n-1} (-1)^k L^k.$$

For some matrices, i.e., diagonally dominant ones, $\|L^k\|$ decreases rapidly as k increases, and the sum can be approximated with a polynomial of low degree (say, 1 or 2) in L. For example, to first order,

$$G = \left(I - L^{\mathrm{T}}\right)\widehat{D}^{-1}(I-L) \approx M^{-1} \approx A^{-1}$$

can be regarded as a factorized sparse approximate inverse of A.

Because the SSOR preconditioner does not require any actual computation (except possibly for the estimation of ω), this is a virtually free preconditioner. It is also very easy to implement. On the other hand, the effectiveness of this method is restricted to problems for which the standard SSOR preconditioner works well, and this is not a very general class of problems. Furthermore, truncation of the Neumann expansion to a polynomial of low degree may result in a serious degradation of the rate of convergence, particularly for problems which are far from being diagonally dominant. The idea of truncated Neumann expansions can be also applied to incomplete Cholesky and, more generally, incomplete LU factorization preconditioning. While this approach seems to be somewhat more robust than the one based on SSOR, all the caveats concerning the drawbacks of inverse ILU methods apply.

3. Notes on implementation

This section is devoted to practicalities concerning the implementation of the preconditioners considered in this paper. Our main goal here is to give an idea of the programming effort, storage requirements, and type of data structures needed for implementing each algorithm. These are important aspects that should be taken into account when comparing different methods. Although our implementation was written for a uniprocessor, we also comment on certain aspects relevant for parallel computation.

3.1. Implementation of Frobenius norm-based methods

We begin with the SPAI preconditioner (Algorithm 1). Our implementation is as close as possible to the description of the algorithm given in [54]. Input data structures include the original matrix A stored by columns (that is, A is stored in CSC, or Harwell–Boeing format) and the matrix structure stored by rows. This means that in all we store five arrays: the array `val` of numerical values, and the arrays of integers `rowind, colptr, colind` and `rowptr` (see [9, Section 4.3]). Explicit storage of the additional row information contained in the vectors `colind` and `rowptr` is necessary in order to efficiently perform searches on the submatrices $\widehat{A}$ determined by sets of row and column indices. In step (1) of Algorithm 1 we used a diagonal sparsity pattern as the initial guess. The SPAI algorithm needs to solve a set of dense least squares problems in each of the n outer steps required to compute individual columns of the preconditioner. These calculations are based on incremental QR factorizations of dense submatrices; thus, additional workspace is needed to store the factors Q and R. We used LAPACK level 3 BLAS routines [3] for this task. Computed columns of the preconditioner M are stored in a static data structure in CSC format.

As noted in [54], a nice feature of SPAI is that the computation of each column of the preconditioner can be regarded as an iterative process. If the iterative solver preconditioned with SPAI does not converge, it is possible to improve the preconditioner by performing additional steps of Algorithm 1 on at least some of the columns of M, without having to recompute an approximate inverse from scratch. However, there are two difficulties with this incremental approach. First, if the (dense) QR factors of the individual submatrices are not stored, the recomputation will take a nontrivial time. Hence, the storage requirements could be very stringent. Second, it is not clear that the columns corresponding to the largest least-squares residuals should be precisely the ones to be recomputed (as was suggested in [54]), since in general it is not clear what is the relation, if any, between the size of the least-squares residuals and the rate of convergence of the preconditioned iteration.

The need to access the matrix structure both by rows and by columns in a dynamic and impredictable way is the main reason why a parallel implementation of SPAI is nontrivial, especially in a distributed memory environment. If we assume some initial distribution of the matrix among the processors, then the processors which compute different columns need to communicate matrix elements during the course of the computation. For a sophisticated solution to this problem using MPI, see [7,8].

Our implementation of Chow and Saad's MR preconditioner (with or without self-preconditioning) is based on the descriptions in [27]. The storage requirements for the basic MR technique (Algorithm 2) are very limited. The input matrix A is stored by columns, and apart from some auxiliary vectors, no additional workspace is needed. Recall that the key ingredient is the use of matrix–vector products performed in sparse–sparse mode. These can be performed efficiently when A is stored by columns. In output, the preconditioner M is stored in a static data structure, also in CSC format. The initial guess used was a scaled identity matrix, $M_0 = \alpha I$ (using $M_0 = \alpha A^{\mathrm{T}}$ gave poorer results). This is consistent with the initial guess in SPAI and also in AINV, where we start from the identity matrix I. Dropping during the iterative MR process is based on the local strategy described in [27]. Elements in a candidate column of M are dropped after the update at each MR iteration. We did not apply dropping in the search direction. In the candidate column of the approximate inverse we keep only those entries that give the strongest estimated decrease of the column residual norm. The number of elements which are kept in each candidate column after this dropping is bounded by an input parameter *lfil*, similarly to the dual threshold strategy in the ILUT preconditioner. We did not use a drop tolerance to restrict the set of positions which are tested for this decrease (as suggested in [27]) because we found that this causes a deterioration in the quality of the preconditioner.

For the numerical experiments with the self-preconditioned MR iteration (Algorithm 3) we chose to update the preconditioning matrix after each outer iteration. That is, we opted for a Jacobi-type approach rather than for a Gauss–Seidel-type one. The reason is that we tried both strategies and, somewhat surprisingly, we found that the former approach is more effective than the latter. The implementation of self-preconditioned MR is more demanding in terms of storage, since at the end of each outer iteration a new approximation of the preconditioner must be stored, requiring an additional space of the size of the preconditioner.

To our knowledge, no parallel implementation of MR preconditioning has been developed yet. However, similar remarks as for SPAI apply concerning the computation of any of the MR preconditioners in a parallel environment. The nonzero structure of each column of the approximate inverse changes dynamically in the course of the MR iteration, thus requiring interprocessor communication to perform the sparse matrix–vector products. Again, the memory access patterns are not known in advance.

3.2. Implementation of factorized approximate inverses

We discuss here the implementation of factorized approximate inverse techniques. These preconditioners are applicable to symmetric and nonsymmetric problems as well. The symmetric codes require roughly half the operations and space needed in the nonsymmetric case.

Computing the FSAI preconditioner with sparsity pattern equal to that of A is very simple. For extracting the dense submatrices we only need A, stored in a static data structure. The preconditioner is computed in place and therefore is held in a static data structure as well. In addition, we need enough workspace to hold the dense submatrix of maximal dimension arising during the computation of any column of the preconditioner. Dense factorizations are based on level 3 BLAS routines from LAPACK. Unlike the SPAI and MR preconditioners, which were applied only to nonsymmetric problems, the FSAI preconditioner can also be used on SPD problems with the conjugate gradient method. Thus, implementations of FSAI were developed both for symmetric and nonsymmetric problems.

The construction of the AINV preconditioner has many similarities with the computation of standard incomplete Cholesky (IC) or ILU preconditioners, only slightly more complicated. This complication is mostly caused by the more general rules governing the generation of fill-in. The input matrix A is stored by rows. Note that even if A is symmetric we store both its lower and upper triangular parts, since AINV needs to perform dot products with the rows of A, and we want this operation to be performed efficiently. For very large problems, however, it may be undesirable or even impossible to store all of A. In this case, efficiency may be retained by a careful choice of the data structure holding A; a possible solution has been proposed in the LINSOL package [73].

If A is nonsymmetric, besides A stored by rows, we store A by columns, since in this case AINV performs dot products also with the columns of A. However, it should be kept in mind that the two processes (the one for Z and the one for W) are completely uncoupled, therefore it is not necessary to have A stored both by rows and by columns at the same time. On the other hand, on a computer with at least two processors, where Z and W can be computed concurrently, one may want to have A stored both by rows and columns. Notice that this is also the case for certain iterative methods (like Bi-CG or QMR) which perform matrix–vector products with both A and A^{T}. An alternative might be to adopt the strategy used in LINSOL, where a matrix–vector product of the form $u = A^{\mathrm{T}}v$ is computed as $u^{\mathrm{T}} = v^{\mathrm{T}}A$: again, a careful choice of data structures is required for efficient execution of this operation on vector and parallel machines; see [73]. However, we have not tried this strategy within our codes.

The approximate inverse factor $\overline{Z}$ can be computed based on the right-looking Algorithm 4. Obviously, the computation of the approximate inverse factor $\overline{W}$, which is needed in the nonsymmetric case, is identical to that of $\overline{Z}$, only with A^{T} replacing A. With the right-looking process, fill-in at step i of Algorithm 4 is introduced into the submatrix formed by columns $i+1,\ldots,n$ of the upper triangular matrix $\overline{Z}$. Sparsity in $\overline{Z}$ is preserved by removing fill, either on the basis of position or on the basis of value using a drop tolerance. If a sparsity pattern is prescribed, the computation is straightforward and it can be carried out using static data structures only, as for FSAI. Indeed, the computation consists exclusively of vector updates (saxpy's) and dot products. For general sparse problems, however, this approach is not recommended, due to the difficulty of guessing a good sparsity pattern. Experiments performed using the same sparsity pattern as the coefficient matrix A indicated that this is not a robust strategy. Numerical dropping based on a drop tolerance is far more effective, although more complicated to implement. While structural predictions of fill-in could be used in principle to set up static working data structures as in the case of Cholesky factorization, these predictions are often too pessimistic to be

useful. Therefore, $\overline{Z}$ is stored by columns using dynamic data structures, similar to standard right-looking implementations of sparse unsymmetric Gaussian elimination; see, e.g., [39,83]. Nonzero entries of each column are stored consecutively as a segment of a larger workspace. During the AINV algorithm the length of the individual segments grows as a result of new fill-in or, temporarily, shrinks due to dropping of some entries. If a column segment needs more consecutive positions to store the new entries than it has free space available around, it is moved to some other part of the workspace. The workspace is periodically shrunk to decrease its fragmentation into small parts. This process is called a *garbage collection*. The number of garbage collections is reduced if enough *elbow room*, which is an additional free space for $\overline{Z}$, is provided. Providing sufficient elbow room can speed up the computation of the preconditioner significantly, especially for problems with a relatively large fraction of nonzero entries. For the larger test problems, elbow room was approximately two times the anticipated storage for the preconditioner. To ensure efficiency of the right-looking approach, we use additional information about the factors storing their structure also by rows. This row structure is needed to efficiently perform the sparse vector updates in Algorithm 4. At step i, the row structure information for $\overline{Z}$ is used and then updated.

Similar to ILUT, it is possible to limit the number of nonzeros allowed in a column of $\overline{Z}$, so that the storage requirements can be predicted. However, based on a limited number of experiments, we noticed that this strategy is rather restrictive in some cases. The reason is that for some problems there are certain columns of Z (and W), usually the last few, which contain a large proportion of nonnegligible entries. As already observed in [15], the work in AINV is distributed over algorithmic steps less uniformly than in the case of sparse Gaussian elimination. Hence, limiting the number of nonzeros to a same prescribed small number for all columns of $\overline{Z}$ may result in relatively poor preconditioner performance. This also explains, at least in part, why preserving sparsity based on position does not work very well.

Before the AINV process, the coefficient matrix A and the right-hand side b are rescaled by dividing them by $\max_{1\leqslant i,j\leqslant n}|a_{ij}|$. Thus, all matrix entries lie in the interval $[-1,1]$. This global scaling has no effect on the spectral properties of A, but it helps in the choice of the value for the drop tolerance, typically a number $0 < \mathit{Tol} < 1$. In our experience, $\mathit{Tol} = 0.1$ is usually a good guess, producing in many cases an approximate inverse with sparsity comparable to that of A. For consistency, the same scaling was used for all the preconditioners which use a drop tolerance. Another option available in our code is the possibility of using a relative drop tolerance. The value of *Tol* can be dynamically adjusted at each step, taking into account the size of the elements involved in the updates at that step. However, compared to the simpler strategy based on a constant value of *Tol*, we did not see in our tests a significant difference in the behavior of the preconditioner.

The AINV algorithm needs to be safeguarded against the risk of breakdowns or near-breakdowns. Diagonal elements p_i were modified (shifted) if they were found to be too small. Each time a diagonal element p_i was found to be smaller than an approximate machine precision (10^{-15}) in absolute value, we changed its value to 10^{-1} for SPD problems, and to $\mathrm{sgn}(p_i)\cdot 10^{-1}$ for nonsymmetric ones. The choice of the constant 10^{-1} was arbitrary; the results did not change significantly when other values were tried. It should be mentioned that for the test matrices used in this paper, diagonal modifications were seldom needed.

Concerning the implementation of the AIB preconditioner, it has already been noted that most of the work in the algorithm is concentrated in the four matrix–vector products with Z_k, W_k and their transposes. For efficiency, it is important that such products be computed in sparse matrix–sparse vector mode. Therefore, a careful choice of data structures for the partial factors of the approximate inverse

is needed. If a sparse matrix is stored by rows, matrix–vector products with it will necessarily require a number of operations proportional to the number of nonzeros in the matrix. On the other hand, if the matrix is stored by columns, the number of operations required for a sparse matrix–sparse vector product will be proportional to the total number of nonzeros in the columns determined by the sparsity structure of the vector. Therefore, an efficient implementation of the bordering scheme requires to store the approximate inverse factors twice: by columns and by rows. Note that it could be possible to store the numerical values of the factors only once. In this case, however, the number of operations for a matrix–vector multiply with the matrix stored by rows and its structural pattern stored also by columns would increase by a count proportional to the number of nonzeros in rows of the matrix determined by the sparsity pattern of the resulting product. In some cases this strategy may significantly increase the time for the preconditioner computation.

A dual threshold dropping strategy was used to preserve sparsity in the AIB preconditioner. Imposing a bound on the number of nonzeros in columns of the approximate inverse factors $\overline{Z}$ and $\overline{W}$ enables the efficient construction of these factors throughout the algorithm. Consider the case of $\overline{Z}$. Once we have computed a column of $\overline{Z}$ containing *lfill* entries at most, we update the rows of $\overline{Z}$ using the loose bound of $2 \cdot lfill$ on the number of entries in the updated rows. Not imposing a bound could result in a much less efficient computation. Note that AINV does not need such restriction which, as we mentioned, can sometimes result in a preconditioner of lesser quality. Diagonal elements are modified if considered too small, according to the same rule as for the AINV algorithm. This safeguarding was also used for the SPD case, since in inexact arithmetic breakdowns are possible.

3.3. Implementation of inverse ILU methods

In this subsection we provide some information on the implementation of preconditioners based on the approximate inversion of some given IC or ILU factors. These factors may be obtained either by a level-of-fill incomplete factorization ILU(k), or by a dual threshold (ILUT) approach, or by the equivalent incomplete Cholesky factorizations in the SPD case. We omit the details of the implementations of these factorizations, which are standard (see [71]).

We begin with an implementation based on the algorithms in [80]. It is sufficient to describe the inverse IC/ILU process for a lower triangular matrix $\overline{L}$. From the point of view of storage, besides that for $\overline{L}$ we need some working vectors and static data structures to store, in output, the columns of the approximate inverse factor. More specifically, to compute a sparse approximation to the ith column of $\overline{L}^{-1}$, it is necessary to have access to the submatrix comprising the last $n - i + 1$ rows and columns of $\overline{L}$. To see this, it is sufficient to consider the back substitution for solving $\overline{L}x = e_i$. This shows that, while the columns of the approximate inverse of $\overline{L}$ can in principle be computed independently, a parallel distributed memory implementation of this approach would be nontrivial, due to the need of taking into account communication and load balancing. Another drawback, of course, is the fact that an ILU factorization must be previously computed, and this is a highly sequential task, in general.

First, a search of the structure of the columns of $\overline{L}$ is performed. This search sequentially determines which of those columns contribute to a given column of $\overline{L}^{-1}$. Every time a column of $\overline{L}$ is found which updates the ith column of $\overline{L}^{-1}$, the structure of the ith column of $\overline{L}^{-1}$ is recomputed to reflect the situation *after* this update and then its numerical values are actually computed. The computation of each update contributing to a given column of $\overline{L}^{-1}$ is therefore performed interleaving two phases, a symbolic and a

numeric one. Sparsity in the columns of the inverse of $\overline{L}$ can be preserved on the basis of position (more generally, using a level-of-fill mechanism) or on the basis of value using drop tolerances. In all cases, dropping is done after each update.

A somewhat different approach was used in [2,34]. There the dropping was performed after computing the exact columns of $\overline{L}^{-1}$. Although this strategy may result in more accurate preconditioners than the previously described one, it is not a practical one. This can be seen, for instance, considering the case of matrices with a band structure, for which the inverse triangular factor $\overline{L}^{-1}$ is completely filled below the main diagonal.

3.4. Further notes on implementations

In addition to the previously described algorithms, several other techniques were implemented, including diagonal scaling, different versions of IC and ILU factorizations and, for SPD problems only, a least-squares polynomial preconditioner with Jacobi weight function and parameters $\mu = \frac{1}{2}$, $\nu = -\frac{1}{2}$. For the latter, we estimated the end points of the spectrum using Gerschgorin circles, and Horner's scheme was used to compute the action of the polynomial preconditioner on a vector; see [67,71] for details. Horner's scheme was also used with the truncated Neumann expansion methods.

Because all the test matrices used in this study have a zero-free diagonal, the simple preconditioner based on diagonal scaling is always well-defined. The reciprocals of the diagonal entries are computed and stored in a vector before the iteration begins; applying the preconditioner requires n multiplications at each step, where n is the problem size. For matrices which have zero entries on the main diagonal, nonsymmetric permutations can be used to produce a zero-free diagonal [41], as was done in [18].

The implementation of the Krylov subspace accelerators (conjugate gradients, GMRES, etc.) was fairly standard. All the matrix–vector products with the coefficient matrix A and with each of the (possibly factorized) approximate inverse preconditioners are vectorizable operations. To this end, after the approximate inverse preconditioners have been computed, they are transformed into the JAD, or jagged diagonal, format (see [56,68]). The same is done with the coefficient matrix A. Although the matrix–vector products still involve indirect addressing, using the JAD format results in good, if not outstanding, vector performance. For execution on multivector computers, there exists a blocked variant of the JAD format which can be used to take advantage of multiple vector processing units; see [56]. We remark that the use of the JAD format in matrix–vector multiplications involves a permutation of the product vector at each iteration in all cases except for factorized approximate inverses applied to SPD problems, since in this case the permutations of $\overline{Z}$ and $\overline{Z}^{\mathrm{T}}$ cancel each other out. Transformation to JAD format was also used to vectorize the inverse IC/ILU preconditioners based on approximate inversion and truncated Neumann expansions. A vectorizable version of the SSOR preconditioner based on truncated Neumann expansions was also coded. Due to the difficulty of determining a good value of the relaxation parameter ω for all the test problems, in our experiments we used $\omega = 1$. Thus, the SSOR preconditioner is really a Symmetric Gauss–Seidel (SGS) one. Only for the IC/ILU preconditioners the factors were not transformed into JAD data structures. No attempt was done to vectorize the triangular solves, and this part of the computation runs at scalar speed on the Cray C98, at least for the test problems considered here, which are quite sparse. Vector performance can be obtained in some cases if the incomplete factors are sufficiently dense; see, e.g., [14].

It is worth mentioning that the matrix–vector products with the factorized approximate inverse preconditioners achieve better performance in the nonsymmetric case than in the symmetric case. The

reason is that in the symmetric case, only $\overline{Z}$ is explicitly stored. Thus, matrix–vector products with $\overline{Z}^{\mathrm{T}}$ do not vectorize as well as matrix–vector products with $\overline{Z}$. This problem does not occur in the nonsymmetric case, where both $\overline{Z}$ and $\overline{W}$ must be explicitly stored. The performance in the symmetric case can be significantly enhanced at the price of storing $\overline{Z}^{\mathrm{T}}$ explicitly.

We conclude this section on implementation with a brief discussion of issues related to cache reuse (possibly with different cache levels), which is important given the fast-growing popularity of SMP architectures and of distributed memory machines based on microprocessors. On machines with hierarchical memories, it is important to design algorithms that reuse data in the top level of the memory hierarchy as much as possible. If we look at the preconditioner construction phase, two of the methods discussed here have a better chance of allowing cache reuse than all the remaining ones, and these are SPAI and FSAI. The reason is that these are the only two algorithms which make use of level 3 BLAS. Some remarks on latency hiding and cache reuse issues within SPAI can be found in [7,8]. In principle, blocking in the dense matrix operations required by these two algorithms can be used to obtain performance; see [37,50]. The other algorithms suffer from such disadvantages as the nonlocal character of the computations and, in some cases, a high proportion of non-floating-point operations. As an example, consider the construction of the AINV preconditioner (Algorithm 4; similar considerations apply to other methods as well). The work with both the row and column lists in each step of the outer loop is rather irregular. For larger problems, most operations are scattered around the memory and are out-of-cache. As a consequence, it is difficult to achieve high efficiency with the code, and any attempt to parallelize the computation of the preconditioner in this form will face serious problems (see [83] for similar comments in the context of sparse unsymmetric Gaussian elimination).

In an attempt to mitigate these problems, an alternative implementation of the AINV algorithm was considered in [18]. This left-looking, delayed update version of the biconjugation algorithm can be implemented using static data structures only, at the price of increasing the number of floating-point operations. This increase in arithmetic complexity is more or less pronounced, depending on the problem and on the density of the preconditioner. On the other hand, this formulation greatly decreases the amount of irregular data structure manipulations. It also appears better suited to parallel implementation, as discussed in [18]. The question arises whether the increased operation count is so great as to offset the advantages of a more regular arrangement of the computations. Numerical experiments performed on a SGI Crimson workstation with RISC processor R4000 indicate that the left-looking implementation is beneficial for problems of small size. For larger problems, on the other hand, the alternative implementation is actually slower than the original one; see [18]. However, more work is needed before definite conclusions can be drawn.

Concerning the application of approximate inverse preconditioners at each step of an iterative method, this amounts to performing matrix–vector products involving a sparse matrix and a dense vector. For a discussion of the problem of cache reuse in sparse matrix–vector products, see [74].

4. Numerical experiments

In this section we present the results of numerical experiments carried out on one processor of a Cray C98 vector computer (located at Météo France in Toulouse). Although the main interest of approximate inverse techniques lies in their potential for parallelization, we think that results on a vector processor are also relevant and can give some idea of the advantages afforded by this class of preconditioners. The

Table 1
SPD test problems information

Matrix	n	nnz	Application
NOS3	960	8402	Biharmonic equation
NOS7	729	2673	Diffusion equation
685BUS	685	1967	Power system network
1138BUS	1138	2596	Power system network
NASA2146	2146	37198	Structural analysis
NASA2910	2910	88603	Structural analysis
BCSSTK21	3600	15100	Structural analysis
BCSSTK23	3134	24156	Structural analysis
FDM1	6050	18028	Diffusion equation
FDM2	32010	95738	Diffusion equation

various algorithms were tested on a selection of sparse matrices representative of a variety of different applications: finite difference and finite element discretizations of partial differential equations, power systems networks, circuit analysis, etc. We group these test problems in two classes: SPD problems and general (nonsymmetric, possibly indefinite) problems. The preconditioned conjugate gradient (PCG) method [51] is used for solving the problems in the first class. For the problems in the second class, three popular Krylov subspace methods were tested: restarted GMRES [72], Bi-CGSTAB [79], and TFQMR [49]. Good general references on iterative methods are [53] and [71]; see also [9] for a concise introduction.

All codes developed for the tests [5] were written in Fortran 77 and compiled using the optimization option $-Zv$. In the experiments, convergence is considered attained when the 2-norm of the (unpreconditioned) residual is reduced to less than 10^{-9}. The initial guess is always $x_0 = 0$, and the right-hand side is chosen so that the solution of the linear system is the vector of all ones; other choices for the right-hand side and different stopping criteria (e.g., relative ones) were also tried, without significant differences in the results. For the nonsymmetric problems, right preconditioning was always used. All timings reported are CPU times in seconds.

4.1. Experiments with SPD problems

Here we summarize the results of numerical experiments with a set of ten sparse symmetric positive definite matrices arising in various applications. Table 1 gives, for each matrix, the order n, the number of nonzeros in the lower triangular part nnz, and the application in which the matrix arises.

Matrices NOS$*$, $*$BUS, and BCSSTK$*$ are extracted from the Harwell–Boeing collection [42]; the NASA$*$ matrices are extracted from Tim Davis' collection [35], and the FDM$*$ matrices were kindly provided by Carsten Ullrich of CERFACS. Matrices NOS3, NASA$*$ and BCSSTK$*$ arise from finite

[5] These codes can be obtained from the authors for research purposes.

element modeling, matrices NOS7 and FDM$*$ from finite difference modeling. Problem FDM1 is a five-point stencil discretization of a diffusion equation with Dirichlet boundary conditions on a uniform mesh; problem FDM2 represents the same equation, but on a locally refined mesh. The most difficult problems in this set are the ones from structural analysis, especially NASA2910 and BCSSTK23. Although this is not a very large data set, it is sufficiently representative of the large number of matrices that we have experimented with over the last few years.

The following preconditioners were tested:

- Diagonal scaling, DS;
- No-fill incomplete Cholesky, IC(0);
- Drop tolerance-based incomplete Cholesky, IC(*Tol*);
- Least-squares Jacobi polynomial preconditioning of degree ℓ, JP(ℓ);
- Kolotilina and Yeremin's factorized sparse approximate inverse, FSAI;
- Drop tolerance-based incomplete conjugation, AINV(*Tol*);
- Dual threshold factorized approximate inverse by bordering, AIB(*Tol*, *lfill*);
- Various inverse IC methods based on level-of-fill or drop tolerances, IIC($*$);
- Truncated Neumann Symmetric Gauss–Seidel of degree ℓ, TNSGS(ℓ);
- Truncated Neumann IC($*$) of degree ℓ, TNIC($*$, ℓ).

For the FSAI preconditioner, the sparsity pattern of the lower triangular part of A was imposed on the approximate inverse factor. In addition, the conjugate gradient method without preconditioning was also tested. The nonfactorized approximate inverse preconditioners based on Frobenius norm minimization cannot be used with the conjugate gradient method and were not included.

The DS, IC(0), and FSAI preconditioners are parameter-free: they are easy to implement and use, but cannot be tuned. The preconditioners which use a drop tolerance, like AINV and the dual threshold versions of IC and AIB are relatively easy to use, and can be tuned to cope with difficult problems, usually by allowing more nonzeros in the preconditioner. Likewise, the truncated Neumann SGS preconditioner contains one parameter, the degree ℓ. Normally, increasing ℓ will result in faster convergence, but usually not in faster execution due to the additional work required at each PCG iteration. The hardest preconditioners to use are the ones that require the highest number of user-defined parameters, namely, the various forms of inverse IC methods. For the preconditioners which use a drop tolerance, we tried preconditioners with different amount of nonzeros, typically between half and twice the number of nonzeros in the original matrix. In most cases, increasing the amount of nonzeros in the preconditioner results in faster convergence, as measured by the number of iterations. Again, this does not always translate in smaller solution times, since the cost of constructing and applying the preconditioner grows with the number of nonzeros. In the majority of cases, the fastest execution times were obtained with preconditioners having approximately the same number of nonzeros as the coefficient matrix A.

Given the number of methods considered, it would be impractical to present tables with the results of all the runs performed. Instead, we will try to synthesize what we have learned based on these runs.

First, we look at the robustness of the various methods. All iterations were stopped when the prescribed reduction of the residual norm was achieved, or a maximum number *maxit* of iterations was reached. This maximum was different for different problems and preconditioners: for no preconditioning or diagonal scaling, we set $maxit = n$, where n is the dimension of the problem. For the other preconditioners, we set $maxit = \min\{n, 1000\}$. In the following, we say that a preconditioner *fails* if the PCG iteration with that preconditioner did not converge in *maxit* iterations or less. Failures are invariably associated with stagnating residual norms.

The most robust preconditioner is IC with a drop tolerance. With this preconditioner, it was possible to solve all ten problems, using at most twice the number of nonzeros as in the original matrix. In particular, matrix BCSSTK23 could be solved only with this type of preconditioner. The following preconditioners were able to solve all problems except for BCSSTK23: DS, FSAI, and AIB. The AINV preconditioner was able to solve eight problems; it failed on NASA2910 and BCSSTK23, due to the occurrence of a large number of diagonal modifications prompted by small pivots. Of the various inverse IC preconditioners tried, the best results were obtained with IIC(Tol, 0), which is constructed from an IC factorization with a drop tolerance Tol followed by approximate inversion of the incomplete Cholesky factor $\overline{L}$ with sparsity pattern equal to that of $\overline{L}$. This preconditioner also solved eight problems, and again it failed on NASA2910 and BCSSTK23. The other inverse IC preconditioners were less robust or were extremely costly to compute. The no-fill IC preconditioner and the truncated Neumann SGS methods failed on three problems: besides NASA2910 and BCSSTK23, IC(0) failed on BCSSTK21, and TNSGS failed on NASA2146. Truncated Neumann versions of IC factorizations based on a drop tolerance failed on the same three problems as TNSGS, while truncated Neumann versions of IC(0) failed on all four structural analysis problems. The conjugate gradient method without preconditioning failed in five cases, and the polynomial preconditioner JP(ℓ) was, with six failures, the least robust method of all. Concerning this method, it was noticed that it worked really well only for the simple model problem FDM1 (a diffusion equation with Dirichlet boundary conditions on a square), where it was in fact the fastest solver overall. This, incidentally, shows the perils of testing algorithms only on simple model problems. The fact that, apart from direct solvers, only IC preconditioning seems to be able to handle very ill-conditioned linear systems arising from finite element modeling in structural engineering was also observed in [14].

The drop tolerance-based IC methods were also the most effective at reducing the number of PCG iterations, although this did not always result in the shortest time to solution, due to the inefficiency of the triangular solves. The AINV and AIB preconditioners are less effective than IC, but more effective than both FSAI and the truncated Neumann techniques. Diagonal scaling, no preconditioning and polynomial preconditioning resulted in the slowest convergence rates.

When looking at efficiency, in terms of time to solution, it is convenient to break the total solution time into the time for computing the preconditioner and the time for the PCG iteration. This is of interest in the common situation where a sequence of several linear systems with the same coefficient matrix (or a slowly varying one) and different right-hand sides have to be solved, for in this case the cost of constructing the preconditioner becomes less important. In this case, costly preconditioners may become attractive if they are very effective at reducing the number of iterations.

If efficiency is defined as the shortest time to solution including the time for constructing the preconditioner, then the most efficient method overall is IC(Tol), which was fastest in four cases. However, this is in part due to the fact that this preconditioner was the only one that never failed. The simple DS preconditioner, which failed in one case, was fastest on three problems, and TNSGS(1) was fastest on two. It should be remembered, however, that TNSGS failed on three problems. JP was fastest on one problem. We found that using $\ell > 1$ in JP(ℓ) and in the truncated Neumann methods almost always resulted in higher computing times, as predicted by the result of Axelsson mentioned in the Introduction. The approximate inverse preconditioners are not very efficient, at least in a uniprocessor implementation, due to the high cost of the set-up phase. DS, TNSGS, IC(0) and TNIC(0, ℓ) are the least expensive preconditioners to compute, followed by IC(Tol) and TNIC(Tol, ℓ). Next come FSAI, AINV and AIB, which have comparable cost, while the inverse IC preconditioners based on drop tolerances are, on average, the most expensive to compute. These methods are most expensive when applied to matrices

with a regular banded structure, since in this case $\overline{L}^{-1}$ is completely filled below the main diagonal. Of the various possible implementations, the one proposed in [80] is the only practical one, the one in [34] being prohibitively expensive. The cheapest inverse IC methods are the ones where a sparsity pattern equal to that of $\overline{L}$ is used for the approximate inverse of $\overline{L}$. Of all the preconditioners, FSAI is the only one which benefits from vectorization in the set-up phase, due to the use of level 3 BLAS.

If we look at the time for the iterative part only, then approximate inverse techniques become attractive, even on a uniprocessor, because of the impact of vectorization on the matrix–vector products and the relatively good convergence rates. However, the fastest method overall is still IC(*Tol*) (four problems). AINV was fastest on three problems, TNSGS(1) on two, and AIB on one problem. The performance of AIB is very close to that of AINV, with very small differences in the timings. As for FSAI, it is almost always outperformed by both AINV and AIB, due to somewhat slower convergence. In a parallel implementation the approximate inverse methods would be even more attractive, particularly AIB and AINV, but also FSAI due to the fact that this method is completely parallelizable and relatively easy to implement on a distributed memory machine.

In Table 2 we present results for a few selected preconditioners applied to the largest problem in our data set, matrix FDM2. We report the time for computing the preconditioner (P-time), the time for the iterative part only (It-time), and the number of iterations (Its). For the drop tolerance-based preconditioners IC, AINV and AIB we give results for two cases, the first corresponding to a preconditioner with approximately the same number of nonzeros as the coefficient matrix, and the second with about twice as many nonzeros.

From this table, it appears that the AIB preconditioner with approximately the same number of nonzeros as the original matrix becomes attractive, provided that it can be reused a sufficient number of times. Assuming each right-hand side takes the same It-time, it can be estimated that AIB preconditioning is the fastest method if at least 35 right-hand sides are present, otherwise DS is fastest. Matrix FDM2

Table 2
Results for problem FDM2

Precond.	P-time	It-time	Its
None	–	10.22	3359
DS	0.08	2.55	804
IC(0)	0.21	15.02	209
IC($2 \cdot 10^{-3}$)	0.43	13.29	183
IC(10^{-4})	0.76	6.57	87
FSAI	5.24	3.07	548
AINV(0.25)	3.52	2.74	422
AINV(0.1)	3.77	2.82	298
AIB(0.2,3)	3.48	2.45	393
AIB(0.08,5)	3.79	2.67	282

arises in an application which requires a very large number of solves with the same coefficient matrix and different right-hand sides, so AIB can be recommended in this case. It should be mentioned that the time for the iterative part of the FSAI, AINV and AIB preconditioners can be reduced by about 25% if the transpose approximate inverse factor $\overline{Z}^{\mathrm{T}}$ is explicitly stored (in JAD format).

For completeness, we mention that additional experiments were conducted with AINV preconditioning with a prescribed sparsity pattern, equal to that of A. This preconditioner is not robust: it failed on the four structural analysis problems and also on 1138BUS. Hence, it is much less robust than FSAI, which also uses the nonzero structure of A. On the other hand, it is interesting to observe that for the three problems from finite difference modelling (NOS7, FDM$*$) the static AINV preconditioner resulted in better convergence rates than the "optimal" FSAI preconditioner. In the case of NOS7, this difference was particularly marked: static AINV required 33 iterations, as compared to 89 for FSAI. Hence, for a fixed sparsity pattern, optimality in the sense of Frobenius norm minimization does not necessarily guarantee optimal, or even near-optimal, convergence behavior.

Based on our experiments, we can draw the following main conclusions concerning preconditioning of SPD problems:

- Drop tolerance-based IC is the most robust method among those tested;
- On a single processor, IC(*Tol*) is also the fastest method overall;
- FSAI and AIB are the most robust among approximate inverse methods;
- DS is a simple and viable option for problems with only one or few right-hand sides;
- TNSGS sometimes performs quite well, but it lacks robustness;
- For problems with several right-hand sides, AINV and AIB can be efficient even on a single (vector) processor;
- The inverse IC methods are not competitive with the other factorized approximate inverse methods;
- The various inverse and truncated Neumann IC methods and the polynomial preconditioner JP are not competitive with the other preconditioners.

Hence, for parallel preconditioning of the conjugate gradient method, the approximate inverse preconditioners AIB, AINV and FSAI should all be considered, although they may fail on very difficult problems. These techniques performed quite well on the finite difference and power systems problems, not so well on the problems from finite element modelling. We emphasize that the performance of FSAI may be further improved by a more sophisticated choice of the sparsity pattern; see [62,82].

4.2. Experiments with nonsymmetric problems

Here we summarize the results of numerical experiments with a set of twenty sparse nonsymmetric matrices arising in various applications. Table 3 gives, for each matrix, the order n, the number of nonzeros nnz, and the application in which the matrix arises.

All the matrices from oil reservoir simulations are extracted from the Harwell–Boeing collection. The matrices from circuit modelling and semiconductor device simulation are from Tim Davis' collection. Matrix 3DCD comes from a seven-point discretization on a $20 \times 20 \times 20$ grid of a convection–diffusion operator on the unit cube with homogeneous Dirichlet boundary conditions. Matrix ALE3D was made available by Steve Barnard of NASA Ames (see [8]). This matrix was permuted as to have a zero-free diagonal. Although most matrices come from the same field of petroleum engineering, this data set represents a broad collection of problems which are quite diverse, both structurally and numerically.

Table 3
Nonsymmetric test problems information

Matrix	n	nnz	Application
3DCD	8000	53600	Convection–diffusion equation
ALE3D	1590	45090	Metal forming simulation
ORSREG1	2205	14133	Oil reservoir simulation
ORSIRR1	1030	6858	Oil reservoir simulation
ORSIRR2	886	5970	Oil reservoir simulation
SAYLR3	1000	3750	Oil reservoir simulation
SAYLR4	3564	22316	Oil reservoir simulation
ADD32	4960	23884	Circuit modelling
ADD20	2395	17319	Circuit modelling
MEMPLUS	17758	99147	Circuit modelling
SWANG1	3169	20841	Semiconductor device simulation
SHERMAN1	1000	3750	Oil reservoir simulation
SHERMAN2	1080	23094	Oil reservoir simulation
SHERMAN3	5005	20033	Oil reservoir simulation
SHERMAN4	1104	3786	Oil reservoir simulation
SHERMAN5	3312	20793	Oil reservoir simulation
PORES2	1224	9613	Oil reservoir simulation
PORES3	532	3474	Oil reservoir simulation
WATT1	1856	11360	Oil reservoir simulation
WATT2	1856	11550	Oil reservoir simulation

Again, our experience encompasses a much larger set of problems; the present collection is chosen because it is sufficiently representative.

The following preconditioners were tested:

- Diagonal scaling, DS;
- No-fill incomplete LU, ILU(0);
- Dual threshold incomplete ILU, ILUT(*Tol*, *lfill*);
- Grote and Huckle's preconditioner, SPAI;
- Chow and Saad's preconditioner, MR;
- MR with self-preconditioning, MRP;
- Drop tolerance-based incomplete biconjugation, AINV(*Tol*);
- Dual threshold factorized approximate inverse by bordering, AIB(*Tol*, *lfill*);

- Various inverse ILU methods based on level-of-fill or drop tolerances, IILU($*$);
- Truncated Neumann Symmetric Gauss–Seidel of degree ℓ, TNSGS(ℓ);
- Truncated Neumann ILU(0) and ILUT of degree ℓ.

For the preconditioners which use a drop tolerance, we tuned the parameters so as to allow a number of nonzeros between about half and twice that of the coefficient matrix. Again, the best results were often obtained for preconditioners of approximately the same density as the coefficient matrix. In addition, we conducted experiments with a nonsymmetric version of FSAI and with a static version of AINV (in both cases imposing the sparsity pattern of the coefficient matrix A), and also with a dual threshold incomplete Gauss–Jordan (IGJ) preconditioner [32,81]. These approximate inverse methods are not robust: FSAI failed on seven problems, static AINV on twelve, and IGJ on thirteen. Moreover, when these preconditioners succeeded, convergence was generally slow. Therefore, these preconditioners cannot be recommended. A similar conclusion with respect to IGJ preconditioning was reached by Weiss in [81, p. 172].

Although we experimented with three accelerators (GMRES(20), Bi-CGSTAB, and TFQMR), here we will report results for Bi-CGSTAB only. On average, this method appeared to be superior to the other two solvers, if only slightly. It can be said with some confidence that the preconditioner is more important than the iterative solver, in the sense that the convergence rate usually depends on the quality of the preconditioner, almost independently of the underlying Krylov subspace method used.

For this set of problems, we allow a maximum of $maxit = n$ iterations for no preconditioning or diagonal scaling, and $maxit = 500$ for all the other preconditioners. Bi-CGSTAB with no preconditioning fails on six problems, while DS fails on two problems. The most robust preconditioners are ILU(0) and ILUT, which never failed. AINV and truncated Neumann versions of ILU(0) and ILUT failed on one problem, SHERMAN2. No approximate inverse method was able to solve SHERMAN2, at least within the sparsity constraints we imposed (at most twice the number of nonzeros as the coefficient matrix). The failure of AINV on SHERMAN2 was not due to the occurrence of small pivots, but to the difficulty of finding good sparse approximations to the inverse factors. An inverse ILUT method where a dual threshold strategy was used in both stages of the preconditioner computation failed on two problems (SHERMAN2 and SHERMAN3). SPAI, AIB and truncated Neumann SGS failed on three problems. However, SPAI can be made to succeed on one of these problems by computing a left approximate inverse, and on another one by allowing considerably more fill-in. The MR preconditioner failed on four problems. The self-preconditioned version MRP of MR is not robust: it failed on eight problems. This is probably due to the fact that self-preconditioning with a poor approximate inverse can spoil the convergence of the MR iteration; see [27]. Indeed, we found that self-preconditioning improved the MR preconditioner in nine cases, but made things worse in eight cases (it did not have an appreciable impact in three cases). Although self-preconditioning might be useful, especially for difficult problems, it is difficult to decide in advance when it should be used. Finally, inverse ILU preconditioners based on levels of fill, with ten failures, are not robust.

Again, ILU techniques do a better job than approximate inverse ones at reducing the number of iterations, at least on average. But this does not usually translate in the fastest execution, due to the inefficiency of the triangular solves. The fastest methods, including the set-up times for the preconditioner, are truncated Neumann versions of ILUT and SGS: each of these resulted in the shortest times to solution in six cases. Truncated Neumann ILU(0) was fastest in two cases, as were ILUT and no preconditioning. DS and ILU(0) were fastest on one problem each. It should also be mentioned that TNSGS was among the three fastest methods in twelve cases, TNILU(0) in ten cases, and TNILUT

in nine cases. Thus, for the case of a single (or a few) right-hand sides, truncated Neumann techniques deserve attention. We note that TNILU(0) was faster than ILU(0) in sixteen cases, and TNILUT was faster than ILUT in fifteen. This means that in a majority of cases, the degradation in the rate of convergence due to the truncation in the Neumann expansion was more than compensated by the impact of vectorization on the application of the preconditioner.

Sparse approximate inverse methods, which are not competitive (on one processor) if only one or very few right-hand sides are present, become much more attractive in the case of many right-hand sides. If we look at the time for the iterative part only, we find that AINV was fastest in nine cases, and IILUT in seven. SPAI, MRP, ILUT and no preconditioning were fastest on one problem each. Perhaps more indicatively, AINV was among the three fastest methods in eighteen cases, SPAI in thirteen, IILUT in ten, and AIB in eight cases. AINV was the approximate inverse method that resulted in the fastest convergence. As expected, factorized approximate inverse methods delivered (on average) faster convergence rates than SPAI or MR, for the same number of nonzeros in the preconditioner. From our experiments, SPAI appears to be more effective than MR. SPAI is more robust than MR and, in nearly all problems, SPAI resulted in faster convergence than MR. Also, we did not find MR less expensive to compute than SPAI, on average (and certainly not when self-preconditioning is used). This is in part due to the fact that SPAI takes advantage, albeit in a limited way, of vectorization in the calculations involving level 3 BLAS. On a scalar processor, MR would be somewhat faster to compute than SPAI (for a comparable density in the preconditioner). An advantage of MR over SPAI appears to be the fact that it requires less storage than SPAI, except when self-preconditioning is used. The truncated Neumann techniques resulted in slower convergence than the sparse approximate inverse preconditioners; the ILU-type methods, on the other hand, produced better convergence rates, but higher timings due to the triangular solves. Thus, sparse approximate inverse methods are the most efficient when several right-hand sides are present, even on a single (vector) processor. These techniques are even more attractive on multiprocessors. It is interesting to observe that approximate inverse techniques were particularly effective on the problems from circuit modelling.

Clearly, the minimum number of right-hand sides that must be present before approximate inverse techniques become viable depends on the time it takes to construct the preconditioner, and on the rate of convergence. On one processor, the least expensive approximate inverse preconditioner was found to be AIB, which required the shortest set-up time in sixteen cases (among the approximate inverse techniques only, of course). AINV was a close second. Indeed, AINV was among the three fastest approximate inverse preconditioners to compute on all test problems. The inverse ILU methods are more expensive than AINV or AIB, but less expensive than SPAI, MRP and MR. Of course, the situation could be different in a parallel implementation, since the SPAI and MR-type preconditioners have higher potential for parallelization than AINV, AIB and the inverse ILU methods. We note that SPAI was the only preconditioner to benefit from vectorization in the set-up phase, a consequence of the fact that this method is the only one that makes use of level 3 BLAS.

In Table 4 we illustrate the performance of Bi-CGSTAB with a few selected preconditioners on two matrices with completely different features, namely, the convection–diffusion problem 3DCD and the circuit problem MEMPLUS. For the drop tolerance-based methods ILUT, AINV, AIB, MR and SPAI, two preconditioners were computed. For 3DCD, the first preconditioner contained about the same number of nonzeros as the coefficient matrix, and the second one about twice as many. For MEMPLUS, the first preconditioner contained roughly half the number of nonzeros as the coefficient matrix, and

Table 4
Results for problems 3DCD and MEMPLUS

Matrix	Precond.	P-time	It-time	Its
3DCD	DS	0.02	0.25	156
	ILU(0)	0.06	0.55	15
	ILUT(0.1,0)	0.22	0.55	15
	ILUT(0.01,5)	0.64	0.29	7
	TNILU(0,1)	0.06	0.10	31
	TNSGS(1)	0.02	0.41	131
	AINV(0.17)	1.88	0.07	25
	AINV(0.1)	2.52	0.09	22
	AIB(0.25,5)	1.55	0.09	32
	AIB(0.065,6)	2.14	0.08	22
	MR(0.1,7,5)	11.9	0.14	51
	MR(0.01,14,5)	17.5	0.17	43
	SPAI(0.43,6,5)	10.6	0.11	40
	SPAI(0.31,5,5)	30.5	0.14	32
MEMPLUS	DS	0.03	6.71	730
	ILU(0)	1.91	16.4	224
	ILUT(0.05,4)	0.31	6.53	108
	ILUT(0.002,9)	0.71	3.15	40
	TNILU(0,5)	1.91	8.55	243
	TNSGS(1)	0.03	1.86	151
	AINV(0.1)	4.60	2.85	430
	AINV(0.02)	9.17	0.27	29
	AIB(0.2,2)	3.10	2.40	387
	AIB(10^{-7}, 5)	4.91	3.29	378
	MR(0.5,5,5)	220	1.65	223
	MR(0.1,10,5)	536	1.69	180
	SPAI(0.5,10,5)	129	1.01	140
	SPAI(0.3,10,5)	354	0.94	117

the second one had approximately the same density as MEMPLUS. Here MR(ε, m, n_i) stands for the MR preconditioner (Algorithm 2) corresponding to a drop tolerance ε, a maximum of m nonzeros per column, and n_i iterations. Similarly, SPAI(ε, m, n_i) is the SPAI preconditioner (Algorithm 1) with residual tolerance ε, a maximum of m new candidates added at each step, and n_i steps.

The fastest methods overall are TNILU(0) for 3DCD, and TNSGS for MEMPLUS. However, if we look at the time for the iterative part only, we see that AINV is the fastest method. Both SPAI and MR result in slower convergence than AINV, and they are far more expensive to compute. We mention that a detailed comparison between ILU(0), AINV and SPAI, with timings for all twenty test matrices used here, can be found in [17].

Based on our experiments, we can draw the following main conclusions concerning preconditioning of nonsymmetric problems:

- ILU-type preconditioners are the most robust methods among those tested;
- For one or few right-hand sides, the most efficient schemes overall are truncated Neumann versions of SGS and ILU (but robustness could be a problem);
- AINV is the most robust and effective of the approximate inverse methods;
- The inverse ILU methods are not competitive with the other factorized approximate inverse methods;
- Factorized approximate inverses are more effective than nonfactorized ones (for the same number of nonzeros in the preconditioner);
- On a single processor, factorized approximate inverses are much less expensive to construct than nonfactorized ones;
- SPAI is more robust and effective than MR;
- For problems with several right-hand sides, approximate inverse techniques can be competitive even on a single (vector) processor.

We see that the conclusions are to a large extent similar in the SPD and in the nonsymmetric case. One difference is that in the SPD case the best factorized approximate inverse method is AIB, whereas in the nonsymmetric case AINV seems to be better. Hovewer, these two methods performed similarly on most test problems. It should also be kept in mind that AIB requires almost twice the storage required by AINV. Another difference is that factorized approximate inverses were somewhat more efficient in the nonsymmetric case than in the SPD case. This was due in part to the fact that in the symmetric case, matrix–vector products with the transpose inverse factor $\overline{Z}^{\mathrm{T}}$, which is not explicitly available, vectorize less well than matrix–vector products with $\overline{Z}$. Finally, we note that diagonal scaling, which performed quite well on SPD problems, is much less effective in the nonsymmetric case.

5. Conclusions and perspectives

In this paper, a comprehensive study of sparse approximate inverse preconditioners was presented. Virtually all the methods described in the literature were surveyed, and their implementation on a vector computer was described. Extensive numerical tests on matrices from a variety of applications were performed, with the goal of assessing the effectiveness and efficiency of the various techniques. We also compared these new techniques with well-established preconditioning strategies, like incomplete factorization methods.

Incomplete factorization preconditioners have been vigorously developed over the course of several decades, and constitute a mature technology. In contrast, approximate inverse methods have received attention only in recent years. Thus, it is not surprising that the incomplete factorization methods were found to be somewhat more robust than approximate inverse ones. In particular, approximate inverse preconditioners seem to have some trouble solving some of the very ill-conditioned linear systems arising in structural engineering, which incomplete factorization methods are able to solve. We think that this is due to the fact that the entries in the inverses of these matrices decay very slowly away from the main diagonal. Hence, it is inherently difficult to approximate the inverse with a sparse matrix.

On the other hand, approximate inverse techniques were found to perform well on network problems, and were more effective than ILU methods on circuit problems. The matrices from these application areas appear to have an inverse which can be approximated well by a sparse matrix. Our experiments also show that, generally speaking, approximate inverse techniques work quite well on matrices arising from finite difference discretizations of boundary value problems, and less well on matrices arising from finite element modelling. This is probably due to the fact that finite difference matrices tend to be "more diagonally dominant" than finite element ones, a property which results in faster decay of the entries of A^{-1}.

Our experiments also show that factorized approximate inverses are more effective and less expensive to construct than the other approximate inverse methods. The approximate inverse based on bordering was found to be especially effective for symmetric positive definite problems, and the one based on incomplete biconjugation showed good performance for general nonsymmetric, indefinite problems. The main drawback of these methods is that, in their present formulation, they cannot be efficiently constructed in parallel, especially on distributed memory architectures. For these types of architectures, the Frobenius norm-based methods FSAI (for SPD problems) and SPAI (for the nonsymmetric case) deserve special attention.

Other preconditioners, like MR or the inverse IC/ILU techniques, were found to be less robust and effective than the other methods; however, they may very well be useful for special problems. For instance, MR has proved valuable in the solution of very difficult incompressible fluid flow problems for which ILU preconditioning failed [26,70]. These techniques could be a useful complement to the other methods; therefore, it is desirable that software packages for general-purpose preconditioning of sparse linear systems include as many different methods as possible.

Truncated Neumann techniques offer an inexpensive way to introduce parallelism in standard serial preconditioners, and our experiments show that these methods can be quite efficient, particularly when the preconditioner cannot be reused a number of times. On the other hand, sparse approximate inverse preconditioners, which are more expensive to construct, are generally more robust and result in faster convergence. Therefore, approximate inverses should be preferred whenever the cost of forming the preconditioner can be considered as negligible, that is, when the same preconditioner can be used in solving a sequence of linear systems. Based on these considerations, we think that the highest potential for approximate inverse techniques lies in their use as part of sophisticated solving environments for nonlinear and time-dependent problems.

The performance of sparse approximate inverse preconditioners can be further enhanced in a number of ways, the exploration of which has just begun. Among possible improvements, we mention here the use of wavelet compression techniques for PDE problems [25], the combination of sparse approximate inverse methods with approximate Schur complement and other block partitioning schemes [28], and the use of reorderings for reducing fill-in and improving the quality of factorized approximate inverses [19,22].

We also mention that very recently, parallelizable adaptive algorithms for constructing factorized approximate inverses have been suggested in [26], although it is not clear at this time whether these algorithms will result in effective preconditioners.

One drawback of sparse approximate inverse preconditioners is that they are usually far from being optimal, in the following sense. Consider the discretization (by finite differences or finite elements) of a partial differential equation. As the discretization is refined, the number of iterations for an iterative method preconditioned by a sparse approximate inverse preconditioner increases, so that the amount of work per grid point grows with problem size. This drawback is shared by other purely algebraic techniques, like diagonal scaling and standard ILU preconditioners (an interesting exception, at least for certain problems, seems to be the NGILU method [77]). As the problem size increases, all these preconditioners behave qualitatively in the same manner. In contrast, multigrid methods are optimal in the sense that the amount of work per grid point remains constant (independent of problem size). However, multigrid methods are applicable only to rather special classes of problems, whereas algebraic preconditioners can be applied to virtually any linear system $Ax = b$. Algebraic multilevel methods, which are currently the object of intense study, attempt to achieve (near) grid-independent convergence by using information from the coefficient matrix only. Often these techniques require the computation of sparse approximate Schur complements, and this is a natural application of sparse approximate inverses, which has been explored in, e.g., [28] and [26]. While some of the parallelism is lost, the combination of a multilevel approach with sparse approximate inverse techniques seems to produce robust preconditioners which are "less far from optimal" than the approximate inverse alone. Furthermore, sparse approximate inverses can be used as parallel smoothers in connection with multigrid. These are interesting developments which warrant further study. Similar considerations motivated the idea of combining discrete wavelet transforms with sparse approximate inverses [25], which results in a technique closely related to the hierarchical basis preconditioner.

In conclusion, we believe that approximate inverse techniques will play an increasingly important role for high-performance preconditioning of large-scale linear systems.

Acknowledgements

This paper is a result of a collaboration spanning a period of four years and leading to several exchange visits involving institutions in four different countries: the University of Bologna in Italy, CERFACS in Toulouse, France, the Institute of Computer Science of the Czech Academy of Sciences in Prague, Czech Republic, and Los Alamos National Laboratory in the United States. The support and hospitality of these institutions are greatly appreciated.

References

[1] G. Alléon, M. Benzi and L. Giraud, Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics, *Numer. Algorithms* 16 (1997) 1–15.

[2] F. Alvarado and H. Dağ, Sparsified and incomplete sparse factored inverse preconditioners, in: *Proceedings of the 1992 Copper Mountain Conference on Iterative Methods*, Vol. I (April 9–14, 1992).

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, *LAPACK User's Guide* (SIAM, Philadelphia, PA, 1992).

[4] O. Axelsson, A survey of preconditioned iterative methods for linear systems of algebraic equations, *BIT* 25 (1985) 166–187.
[5] O. Axelsson, *Iterative Solution Methods* (Cambridge University Press, Cambridge, 1994).
[6] O. Axelsson and L.Yu. Kolotilina, Diagonally compensated reduction and related preconditioning methods, *Numer. Linear Algebra Appl.* 1 (1994) 155–177.
[7] S.T. Barnard, L.M. Bernardo and H.D. Simon, An MPI implementation of the SPAI preconditioner on the T3E, Technical Report LBNL-40794 UC405, Lawrence Berkeley National Laboratory (1997).
[8] S.T. Barnard and R.L. Clay, A portable MPI implementation of the SPAI preconditioner in ISIS++, in: M. Heath et al., eds., *Proceedings of the Eight SIAM Conference on Parallel Processing for Scientific Computing* (SIAM, Philadelphia, PA, 1997) (CD-ROM).
[9] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (SIAM, Philadelphia, PA, 1994).
[10] M.W. Benson, Iterative solution of large scale linear systems, Master's Thesis, Lakehead University, Thunder Bay, Canada (1973).
[11] M.W. Benson and P.O. Frederickson, Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems, *Utilitas Math.* 22 (1982) 127–140.
[12] M. Benson, J. Krettmann and M. Wright, Parallel algorithms for the solution of certain large sparse linear systems, *Internat. J. Comput. Math.* 16 (1984) 245–260.
[13] M. Benzi, A direct row-projection method for sparse linear systems, Ph.D. Thesis, Department of Mathematics, North Carolina State University, Raleigh, NC (1993).
[14] M. Benzi, R. Kouhia and M. Tůma, An assessment of some preconditioning techniques in shell problems, to appear in *Comm. Numer. Methods Engrg.*
[15] M. Benzi, C.D. Meyer and M. Tůma, A sparse approximate inverse preconditioner for the conjugate gradient method, *SIAM J. Sci. Comput.* 17 (1996) 1135–1149.
[16] M. Benzi, D.B. Szyld and A. van Duin, Orderings for incomplete factorization preconditioning of nonsymmetric problems, *SIAM J. Sci. Comput.*, to appear.
[17] M. Benzi and M. Tůma, Numerical experiments with two approximate inverse preconditioners, *BIT* 38 (1998) 234–247.
[18] M. Benzi and M. Tůma, A sparse approximate inverse preconditioner for nonsymmetric linear systems, to appear in *SIAM J. Sci. Comput.* 19 (1998).
[19] M. Benzi and M. Tůma, Sparse matrix orderings for factorized inverse preconditioners, in: *Proceedings of the 1998 Copper Mountain Conference on Iterative Methods* (March 30–April 3, 1998).
[20] E. Bodewig, *Matrix Calculus, 2nd revised and enlarged edition* (Interscience, New York; North-Holland, Amsterdam, 1959).
[21] C. Brezinski, *Projection Methods for Systems of Equations* (North-Holland, Amsterdam, 1997).
[22] R. Bridson and W.-P. Tang, Ordering, anisotropy and factored sparse approximate inverses, Preprint, Department of Computer Science, University of Waterloo (February 1998).
[23] A.M. Bruaset, *A Survey of Preconditioned Iterative Methods* (Longman, Essex, England, 1995).
[24] L. Cesari, Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive, *Atti Accad. Naz. Lincei, Rend. Cl. Sci. Fis. Mat. Nat.* 25 (1937) 422–428.
[25] T. Chan, W.-P. Tang and W. Wan, Wavelet sparse approximate inverse preconditioners, *BIT* 37 (1997) 644-660.
[26] E. Chow, Robust preconditioning for sparse linear systems, Ph.D. Thesis, Department of Computer Science, University of Minnesota, Minneapolis, MN (1997).
[27] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations, *SIAM J. Sci. Comput.* 19 (1998) 995–1023.
[28] E. Chow and Y. Saad, Approximate inverse techniques for block-partitioned matrices, *SIAM J. Sci. Comput.* 18 (1997) 1657–1675.

[29] E. Chow and Y. Saad, Parallel approximate inverse preconditioners, in: M. Heath et al., eds., *Proceedings of 8th SIAM Conference on Parallel Processing for Scientific Computing* (SIAM, Philadelphia, PA, 1997) (CD-ROM).
[30] E. Chow and Y. Saad, Experimental study of ILU preconditioners for indefinite matrices, *J. Comput. Appl. Math.* 86 (1997) 387–414.
[31] M.T. Chu, R.E. Funderlic and G.H. Golub, A rank-one reduction formula and its applications to matrix factorizations, *SIAM Rev.* 37 (1995) 512–530.
[32] R. Cook, A reformulation of preconditioned conjugate gradients suitable for a local memory multiprocessor, in: R. Beauwens and P. de Groen, eds., *Iterative Methods in Linear Algebra* (IMACS, North-Holland, Amsterdam, 1992) 313–322.
[33] J.D.F. Cosgrove, J.C. Díaz and A. Griewank, Approximate inverse preconditioning for sparse linear systems, *Internat. J. Comput. Math.* 44 (1992) 91–110.
[34] H. Dağ, Iterative methods and parallel computation for power systems, Ph.D. Thesis, Department of Electrical Engineering, University of Wisconsin, Madison, WI (1996).
[35] T. Davis, Sparse matrix collection, *NA Digest* 94(42) (October 1994) `http://www.cise.ufl.edu/~davis/sparse/`.
[36] S. Demko, W.F. Moss and P.W. Smith, Decay rates for inverses of band matrices, *Math. Comp.* 43 (1984) 491–499.
[37] J.J. Dongarra and D.W. Walker, Software libraries for linear algebra computations on high performance computers, *SIAM Rev.* 37 (1995) 151–180.
[38] P.F. Dubois, A. Greenbaum and G.H. Rodrigue, Approximating the inverse of a matrix for use in iterative algorithms on vector processors, *Computing* 22 (1979) 257–268.
[39] I.S. Duff, MA28—A set of Fortran subroutines for sparse unsymmetric linear equations, Harwell Report AERE-R.8730, Harwell Laboratories, UK (1980).
[40] I.S. Duff, A.M. Erisman, C.W. Gear and J.K. Reid, Sparsity structure and Gaussian elimination, *SIGNUM Newsletter* 23 (1988) 2–9.
[41] I.S. Duff, A.M. Erisman and J.K. Reid, *Direct Methods for Sparse Matrices* (Clarendon Press, Oxford, 1986).
[42] I.S. Duff, R.G. Grimes and J.G. Lewis, Sparse matrix test problems, *ACM Trans. Math. Software* 15 (1989) 1–14.
[43] I.S. Duff and G.A. Meurant, The effect of ordering on preconditioned conjugate gradients, *BIT* 29 (1989) 635–657.
[44] H.C. Elman, A stability analysis of incomplete LU factorizations, *Math. Comp.* 47 (1986) 191–217.
[45] M.R. Field, An efficient parallel preconditioner for the conjugate gradient algorithm, Hitachi Dublin Laboratory Technical Report HDL-TR-97-175, Dublin, Ireland (1997).
[46] L. Fox, *An Introduction to Numerical Linear Algebra* (Oxford University Press, Oxford, 1964).
[47] L. Fox, H.D. Huskey and J.H. Wilkinson, Notes on the solution of algebraic linear simultaneous equations, *Quart. J. Mech. Appl. Math.* 1 (1948) 149–173.
[48] P.O. Frederickson, Fast approximate inversion of large sparse linear systems, Math. Report 7, Lakehead University, Thunder Bay, Canada (1975).
[49] R. Freund, A transpose-free quasi-minimal residual method for non-Hermitian linear systems, *SIAM J. Sci. Comput.* 14 (1993) 470–482.
[50] K.A. Gallivan, R.J. Plemmons and A.H. Sameh, Parallel algorithms for dense linear algebra computations, *SIAM Rev.* 32 (1990) 54–135.
[51] G.H. Golub and C.F. van Loan, *Matrix Computations*, 3rd ed. (Johns Hopkins University Press, Baltimore, MD, 1996).
[52] N.I.M. Gould and J.A. Scott, Sparse approximate-inverse preconditioners using norm-minimization techniques, *SIAM J. Sci. Comput.* 19 (1998) 605–625.
[53] A. Greenbaum, *Iterative Methods for Solving Linear Systems* (SIAM, Philadelphia, PA, 1997).

[54] M. Grote and T. Huckle, Parallel preconditioning with sparse approximate inverses, *SIAM J. Sci. Comput.* 18 (1997) 838–853.
[55] I. Gustafsson and G. Lindskog, Completely parallelizable preconditioning methods, *Numer. Linear Algebra Appl.* 2 (1995) 447–465.
[56] M.A. Heroux, P. Vu and C. Yang, A parallel preconditioned conjugate gradient package for solving sparse linear systems on a Cray Y-MP, *Appl. Numer. Math.* 8 (1991) 93–115.
[57] T.K. Huckle, Approximate sparsity patterns for the inverse of a matrix and preconditioning, in: R. Weiss and W. Schönauer, eds., *Proceedings of the 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics*, Vol. 2 (IMACS, New Brunswick, 1997) 569–574.
[58] O.G. Johnson, C.A. Micchelli and G. Paul, Polynomial preconditioning for conjugate gradient calculations, *SIAM J. Numer. Anal.* 20 (1983) 362–376.
[59] I.E. Kaporin, New convergence results and preconditioning strategies for the conjugate gradient method, *Numer. Linear Algebra Appl.* 1 (1994) 179–210.
[60] L.Yu. Kolotilina, Explicit preconditioning of systems of linear algebraic equations with dense matrices, *J. Soviet Math.* 43 (1988) 2566–2573. [English translation of a paper first published in *Zap. Nauchn. Sem. Leningradskogo Otdel. Mat. Inst. Steklov.* (POMI) 154 (1986) 90–100.]
[61] L.Yu. Kolotilina and A.Yu. Yeremin, Factorized sparse approximate inverse preconditioning I. Theory, *SIAM J. Matrix Anal. Appl.* 14 (1993) 45–58.
[62] L.Yu. Kolotilina and A.Yu. Yeremin, Factorized sparse approximate inverse preconditioning II. Solution of 3D FE systems on massively parallel computers, *Internat. J. High Speed Comput.* 7 (1995) 191–215.
[63] T.A. Manteuffel, An incomplete factorization technique for positive definite linear systems, *Math. Comp.* 34 (1980) 473–497.
[64] 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. Comp.* 31 (1977) 148–162.
[65] N.S. Mendelsohn, Some elementary properties of ill-conditioned matrices and linear equations, *Amer. Math. Monthly* 63 (1956) 285–295.
[66] N.S. Mendelsohn, Some properties of approximate inverses of matrices, *Trans. Roy. Soc. Canada Section III* 50 (1956) 53–59.
[67] Y. Saad, Practical use of polynomial preconditionings for the conjugate gradient method, *SIAM J. Sci. Statist. Comput.* 6 (1985) 865–881.
[68] Y. Saad, Krylov subspace methods on supercomputers, *SIAM J. Sci. Statist. Comput.* 10 (1989) 1200–1232.
[69] Y. Saad, ILUT: A dual threshold incomplete LU factorization, *Numer. Linear Algebra Appl.* 1 (1994) 387–402.
[70] Y. Saad, Preconditioned Krylov subspace methods for CFD applications, in: W.G. Habashi, ed., *Solution Techniques for Large-Scale CFD Problems* (Wiley, New York, 1995) 139–158.
[71] Y. Saad, *Iterative Methods for Sparse Linear Systems* (PWS, Boston, 1996).
[72] Y. Saad and M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 7 (1986) 856–869.
[73] W. Schönauer, H. Häfner and R. Weiss, LINSOL (LINear SOLver), Description and user's guide for the parallelized version, version 0.96, Interner Bericht Nr. 61/95 des Rechenzentrums der Universität Karlsruhe (1995).
[74] W. Schönauer, H. Häfner and R. Weiss, LINSOL, a parallel iterative linear solver package of generalized CG-type for sparse matrices, in: M. Heath et al., eds., *Proceedings of the Eight SIAM Conference on Parallel Processing for Scientific Computing* (SIAM, Philadelphia, PA, 1997) (CD-ROM).
[75] R. Suda, Large scale circuit analysis by preconditioned relaxation methods, in: *Proceedings of PCG '94*, Keio University, Japan (1994) 189–205.
[76] R. Suda, New iterative linear solvers for parallel circuit simulation, Ph.D. Thesis, Department of Information Sciences, University of Tokyo (1996).

[77] A. van der Ploeg, E.F.F. Botta and F.W. Wubs, Nested grids ILU-decomposition (NGILU), *J. Comput. Appl. Math.* 66 (1996) 515–526.
[78] H.A. van der Vorst, A vectorizable variant of some ICCG methods, *SIAM J. Sci. Statist. Comput.* 3 (1982) 350–356.
[79] H.A. van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems, *SIAM J. Sci. Statist. Comput.* 12 (1992) 631–644.
[80] A.C.N. van Duin and H. Wijshoff, Scalable parallel preconditioning with the sparse approximate inverse of triangular systems, Preprint, Computer Science Department, University of Leiden, Leiden, The Netherlands (1996).
[81] R. Weiss, *Parameter-Free Iterative Linear Solvers*, Mathematical Research, Vol. 97 (Akademie Verlag, Berlin, 1996).
[82] A.Yu. Yeremin, L.Yu. Kolotilina and A.A. Nikishin, Factorized sparse approximate inverse preconditionings III. Iterative construction of preconditioners, Research Report 11/97, Center for Supercomputing and Massively Parallel Applications, Computer Center of Russian Academy of Sciences (1997).
[83] Z. Zlatev, *Computational Methods for General Sparse Matrices* (Kluwer, Dordrecht, 1991).