A Sparse Approximate Inverse Preconditioner for Nonsymmetric Linear Systems

Benzi, M.
1995

A SPARSE APPROXIMATE INVERSE PRECONDITIONER FOR NONSYMMETRIC LINEAR SYSTEMS

MICHELE BENZI * AND MIROSLAV TŮMA †

Abstract. This paper is concerned with a new approach to preconditioning for large, sparse linear systems. A procedure for computing an incomplete factorization of the inverse of a nonsymmetric matrix is developed, and the resulting factorized sparse approximate inverse is used as an explicit preconditioner for conjugate gradient-type methods. Some theoretical properties of the preconditioner are discussed, and numerical experiments on test matrices from the Harwell-Boeing collection and from Tim Davis' collection are presented. Our results indicate that the new preconditioner is cheaper to construct than other approximate inverse preconditioners. Furthermore, the new technique insures convergence rates of the preconditioned iteration which are comparable with those obtained with standard implicit preconditioners.

Key words. Preconditioning, approximate inverses, sparse linear systems, sparse matrices, incomplete factorizations, conjugate gradient-type methods.

AMS(MOS) subject classification. 65F10, 65F35, 65F50, 65Y05 .

1. Introduction.

In this paper we consider the solution of nonsingular linear systems of the form

$$Ax = b \tag{1}$$

where the coefficient matrix $A \in \mathbb{R}^{n\times n}$ is large and sparse. In particular, we are concerned with the development of preconditioners for conjugate gradient-type methods. It is well-known that the rate of convergence of such methods for solving (1) is strongly influenced by the spectral properties of A. It is therefore natural to try to transform the original system into one having the same solution but more favorable spectral properties. A *preconditioner* is a matrix that can be used to accomplish such a transformation. If G is a nonsingular

* Dipartimento di Matematica, Università di Bologna, Italy and CERFACS, 42 Ave. G. Coriolis, 31057 Toulouse Cedex, France (benzi@cerfacs.fr). This work was supported in part by a grant under the scientific cooperation agreement between the CNR and the Czech Academy of Sciences.

† Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, 182 07 Prague 8 - Libeň, Czech Republic (tuma@uivt.cas.cz). The work of this author was supported in part by grants GA CR No. 201/93/0067 and GA AS CR No. 230401 and by NSF under grant number INT-9218024.

matrix which approximates A^{-1} ($G \approx A^{-1}$), the transformed linear system

$$GAx = Gb \tag{2}$$

will have the same solution as system (1) but the convergence rate of iterative methods applied to (2) may be much higher. Problem (2) is preconditioned from the left, but *right* preconditioning is also possible. Preconditioning on the right leads to the transformed linear system

$$AGy = b. \tag{3}$$

Once the solution y of (3) has been obtained, the solution of (1) is given by $x = Gy$. The choice between left or right preconditioning is often dictated by the choice of the iterative method. It is also possible to use both forms of preconditioning at once (*split* preconditioning), see [3] for further details.

Note that in practice it is not required to compute the matrix product GA (or AG) explicitly, because conjugate gradient-type methods only necessitate the coefficient matrix in the form of matrix-vector multiplies. Therefore, applying the preconditioner within a step of a gradient-type method reduces to computing the action of G on a vector.

Loosely speaking, the closer G is to the exact inverse of A, the higher the rate of convergence of iterative methods will be. Choosing $G = A^{-1}$ yields convergence in one step, but of course constructing such a preconditioner is equivalent to solving the original problem. In practice, the preconditioner G should be easily computed and applied, so that the total time for the preconditioned iteration is less than the time for the unpreconditioned one. Typically, the cost of applying the preconditioner at each iteration of a conjugate gradient-type method should be of the same order as the cost of a matrix-vector multiply involving A. For a sparse A, this implies that the preconditioner should also be sparse with a density of nonzeros roughly of the same order as that of A.

Clearly, the effectiveness of a preconditioning strategy is strongly problem and architecture dependent. For instance, a preconditioner which is expensive to compute may become viable if it is to be reused many times, since in this case the initial cost of forming the preconditioner can be amortized over several linear systems. This situation occurs, for instance, when dealing with time-dependent or nonlinear problems, whose numerical solution gives rise to long sequences of linear systems having the same coefficient matrix (or a slowly varying one) and different right-hand sides. Furthermore, preconditioners that are very efficient in a scalar computing environment may show poor performance on vector and parallel machines, and conversely.

A number of preconditioning techniques have been proposed in the literature (see, e.g., [2],[3] and the references therein). While it is generally agreed that the construction of efficient general-purpose preconditioners is not possible, there is still considerable interest in developing methods which will perform well on a wide range of problems and are well-suited for state-of-the-art computer architectures. Here we introduce a new algebraic preconditioner based on an incomplete triangular factorization of A^{-1}. This paper is the natural continuation of [8], where the focus was restricted to symmetric positive definite systems and to the preconditioned conjugate gradient method (see also [5],[7]).

The paper is organized as follows. In §2 we give a quick overview of implicit and explicit preconditioning techniques, considering the relative advantages as well as the limitations of the two approaches. In §3 we summarize some recent work on the most popular approach to approximate inverse preconditioning, based on Frobenius norm minimization. In §4 we introduce the new incomplete inverse triangular decomposition technique and describe some of its theoretical properties. A graph-theoretical characterization of fill-in in the inverse triangular factorization is presented in §5. In §6 we consider the use of preconditioning on matrices which have been reduced to block triangular form. Implementation details and the results of numerical experiments are discussed in §§7 and 8, and some concluding remarks and indications for future work are given in §9. Our experiments suggest that the new preconditioner is cheaper to construct than preconditioners based on the optimization approach. Moreover, good rates of convergence can be achieved by our preconditioner, comparable with those insured by standard ILU-type techniques.

2. Explicit vs. implicit preconditioning. Most existing preconditioners can be broadly classified as being either of the *implicit* or of the *explicit* kind. A preconditioner is implicit if its application, within each step of the chosen iterative method, requires the solution of a linear system. A nonsingular matrix $M \approx A$ implicitly defines an approximate inverse $G := M^{-1} \approx A^{-1}$, and applying G requires solving a linear system with coefficient matrix M. Of course, M should be chosen so that solving a system with matrix M is easier than solving the original problem (1). Perhaps the most important example is provided by preconditioners based on an Incomplete LU (ILU) decomposition. Here $M = \bar{L}\bar{U}$ where $\bar{L}$ and $\bar{U}$ are sparse triangular matrices which approximate the exact L and U factors of A. Applying the preconditioner requires the solution of two sparse triangular systems (the forward and backward solves). Other notable examples of implicit preconditioners include the ILQ, SSOR and ADI preconditioners, see [3].

In contrast, with explicit preconditioning a matrix $G \approx A^{-1}$ is known (possibly as the product of sparse matrices) and the preconditioning operation reduces to forming one (or more) matrix-vector product. For instance, many polynomial preconditioners belong to this class [37]. Other explicit preconditioners will be described in the subsequent sections.

Implicit preconditioners have been intensively studied, and they have been successfully employed in a number of applications. In spite of this, in the last few years an increasing amount of attention has been devoted to alternative forms of preconditioning, especially of the explicit kind. There have been so far two main reasons for this recent trend. In the first place, shortly after the usage of modern high-performance architectures became widespread, it was realized that straightforward implementation of implicit preconditioning in conjugate gradient-like methods could lead to severe degradation of the performance on the new machines. In particular, the sparse triangular solves involved in ILU-type preconditioning were found to be a serial bottleneck (due to the recursive nature of the computation), thus limiting the effectiveness of this approach on vector and parallel computers. It should be mentioned that considerable effort has been devoted to overcoming this difficulty. As a result, for some architectures and types of problems it is possible to introduce nontrivial parallelism and to achieve reasonably good performance in the triangular solves by means of suitable reordering strategies (see, e.g., [1],[38],[54]). However, the triangular solves remain the most problematic aspect of the computation, both on shared memory [33] and distributed memory [10] computers, and for many problems the efficient application of an implicit preconditioner in a parallel environment still represents a serious challenge.

Another drawback of implicit preconditioners of the ILU-type is the possibility of breakdowns during the incomplete factorization process, due to the occurrence of zero or exceedingly small pivots. This situation typically arises when dealing with matrices which are strongly unsymmetric and/or indefinite, even if pivoting is applied (see [11],[49]), and in general it may even occur for definite problems unless A exhibits some degree of diagonal dominance. Of course, it is always possible to safeguard the incomplete factorization process so that it always runs to completion, producing a nonsingular preconditioner, but there is also no guarantee that the resulting preconditioner will be of acceptable quality. Furthermore, as shown in [23], there are problems for which standard ILU techniques produce unstable incomplete factors, resulting in useless preconditioners.

Explicit preconditioning techniques, based on directly approximating A^{-1}, have been developed in an attempt to avoid or mitigate such difficulties. Applying an explicit preconditioner only requires sparse matrix-vector products, which should be easier to parallelize

than the sparse triangular solves, and in some cases the construction of the preconditioner itself is well-suited for parallel implementation. In addition, the construction of an approximate inverse is sometimes possible even if the matrix does not have a stable incomplete LU decomposition. Moreover, we mention that sparse incomplete inverses are often used when constructing approximate Schur complements (pivot blocks) for use in incomplete block factorization and other two-level preconditioners, see [2],[3],[12],[15].

Of course, explicit preconditioners are far from being completely trouble-free. Even if a sparse approximate inverse G is computed, care must be exercised to ensure that G is nonsingular. For nonsymmetric problems, the same matrix G could be a good approximate inverse if used for left preconditioning and a poor one if used for right preconditioning, see [36, p. 96],[45, p. 66],[48]. Furthermore, explicit preconditioners are sometimes not as effective as implicit ones at reducing the number of iterations, in the sense that there are problems for which they require a higher number of nonzeros in order to achieve the same rate of convergence insured by implicit preconditioners. One of the reasons for this limitation is that an explicit preconditioner attempts to approximate A^{-1}, which is usually dense, with a sparse matrix. Thus, an explicit preconditioner is more likely to work well if A^{-1} contains many entries which are small (in magnitude). A favorable situation is when A exhibits some form of diagonal dominance, but for such problems implicit preconditioning is also likely to be very effective. Hence, for problems of this type, explicit preconditioners can be competitive with implicit ones only if explicitness is fully exploited. Finally, explicit preconditioners are usually more expensive to compute than implicit ones, although this difference may become negligible in the common situation where several linear systems with the same coefficient matrix and different right-hand sides have to be solved. In this case the time for computing the preconditioner is often only a fraction of the time required for the overall computation. It is also worth repeating that the construction of certain sparse approximate inverses can be done, at least in principle, in a highly parallel manner, whereas the scope for parallelism in the construction of ILU-type preconditioners is more limited.

3. Methods based on Frobenius norm minimization. A good deal of work has been devoted to explicit preconditioning based on the following approach: the sparse approximate inverse is computed as the matrix G which minimizes $\|I - GA\|$ (or $\|I - AG\|$ for right preconditioning) subject to some sparsity constraint (see [4], Ch. 8 of [2],[16],[43], [44],[32],[31],[11],[30]). Here the matrix norm is usually the Frobenius norm or a weighted variant of it, for computational reasons. With this choice, the constrained minimization problem decouples into n independent linear least squares problems (one for each row, or

column of G), the number of unknowns for each problem being equal to the number of nonzeros allowed in each row (or column) of G. This immediately follows from the identity

$$\|I - AG\|_F^2 = \sum_{i=1}^{n} \|e_i - Ag_i\|_2^2$$

where e_i is the ith unit vector and g_i is the ith column of G. Clearly, there is considerable scope for parallelism in this approach. The resulting sparse least squares problems can be solved, in principle, independently of each other, either by direct methods (as in [44], [31],[30]) or iteratively ([11],[42]).

In early papers (e.g. [4],[32],[43]) the sparsity constraint was imposed a priori, and the minimizer was found relative to a class of matrices with a predetermined sparsity pattern. For instance, when A is a band matrix with a good degree of diagonal dominance, a banded approximation to A^{-1} is justified, see [18]. However, for general sparse matrices it is very difficult to guess a good sparsity pattern for an approximate inverse, and several recent papers have addressed the problem of adaptively defining the nonzero pattern of G in order to capture "large" entries of the inverse [31],[30]. Indeed, by monitoring the size of each residual $\|e_i - Ag_i\|_2$ it is possible to decide whether new entries of g_i are to be retained or discarded, in a dynamic fashion. Moreover, the information on the residuals can be utilized to derive rigorous bounds on the clustering of the singular values of the preconditioned matrix and therefore to estimate its condition number [31]. It is also possible to formulate conditions on the norm of the residuals which insure that the approximate inverse will be nonsingular. Unfortunately, such conditions appear to be of dubious practical value, because trying to fulfill them could lead to a very dense approximate inverse [16],[11].

A disadvantage of this approach is that symmetry in the coefficient matrix cannot be exploited. If A is symmetric positive definite (SPD), the sparse approximate inverse will not be symmetric in general. Even if a preset, symmetric sparsity pattern is enforced, there is no guarantee that the approximate inverse will be positive definite. This could lead to a breakdown in the conjugate gradient acceleration. For this reason, Kolotilina and Yeremin [43],[44] propose to compute an explicit preconditioner of the form $G = G_L^T G_L$ where G_L is lower triangular. The preconditioned matrix is then $G_L A G_L^T$, which is SPD, and the conjugate gradient method can be applied. The matrix G_L is the solution of a constrained minimization problem for the Frobenius norm of $I - LG_L$ where L is the Cholesky factor of A. In [43] it is shown how this problem can be solved without explicit knowledge of any of the entries of L, using only entries of the coefficient matrix A. The same technique can also be used to compute a factorized approximate inverse of a nonsymmetric matrix by

separately approximating the inverses of the L and U factors. As it stands, however, this technique requires that the sparsity pattern of the approximate inverse triangular factors be specified in advance, and therefore is not suitable for matrices with a general sparsity pattern.

There are additional reasons for considering factorized approximate inverses. Clearly, with the approximate inverse G expressed as the product of two triangular factors it is trivial to insure that G is nonsingular. Another argument in favor of this approach is given in [11], where it is observed that factorized forms of general sparse matrices contain more information for the same storage than if a single product was stored.

The serial cost for the construction of this type of preconditioner is usually very high, although the theoretical parallel complexity can be quite moderate [44],[30]. The results of numerical experiments reported in [44] demonstrate that factorized sparse approximate inverse preconditioners can insure rapid convergence of the preconditioned conjugate gradient (PCG) iteration when applied to certain finite element discretizations of 3D PDE problems arising in elasticity theory. However, in these experiments the preconditioning strategy is not applied to the coefficient matrix directly, but rather to a reduced system (Schur complement) which is better conditioned and considerably less sparse than the original problem. When the approximate inverse preconditioner is applied directly to the original stiffness matrix A, the rate of convergence of the PCG iteration is rather disappointing.

A comparison between a Frobenius norm-based sparse approximate inverse preconditioner and the ILU(0) preconditioner on a number of general sparse matrices has been made in [30]. The reported results show that the explicit preconditioner can insure rates of convergence comparable with those achieved with the implicit ILU-type approach. Again, it is observed that the construction of the approximate inverse is often very costly, but amenable to parallelization.

Factorized sparse approximate inverses have also been considered by other authors, for instance by Kaporin [39],[40],[41], whose approach is also based on minimizing a certain matrix functional and is closely related to that of Kolotilina and Yeremin. In the next sections we present an alternative approach to factorized sparse approximate inverse preconditioning which is not grounded in optimization, but is based instead on a direct method of matrix inversion. As we shall see, the serial cost of forming a sparse approximate inverse with this technique is usually much less than with the optimization approach, while the convergence rates are still comparable, on average, with those obtained with ILU-type preconditioning.

4. A method based on inverse triangular factorization. The optimization approach to constructing approximate inverses is not the only possible one. In this section we consider an alternative procedure based on a direct method of matrix inversion, performed incompletely in order to preserve sparsity. This results in a factorized sparse $G \approx A^{-1}$. Being an incomplete matrix factorization method, our procedure resembles classical ILU-type implicit techniques, and indeed we can draw from the experience accumulated in years of use of ILU-type preconditioning both at the implementation stage and when deriving theoretical properties of the preconditioner G. This paper continues the work in [8], where the symmetric positive definite case was studied (see also [5],[7]).

The construction of our 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 $w_i^T A z_j = 0$ 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

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

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

$$A^{-1} = Z D^{-1} W^T = \sum_{i=1}^{n} \frac{z_i w_i^T}{p_i} \quad . \tag{4}$$

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 $W^{(0)} = Z^{(0)} = I_{n\times n}$: 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.

THE BICONJUGATION ALGORITHM

(1) Let $w_i^{(0)} = z_i^{(0)} = e_i \quad (1 \le i \le n)$

(2) **for** $i = 1, 2, \ldots, n$

for $j = i, i+1, \ldots, n$

$$p_j^{(i-1)} := a_i^T z_j^{(i-1)} ; \qquad q_j^{(i-1)} := c_i^T w_j^{(i-1)}$$

end

if $i = n$ **go to** (3)

for $j = i+1, \ldots, n$

$$z_j^{(i)} := z_j^{(i-1)} - \left(\frac{p_j^{(i-1)}}{p_i^{(i-1)}}\right) z_i^{(i-1)} ; \qquad w_j^{(i)} := w_j^{(i-1)} - \left(\frac{q_j^{(i-1)}}{q_i^{(i-1)}}\right) w_i^{(i-1)}$$

end

end

(3) Let $z_i := z_i^{(i-1)}$, $w_i := w_i^{(i-1)}$ and $p_i := p_i^{(i-1)}$, for $1 \le i \le n$.

Return $Z = [z_1, z_2, \ldots, z_n]$, $W = [w_1, w_2, \ldots, w_n]$ and $D = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n \end{pmatrix}$.

This algorithm is essentially due to L. Fox, see Ch. 6 of [25]. Closely related methods have also been considered by Hestenes and Stiefel [35, pp. 426–427],[34] and by Stewart [52]. A more general treatment is given in the recent paper [14]. Geometrically, the procedure can be regarded as a generalized Gram-Schmidt orthogonalization with oblique projections and nonstandard inner products, see [6],[14].

Several observations regarding this algorithm are in order. In the first place we note that the above formulation contains some redundancy, since in exact arithmetic

$$p_i = w_i^T A z_i = z_i^T A^T w_i = q_i.$$

Therefore, at step i the computation of the dot product $q_i^{(i-1)} = c_i^T w_i^{(i-1)}$ may be replaced by the assignment $q_i^{(i-1)} := p_i^{(i-1)}$. Another observation is the fact that the procedure, as it stands, is vulnerable to breakdown (division by zero), which occurs whenever any of the

quantities $p_i^{(i-1)}$ $(= q_i^{(i-1)})$ happens to be zero. It can be shown that in exact arithmetic, the biconjugation algorithm will not break down if and only if all the leading principal minors of A are nonzero (see below). For any nonsingular matrix A there exists a permutation matrix P (or Q) such that the procedure applied to PA (or to AQ) will not break down. As in the LU decomposition with pivoting, such permutation matrices represent row (or column) interchanges on A which can be performed, if needed, in the course of the computation.

If the biconjugation process can be carried to completion without interchanges, the resulting Z and W matrices are upper triangular,[1] they both have all diagonal entries equal to one, and satisfy the identity

$$A = W^{-T} D Z^{-1}. \tag{5}$$

We recognize in (5) the familiar LDU decomposition $A = LDU$, where L is unit lower triangular, U is unit upper triangular and D is the diagonal matrix with the pivots down the main diagonal. Because this factorization is unique, we have that the biconjugation algorithm explicitly computes

$$W = L^{-T}, \qquad Z = U^{-1}$$

and the matrix D, which is exactly the same in (5) and in $A = LDU$. Hence, the process produces an inverse triangular decomposition of A or, equivalently, a triangular decomposition (of the UDL type) of A^{-1}. The p_i's returned by the algorithm are the pivots in the LDU factorization of A. If we denote by Δ_i the ith leading principal minor of A $(1 \leq i \leq n)$ and let $\Delta_0 = 1$, the identity (5) implies that

$$p_i = \frac{\Delta_i}{\Delta_{i-1}} \qquad (i = 1, \ldots, n)$$

showing that the biconjugation algorithm can be performed without breakdowns if and only if all leading principal minors of A are non-vanishing. In finite precision arithmetic, pivoting may be required to promote numerical stability.

Once Z, W and D are available, the solution of a linear system of the form (1) can be computed, by (4), as

$$x = A^{-1}b = ZD^{-1}W^T b = \sum_{i=1}^{n} \left(\frac{w_i^T b}{p_i} \right) z_i. \tag{6}$$

[1] Note that this is not necessarily true when a matrix other than the identity is used at the outset, i.e. if $Z^{(0)}, W^{(0)} \neq I_{n \times n}$.

In practice, this direct method for solving linear systems is not used on account of its cost: for a dense $n \times n$ matrix, the biconjugation process requires about twice the work as the LU factorization of A. Notice that the cost of the solve phase using (6) is the same as for the forward and backward solves with the L and U factors.

If A is symmetric, the number of operations in the biconjugation algorithm can be halved by observing that W must equal Z. Hence, the process can be carried out using only the rows of A, the z-vectors and the $p_j^{(i-1)}$. The columns of the resulting Z form a set of conjugate directions for A. If A is SPD, no breakdown can occur (in exact arithmetic), so that pivoting is not required and the algorithm computes the $L^T D L$ factorization of A^{-1}. This method was first described in [26]. Geometrically, it amounts to Gram-Schmidt orthogonalization with inner product $\langle x,\ y\rangle := x^T A y$ applied to the unit vectors $e_1, \ldots, e_n$. It is sometimes referred to as the *conjugate Gram-Schmidt process*. The method is still impractical as a direct solver because it requires about twice the work of Cholesky for dense matrices. However, as explained in [5] and [6], the same algorithm can also be applied to nonsymmetric systems, resulting in an implicit LDU factorization where only $Z = U^{-1}$ and D are computed. Indeed, it is possible to compute a solution to (1) for any right-hand side using just Z, D and part of the entries of A. This method has the same arithmetic complexity as Gaussian elimination when applied to dense problems. When combined with suitable sparsity-preserving strategies the method can be useful as a sparse direct solver, at least for some types of problems (see [5],[6]).

For a sparse symmetric and positive definite A, the Z matrix produced by the algorithm tends to be dense (see the next section), but it can be observed experimentally that very often, most of the entries in Z have small magnitude. If fill-in in the Z matrix is reduced by removing suitably small entries in the computation of the z-vectors, the algorithm computes a sparse matrix $\bar{Z}$ and a diagonal matrix $\bar{D}$ such that

$$G := \bar{Z}\,\bar{D}^{-1}\,\bar{Z}^T \approx A^{-1}$$

(i.e., a factorized sparse approximate inverse of A). Hence, G can be used as an explicit preconditioner for the conjugate gradient method. A detailed study of this preconditioning strategy for SPD problems can be found in [8], where it is proven that the incomplete inverse factorization exists if A is an H-matrix (analogously to ILU-type factorizations). The numerical experiments in [8] show that this approach can insure fast convergence of the PCG iteration, almost as good as with implicit preconditioning of the incomplete Cholesky type. The construction of the preconditioner itself, while somewhat more expensive than the computation of the incomplete Cholesky factorization, is still quite cheap. This is in contrast

with the least squares approach described in the previous section, where the construction of the approximate inverse is usually very time consuming, at least in a sequential environment.

In the remainder of this paper we consider an explicit preconditioning strategy based on the biconjugation process described above. Sparsity in the Z and W factors of A^{-1} is preserved by removing "small" fill in the z- and w-vectors. A possibility would be to drop all newly added fill-in elements outside of a preset sparsity pattern above the main diagonal in Z and W; however, for general sparse matrices it is very difficult to guess a reasonable sparsity pattern, and a drop tolerance is used instead. A trivial extension of the results in [8] shows that the incomplete biconjugation process (incomplete inverse factorization) cannot break down, in exact arithmetic, if A is an H-matrix. For more general matrices it is necessary to safeguard the computation in order to avoid breakdowns. This requires pivot modifications and perhaps some form of pivoting —we postpone the details to §7. The incomplete biconjugation algorithm computes sparse unit upper triangular matrices $\bar{Z} \approx Z$, $\bar{W} \approx W$ and a nonsingular diagonal matrix $\bar{D} \approx D$ such that

$$G := \bar{Z}\bar{D}^{-1}\bar{W}^T \approx A^{-1}$$

is a factorized sparse approximate inverse of A which can be used as an explicit preconditioner for conjugate gradient-type methods for the solution of (1).

We conclude this section with a few remarks on properties of the approximate inverse preconditioner G just described. If A is not an H-matrix, as already mentioned, the construction of the preconditioner could break down due to the occurrence of zero or extremely small pivots. However, following [46], we note that there always exists $\alpha > 0$ such that $A + \alpha I$ is diagonally dominant, and hence an H-matrix. Therefore, if the incomplete biconjugation algorithm breaks down, one could try to select $\alpha > 0$ and re-attempt the process on the shifted matrix $A' = A + \alpha I$. Clearly, α should be large enough to insure the existence of the incomplete inverse factorization, but also small enough so that A' is close to A. This approach has several drawbacks: for ill-conditioned matrices, the quality of the resulting preconditioner is typically poor; furthermore, the breakdown that prompts the shift may occur near the end of the biconjugation process, and the preconditioner may have to be recomputed several times before a satisfactory value of α is found. A better strategy is to perform diagonal modifications only as the need arises, shifting pivots away from zero if their magnitude is less than a specified threshold (see §7 for details).

If A is an M-matrix, it follows from the results in [8] that G is a nonnegative matrix.

Moreover, it is easy to see that componentwise the following inequalities hold:

$$D_A^{-1} \leq G \leq A^{-1} \tag{7}$$

where D_A is the diagonal part of A. Furthermore, if G_1 and G_2 are two approximate inverses of the M-matrix A produced by the incomplete biconjugation process and the drop tolerance used for G_1 is greater than or equal to the drop tolerance used for G_2, then

$$D_A^{-1} \leq G_1 \leq G_2 \leq A^{-1}. \tag{8}$$

The same is true if sparsity patterns are used to determine the nonzero structure in $\bar{Z}$ and $\bar{W}$ and the patterns for G_2 include the patterns for G_1. This monotonicity property is shared by other sparse approximate inverses, see for example Ch. 8 in [2]. We note that property (7) is important if the approximate inverse is to be used within an incomplete block factorization of an M-matrix A, because it insures that all the intermediate matrices produced in the course of the incomplete factorization preserve the M-matrix property (see [2, pp. 263–264]).

Finally, after discussing the similarities, we point to a difference between our incomplete inverse factorization and the ILU-type factorization of a matrix. The incomplete factorization of an M-matrix A induces a splitting $A = \bar{L}\bar{U} - R$ which is a *regular* splitting, and therefore convergent: $\rho(I - \bar{U}^{-1}\bar{L}^{-1}A) < 1$, where $\rho(B)$ denotes the spectral radius of a matrix B (see [47],[55]). The same is not true, in general, for our incomplete factorization. If one considers the induced splitting $A = G^{-1} - S$ (where $S = G^{-1} - A$) this splitting need not be convergent. An example is given by the symmetric M-matrix

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}.$$

For this matrix, the incomplete inverse factorization with a drop tolerance $T = 0.5$ (whereby intermediate fill-in is dropped if smaller than T in absolute value) produces an approximate inverse G such that $\rho(I - GA) \approx 1.215 > 1$. This shows that the approximate decomposition

$$A \approx \bar{W}^{-T}\bar{D}\bar{Z}^{-1}$$

cannot be obtained, in general, from an incomplete factorization of A. In this sense, the incomplete inverse factorization is not algebraically equivalent to an incomplete LDU factorization performed on A.

5. Fill-in in the biconjugation algorithm. In this section we give a characterization of the fill-in occurring in the factorized inverse obtained by the biconjugation algorithm. These results may serve as a guideline to predict the structure of the factorized approximate inverse, and have an impact on certain aspects of the implementation.

It is well-known that structural nonzeros in the inverse matrix A^{-1} can be characterized by the paths in the graph of the original matrix A (see [24],[29]). The following lemma states necessary and sufficient conditions for a new entry (fill-in) to be added in one of the z-vectors at the ith step of the biconjugation algorithm. A similar result holds for the w-vectors. We make use of the standard no-cancellation assumption.

LEMMA 5.1. *Let* $1 \le i < j \le n$, $1 \le l \le n$. *Then*

$$z_{lj}^{(i-1)} = 0 \ \wedge \ z_{lj}^{(i)} \neq 0$$

if and only if $l \le i$, $z_{li}^{(i-1)} \neq 0$ *and, at the same time, at least one of the two following conditions holds:*

- $a_{ij} \neq 0$,
- $(\exists k < i)(a_{ik} \neq 0 \ \wedge \ z_{kj}^{(i-1)} \neq 0)$.

Proof. Suppose that $z_{lj}^{(i-1)} = 0 \ \wedge \ z_{lj}^{(i)} \neq 0$. Directly from the update formula for the z-vectors we see that $z_{li}^{(i-1)} \neq 0$ and $l \le i$, since $z_{pi}^{(i-1)} = 0$ for $p > i$. Also, if $z_{lj}^{(i)}$ becomes nonzero in the ith step then clearly $p_j^{(i-1)}$ must be nonzero. But

$$p_j^{(i-1)} = a_i^T z_j^{(i-1)} = a_{ij} + \sum_{k<i} z_{kj}^{(i-1)} a_{ik}$$

and we get the result. The opposite implication is trivial. □

Figures 5.1 through 5.6 provide an illustration of the previous lemma. Figure 5.1 shows the nonzero structure of the matrix FS760 1 of order $n = 760$ from the Harwell-Boeing collection [21]. Figures 5.2–6 show the structure of the factor Z at different stages of the biconjugation algorithm. These pictures show that in the initial steps, when most of the entries of Z are still zero, the nonzeros in Z are induced by nonzeros in the corresponding positions of A. A similar situation occurs, of course, for the process which computes W. In Figure 5.7 we show the entries of Z which are larger (in absolute value) than 10^{-10} and in Figure 5.8 we show the incomplete factor $\bar{Z}$ obtained with drop tolerance $T = 10^{-10}$. It can be seen how well the incomplete process is able to capture the "large" entries in the complete factor Z. The figures were generated using the routines for plotting sparse matrix patterns from SPARSKIT [50].

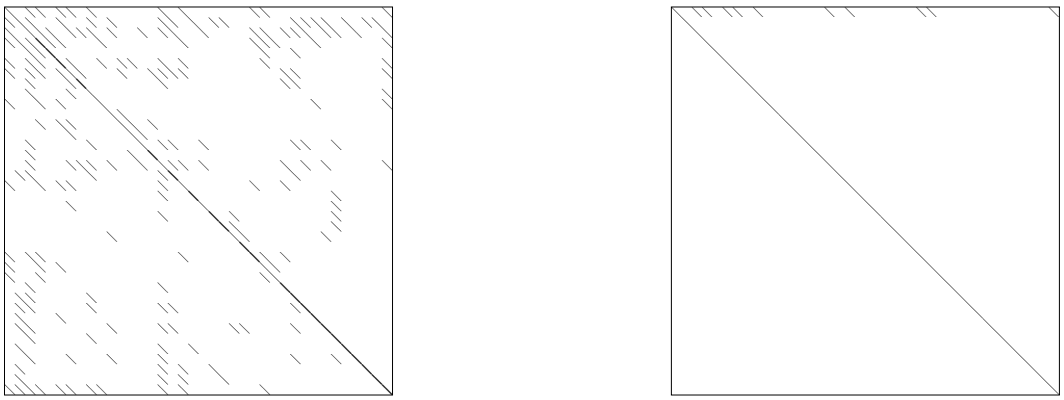

Figure 5.1-2: Structure of the matrix FS760 1 (left) and of the factor Z (right) after 20 steps of the biconjugation process.

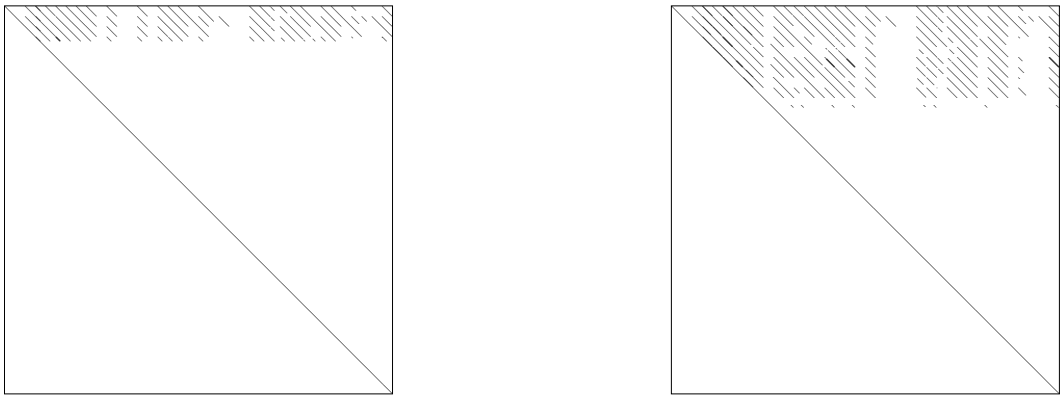

Figure 5.3-4: Structure of Z after 70 steps (left) and 200 steps (right) of the biconjugation process.

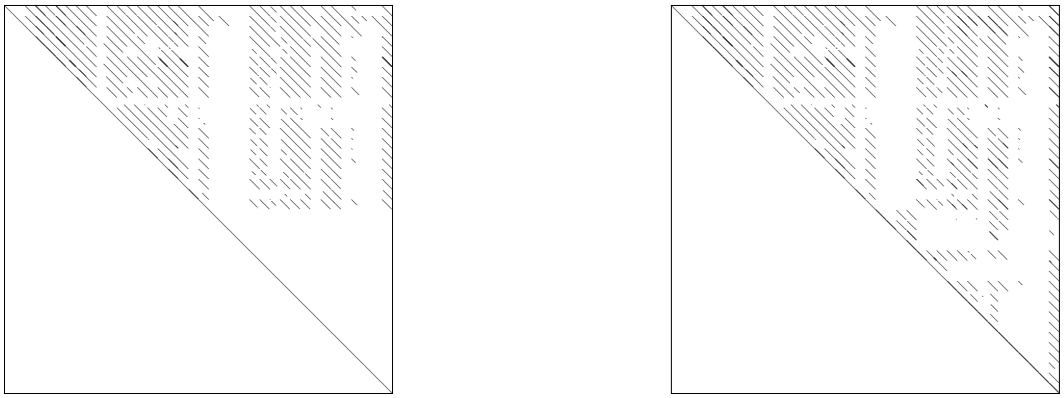

Figure 5.5-6: Structure of Z after 400 steps (left) and 760 steps (right) of the biconjugation process.

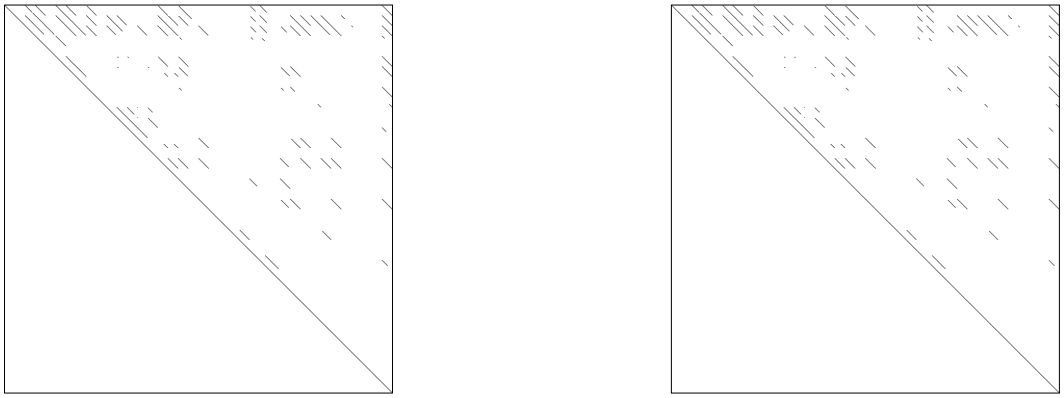

Figure 5.7-8: Structure of entries in Z larger than 10^{-10} (left) and structure of incomplete factor $\bar{Z}$ with drop tolerance $T = 10^{-10}$ (right).

A sufficient condition to have a fill-in in the matrix Z after some steps of the biconjugation algorithm is given by the following Lemma.

LEMMA 5.2. *Let $B = (R, C, E)$ be a bipartite graph with $|R| = |C| = n$ and such that for $1 \leq j, k \leq n$*

$$\{r_j, c_k\} \in E \Longleftrightarrow (a_{jk} \neq 0 \vee j = k).$$

If for some indices i_l, $1 \leq l \leq p$, $0 < i_1 < \cdots < i_p < j \leq n$ there is a path $(c_j, r_{i_1}, c_{i_1}, \ldots, r_{i_p}, c_{i_p})$ in B, then $z_{i_p j}^{(i_p)} \neq 0$.

Proof. We use induction on p. Let $p = 1$. Since $\{r_{i_1}, c_j\} \in E$ then $a_{i_1 j} \neq 0$. Of course, $z_{i_1 i_1}^{(i_1 - 1)} = 1 \neq 0$ and from Lemma 5.1 we get $z_{i_1 j}^{(i_1)} \neq 0$.

Suppose now that Lemma 5.2 is true for all $l < p$. Then, $z_{i_{p-1} j}^{(i_{p-1} - 1)} \neq 0$. But also $a_{i_p i_{p-1}} \neq 0$ since $\{r_{i_p}, c_{i_{p-1}}\} \in E$. Then $z_{i_p j}^{(i_{p-1})} \neq 0$ and using the no-cancellation assumption we also have $z_{i_p j}^{(i_p)} \neq 0$. □

The following theorem gives a necessary and sufficient condition for a nonzero entry to appear in position $(l, j), l < j$, in the inverse triangular factor.

THEOREM 5.3. *Let $1 \leq l < j \leq n$. Then $z_{lj} \neq 0$ if and only if for some $p \geq 1$ there are indices i_k, l_k, $1 \leq k \leq p$, such that $1 \leq i_1 < \cdots < i_p \leq j - 1$, $l_q < i_q$ for $1 \leq q \leq p - 1$, $l_p = l$, $a_{i_1 j} \neq 0$, $a_{i_{k+1} l_k} \neq 0$ for $1 \leq k \leq p - 1$ and $z_{l_k i_k} \neq 0$ for $1 \leq k \leq p$.*

Proof. We first show that the stated conditions are sufficient. By Lemma 5.1, the nonzeros $a_{i_1 j}$ and $z_{l_1 i_1}$ imply that $z_{l_1 j}^{(i_1)}$ is also nonzero. If $p = 1$ we are done. Otherwise, $z_{l_1 j}^{(i_2 - 1)} \neq 0$ and $a_{i_2 l_1} \neq 0$ imply $p_j^{(i_2 - 1)} \neq 0$. Taking into account that $z_{l_2 i_2} \neq 0$ we get that $z_{l_2 j}^{(i_2)}$ is nonzero. Repeating these arguments inductively we finally get $z_{l_p j}^{(i_p)} \neq 0$. Consequently, $z_{lj}^{(i)} \neq 0$.

Assume now that $z_{lj} \neq 0$. Lemma 5.1 implies that at least one of the following two conditions holds: either there exists $i', 1 \leq i' \leq i$ such that $a_{i' j} \neq 0$ and $z_{li'} \neq 0$, or there exist indices $i'', 1 \leq i'' \leq i$ and $k < i''$ such that $a_{i'' k} \neq 0$, $z_{kj}^{(i'' - 1)} \neq 0$ and $z_{li''} \neq 0$. In the former case we have the necessary conditions. In the latter case we can apply Lemma 5.1 inductively to $z_{kj}^{(i'' - 1)}$. After at most j inductive steps we obtain the conditions. □

Clearly, the characterization of fill-in in the inverse triangular factorization is less transparent than the necessary and sufficient condition which characterize nonzeros in the non-factorized inverse.

6. Preconditioning for block triangular matrices. Many sparse matrices arising in real-world applications may be reduced to block triangular form (see Ch. 6 in [20]). In this section we discuss the application of preconditioning techniques to linear systems with a block (lower) triangular coefficient matrix, closely following [30].

The reduction to block triangular form is usually obtained with a two-step procedure, as outlined in [20]. In the first step, the rows of A are permuted to bring nonzero entries on the main diagonal, producing a matrix PA. In the second step, symmetric permutations are used to find the block triangular form [53]. The resulting matrix can be represented as

$$Q(PA)Q^T = \begin{pmatrix} A_{11} & 0 & \cdots & 0 \\ A_{21} & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{pmatrix}$$

where the diagonal blocks A_{ii} are assumed to be irreducible. Because A is nonsingular, the diagonal blocks A_{ii} must also be nonsingular.

Suppose that we compute approximate inverses of the diagonal blocks $A_{11}, \ldots, A_{kk}$ with the incomplete biconjugation algorithm, so that $A_{ii}^{-1} \approx G_{ii} := \bar{Z}_{ii}\bar{D}_{ii}^{-1}\bar{W}_{ii}^T$, $1 \le i \le k$. Then the inverse of A is approximated as follows (cf. [30]):

$$A^{-1} \approx G = Q^T \begin{pmatrix} G_{11}^{-1} & 0 & \cdots & 0 \\ A_{21} & G_{22}^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \cdots & G_{kk}^{-1} \end{pmatrix}^{-1} QP.$$

The preconditioning step in a conjugate gradient-type method requires the evaluation of the action of G on a vector, i.e. the computation of $z = Gd$ for a given vector d, at each step of the preconditioned iterative method. This can be done by a back-substitution of the form

$$\bar{z}_i = G_{ii}\left(\bar{d}_i - \sum_{j=1}^{i-1} A_{ij}\bar{z}_j\right), \qquad i = 1, \ldots, k$$

where

$$\bar{d} = \begin{pmatrix} \bar{d}_1 \\ \vdots \\ \bar{d}_k \end{pmatrix}, \quad \bar{z} = \begin{pmatrix} \bar{z}_1 \\ \vdots \\ \bar{z}_k \end{pmatrix}, \quad z = Q^T\bar{z}, \quad \bar{d} = QPd,$$

with the partitioning of $\bar{z}$ and $\bar{d}$ induced by the block structure of $Q(PA)Q^T$. The computation of $y = G^T c$, which is required by certain preconditioned iterative methods, is accomplished in a similar way.

With this approach, fill-in is confined to the approximate inverses of the diagonal blocks, often resulting in a more sparse preconditioner. Notice also that the approximate inverses G_{ii} can be computed in parallel. The price to pay is the loss of part of the explicitness when the approximate inverse preconditioner is applied, as noted in [30].

For comparison purposes, we apply the same scheme with ILU preconditioning. Specifically, we approximate A as

$$A \approx M = P^T Q^T \begin{pmatrix} \bar{L}_{11}\bar{U}_{11} & 0 & \cdots & 0 \\ A_{21} & \bar{L}_{22}\bar{U}_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \cdots & \bar{L}_{kk}\bar{U}_{kk} \end{pmatrix} Q,$$

where each diagonal block A_{ii} is approximated by an ILU decomposition $\bar{L}_{ii}\bar{U}_{ii}$. Applying the preconditioner requires the solution of a linear system $Mz = d$ at each step of the preconditioned iteration. This can be done with the back-substitution

$$\bar{L}_{ii} y_i = \left(\bar{d}_i - \sum_{j=1}^{i-1} A_{ij} \bar{z}_j \right), \quad \bar{U}_{ii} \bar{z}_i = y_i, \quad i = 1, \ldots, k$$

where

$$\bar{d} = QPd, \qquad z = Q^T \bar{z},$$

with the same partitioning of $\bar{z}$ and $\bar{d}$ as above. The use of transposed ILU preconditioning is similar.

With this type of ILU block preconditioning we introduce some explicitness in the application of the preconditioner. Again, note that the ILU factorizations of the diagonal blocks can be performed in parallel.

We will see in the section on numerical experiments that reduction to the block triangular form influences the behavior of the preconditioned iterations in different ways depending on whether approximate inverse techniques or ILU-type preconditioning are used.

7. Implementation aspects. It is possible to implement the incomplete inverse factorization algorithm in §4 in at least two distinct ways. The first implementation is similar in spirit to the classical submatrix formulation of sparse Gaussian elimination as represented, for instance, in [19],[57]. This approach relies on sparse incomplete rank-one updates of the matrices $\bar{Z}$ and $\bar{W}$, applied in the form of outer vector products. These updates are the most time-consuming part of the computation. In the course of the updates, new fill-in elements whose magnitude is less than a prescribed drop tolerance T are dropped. In this approach, dynamic data structures have to be used for the $\bar{Z}$ and $\bar{W}$ matrices. Note that at step i of the incomplete inverse factorization, only the ith row a_i^T and the ith column c_i^T are required. The matrix A is stored in static data structures both by rows and by columns (of course, a single array is needed for the numerical values of the entries of A).

For this implementation to be efficient, some additional elbow room is necessary. For instance, in the computation of the incomplete $\bar{Z}$ factor the elbow room was twice the space anticipated for storing the nonzeros in the factor itself. As we are looking for a preconditioner with about the same number of nonzeros as the original matrix, the estimated number of nonzeros in $\bar{Z}$ is half the number of nonzeros in the original matrix A. For each column of $\bar{Z}$ we give an initial prediction of fill-in based on the results of §5. Thus, the initial structure of $\bar{Z}$ is given by the structure of the upper triangular part of A. Of course, $\bar{W}$ is handled similarly. If the space initially allocated for a given column is not enough, the situation is solved in a way which is standard when working with dynamic data structures, by looking for a block of free space at the end of the active part of the dynamic data structure large enough to contain the current column, or by a garbage collection (see [57]). Because most of the fill-in in $\bar{Z}$ and $\bar{W}$ appears in the late steps of the biconjugation process, we were able to keep the amount of dynamic data structure manipulations at relatively low levels. In the following, this implementation will be referred to as the DDS implementation.

Despite our efforts to minimize the amount of symbolic manipulations in the DDS implementation, some of its disadvantages such as the nonlocal character of the computations and a high proportion of non-floating-point operations still remain. This is an important drawback of submatrix (right-looking, undelayed) algorithms using dynamic data structures when no useful structural prediction is known and no efficient block strategy is used. Even when all the operations are performed in-core, the work with both the row and column lists in each step of the outer cycle is rather irregular. Therefore, for larger problems, most operations are still 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 [57] for a discussion of the difficulties in parallelizing sparse rank-one updates).

For these reasons we considered an alternative implementation (hereafter referred to as SDS) which only makes use of static data structures, based on a left-looking, delayed update version of the biconjugation algorithm. This amounts to a rearrangement of the computations, as shown below. For simplicity we only consider the Z factor, and assume no breakdown occurs:

(1) Let $z_1^{(0)} = e_1; \quad p_1^{(0)} = a_{11}$

(2) **for** $i = 2, \ldots, n$

$z_i^{(0)} = e_i$

for $j = 1, \ldots, i-1$

$p_i^{(j-1)} := a_j^T z_i^{(j-1)}$

$z_i^{(j)} := z_i^{(j-1)} - \left(\frac{p_i^{(j-1)}}{p_j^{(j-1)}} \right) z_j^{(j-1)}$

end

$p_i^{(i-1)} := a_i^T z_i^{(i-1)}$

end

This procedure can be implemented with only static data structures, at the cost of increasing the number of floating-point operations. Indeed, in our implementation we found it necessary to recompute the dot products $p_i^{(j-1)} = a_j^T z_i^{(j-1)}$ if they are used more than once for updating subsequent columns. 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, because the dot products and the vector updates in the innermost loop can be done in parallel. Notice that with SDS, it is no longer true that a single row and column of A are used at each step of the outer loop. It is worth mentioning that numerically, the DDS and SDS implementations of the incomplete biconjugation process are completely equivalent.

The SDS implementation is straightforward. Suppose the first $j-1$ steps have been completed. In order to determine which columns of the already determined part of $\bar{Z}$ play

a role in the rank-one updates used to form the jth column of $\bar{Z}$ we only need a linked list scanning the structure of the columns of A. This linked list is coded similarly to the mechanism which determines the structure of the jth row of the Cholesky factor L in the numerical factorization in SPARSPAK (see [27],[13]).

In addition to the approximate inverse preconditioner, we also coded the standard row implementation of the classical ILU(0) preconditioner (see, e.g., [50]). We chose a no-fill implicit preconditioner because we are mostly interested in comparing preconditioners with a nonzero density close to that of the original matrix A.

On input, all our codes for the computation of the preconditioners check whether the coefficient matrix has a zero-free diagonal. If not, row reordering of the matrix is used to permute nonzeros on the diagonal. For both the ILU(0) and the approximate inverse factorization, we introduced a simple pivot modification to avoid breakdown. Whenever some diagonal element in any of our algorithms to compute a preconditioner was found to be small, in our case less in absolute value than the IEEE machine precision $\epsilon \approx 2.2 \cdot 10^{-16}$, we increased it to 10^{-3}. We have no special reasons for this choice, other than it worked well in practice. It should be mentioned that in the numerical experiments, this safeguarding measure was required more often for ILU(0) than for the approximate inverse factorization.

For the experiments on matrices which can be nontrivially reduced to block triangular form, we used the routine MC13D from MA28 [19] to get the block triangular form.

8. Numerical experiments. In this section we present the results of numerical experiments on a range of problems from the Harwell-Boeing collection [21] and from Tim Davis' collection [17]. All matrices used were rescaled by dividing their elements by the absolute value of their largest nonzero entry. No other scaling was used. The right-hand side of each linear system was computed from the solution vector x^* of all ones, the choice used, e.g., in [57].

We experimented with several iterative solvers of the conjugate gradient type. Here we present results for three selected methods, which we found to be sufficiently representative: van der Vorst's Bi-CGSTAB method (denoted BST in the tables), the QMR method of Freund and Nachtigal, and Saad and Schultz's GMRES (restarted every 20 steps, denoted G(20) in the tables) with Householder orthogonalization [56]. See [3] for a description of these methods, and the report [9] for experiments with other solvers.

The matrices used in the experiments come from reservoir simulation (ORS*, PORES2, SAYLR* and SHERMAN*), chemical kinetics (FS5414), network flow (HOR131), circuit simulation (JPWH991, MEMPLUS and ADD*), petroleum engineering (WATT* matrices) and incompressible flow computations (RAEFSKY*, SWANG1). The order N and number NNZ of nonzeros for each test problem are given in Table 1, together with the number of iterations and computing times for the unpreconditioned iterative methods. A † means that convergence was not attained in 1000 iterations for Bi-CGSTAB and QMR, and 500 iterations for GMRES(20).

			Its			Time		
MATRIX	N	NNZ	BST	QMR	G(20)	BST	QMR	G(20)
ADD20	2395	17319	378	362	†	7.11	11.7	†
ADD32	4960	23884	66	90	124	2.34	5.52	13.8
FS5414	541	4285	782	865	†	3.91	7.18	†
HOR131	434	4710	†	†	†	†	†	†
JPWH991	991	6027	36	61	94	0.28	0.82	1.98
MEMPLUS	17758	99147	†	886	†	†	270.	†
ORSIRR1	1030	6858	†	807	†	†	12.3	†
ORSIRR2	886	5970	798	571	†	6.01	7.19	†
ORSREG1	2205	14133	480	163	350	9.14	5.25	17.8
PORES2	1224	9613	†	†	†	†	†	†
RAEFSKY1	3242	294276	229	276	†	55.4	106.	†
RAEFSKY5	6316	168658	111	†	69	17.9	†	18.4
SAYLR3	1000	3750	364	479	†	2.46	5.55	†
SAYLR4	3564	22316	†	†	†	†	†	†
SHERMAN1	1000	3750	†	361	†	†	2.43	†
SHERMAN3	5005	20033	†	†	†	†	†	†
SHERMAN4	1104	3786	96	133	†	0.73	1.65	†
SHERMAN5	3312	20793	†	†	†	†	†	†
SWANG1	3169	20841	16	31	33	0.44	1.46	2.34
WATT1	1856	11360	35	125	74	0.56	3.44	3.07
WATT2	1865	11550	†	764	60	†	20.2	2.60

Table 1: Test problems (N= order of matrix, NNZ= nonzeros in matrix) and convergence results for the iterative methods without preconditioning.

All tests were performed on a SGI Crimson workstation with RISC processor R4000 using double precision arithmetic. Codes were written in standard Fortran 77 and compiled with the optimization option *-O4*. CPU time is given in seconds and it was measured using the standard function *dtime*.

The initial guess for the iterative solvers was always $x_0 = 0$. The stopping criterion used was $||r_k||_2 < 10^{-8}$, where r_k is the (unpreconditioned) updated residual. Note that because $r_0 = b = Ax^*$, we have that $1 \leq ||r_0||_\infty \leq nzr$ where nzr denotes the maximum number of nonzeros in a row of A.

The following tables present the results of experiments with the ILU(0) preconditioner and with the approximate inverse preconditioner based on the biconjugation process (hereafter referred to as AIBC). Observe that the number of nonzeros in the ILU(0) preconditioner is equal to the number NNZ of nonzeros in the original matrix, whereas for the AIBC preconditioner fill-in is given by the total number of nonzeros in the factors $\bar{Z}$, $\bar{W}$ and $\bar{D}$. In the tables, the number of nonzeros in AIBC is denoted by $Fill$. Right preconditioning was used for all the experiments.

The comparison between the implicit and the explicit preconditioner is based on the amount of fill and on the rate of convergence as measured by the number of iterations. These two parameters can realistically describe the scalar behavior of the preconditioned iterative methods. Of course, an important advantage of the inverse preconditioner, its explicitness, is not captured by this description.

The accuracy of the AIBC preconditioner is controlled by the value of the drop tolerance T. Smaller drop tolerances result in a more dense preconditioner and very often (but not always) in a higher convergence rate for the preconditioned iteration. For our experiments we consider relatively sparse preconditioners. In most cases we were able to adjust the value of T so as to obtain an inverse preconditioner with a nonzero density close to that of A (and hence of the ILU(0) preconditioner). Due to the scaling of the matrix entries, the choice $T = 0.1$ was very often the right one. We also give results for the approximate inverse obtained with a somewhat smaller value of the drop tolerance, in order to show how the number of iterations can be reduced by allowing more fill-in in the preconditioner. For some problems we could not find a value of T for which the number of nonzeros in AIBC is close to NNZ. In these cases the approximate inverse preconditioner tended to be either very dense or very sparse.

In Table 2 we give the timings for the preconditioner computation, iteration counts and timings for the three iterative solvers preconditioned with ILU(0). The same information is given in Table 3 for the approximate inverse preconditioner AIBC. For AIBC we give two timings for the construction of the preconditioner, the first for the DDS implementation using dynamic data structures and the second for the SDS implementation using only static data structures.

		ILU – Its			ILU – Time		
MATRIX	P-time	BST	QMR	G(20)	BST	QMR	G(20)
ADD20	0.071	128	171	228	4.46	9.67	16.0
ADD32	0.030	26	44	51	1.61	4.61	6.90
FS5414	0.007	6	4	6	0.04	0.10	0.06
HOR131	0.008	40	63	80	0.36	0.88	1.06
JPWH991	0.009	11	20	18	0.17	0.49	0.43
MEMPLUS	0.551	242	257	†	76.1	134.	†
ORSIRR1	0.009	23	39	39	0.39	1.04	1.22
ORSIRR2	0.008	23	38	39	0.34	0.89	1.04
ORSREG1	0.019	24	54	45	0.88	3.10	3.05
PORES2	0.013	25	41	78	0.56	1.46	2.99
RAEFSKY1	2.457	30	38	128	13.9	28.7	67.4
RAEFSKY5	0.293	2	4	4	0.69	2.06	1.59
SAYLR3	0.005	32	45	66	0.37	0.85	1.61
SAYLR4	0.030	30	41	63	1.79	3.90	7.10
SHERMAN1	0.005	32	46	65	0.89	0.37	1.59
SHERMAN3	0.134	66	82	325	5.02	8.75	46.7
SHERMAN4	0.006	24	34	56	6.93	0.69	1.49
SHERMAN5	0.034	27	32	52	1.38	2.65	5.02
SWANG1	0.031	6	13	10	0.33	1.12	0.77
WATT1	0.015	6	16	9	0.25	0.77	0.35
WATT2	0.016	87	56	54	2.58	2.68	4.99

Table 2: Time to form the ILU(0) preconditioner (P-time), number of iterations and time for Bi-CGSTAB, QMR and GMRES(20) with ILU(0) preconditioning.

		P-time		AIBC – Its			AIBC – Time		
MATRIX	Fill	DDS	SDS	BST	QMR	G(20)	BST	QMR	G(20)
ADD20	5900	0.48	1.35	66	64	74	2.09	3.45	4.94
	9752	0.64	1.50	8	15	14	0.28	0.87	0.77
ADD32	8422	1.58	5.31	34	51	64	2.02	5.17	8.68
	15525	1.65	5.29	6	11	11	0.41	1.24	1.39
FS5414	4199	0.30	0.15	37	42	35	0.35	0.64	0.56
	5204	0.34	0.17	10	18	15	0.10	0.29	0.20
HOR131	6078	0.25	0.13	31	54	74	0.31	0.80	1.06
	8394	0.34	0.20	28	45	57	0.32	0.78	0.95
JPWH991	7063	0.31	0.26	15	27	28	0.24	0.67	0.78
	11981	0.37	0.31	12	24	23	0.23	0.71	0.74
MEMPLUS	59547	6.67	65.5	175	85	188	51.9	41.7	110.
	151686	12.9	68.5	22	30	31	8.53	18.7	20.1
ORSIRR1	5219	0.25	0.24	27	46	48	0.42	1.14	1.38
	13117	0.35	0.31	15	26	24	0.31	0.83	0.82
ORSIRR2	5284	0.23	0.20	26	47	47	0.36	1.05	1.21
	12634	0.32	0.25	20	33	24	0.33	0.86	0.85
ORSREG1	11886	0.45	0.98	32	59	51	1.11	3.41	3.39
	24454	0.60	1.07	22	45	37	0.97	3.17	2.89
PORES2	18691	0.55	0.50	84	102	†	2.33	4.47	†
	23867	0.66	0.57	75	103	†	2.36	4.97	†
RAEFSKY1	56607	5.49	12.6	53	75	†	15.7	34.9	†
	145951	16.5	25.1	38	52	†	13.9	29.3	†
RAEFSKY5	33740	4.51	10.1	4	7	7	0.88	2.43	1.94
	104010	4.80	10.3	2	4	4	0.58	1.73	1.43
SAYLR3	3650	0.17	0.20	40	58	83	0.49	1.17	2.10
	11002	0.24	0.24	25	35	44	0.42	0.93	1.30
SAYLR4	42768	0.67	2.57	33	44	74	2.44	5.21	9.63
	48362	0.68	2.63	33	43	64	2.61	5.39	8.63
SHERMAN1	3650	0.17	0.20	40	58	80	0.49	1.16	2.07
	8692	0.24	0.22	28	36	47	0.43	0.88	1.30
SHERMAN3	24439	0.76	5.11	97	120	345	7.01	14.4	52.7
	36296	0.88	5.20	77	100	407	6.36	13.5	65.8
SHERMAN4	3936	0.20	0.25	35	50	133	2.88	1.09	3.76
	4957	0.20	0.16	29	46	124	0.40	1.03	3.60
SHERMAN5	21387	0.77	2.36	37	56	151	1.94	4.83	15.7
	26654	0.89	2.45	30	49	96	1.70	4.54	10.5
SWANG1	7723	0.51	1.94	8	13	13	0.38	0.99	0.89
	13252	0.64	1.99	6	10	10	0.30	0.81	0.67
WATT1	10215	0.37	0.69	7	31	12	0.21	1.46	0.58
	17998	0.46	0.76	7	19	10	0.26	1.05	0.43
WATT2	10148	0.41	0.78	92	71	13	2.63	3.34	0.54
	13547	0.46	0.76	76	62	11	2.34	3.12	0.46

Table 3: Time to form the AIBC preconditioner (P-time) using DDS and SDS implementations, number of iterations and time for Bi-CGSTAB, QMR and GMRES(20) with AIBC preconditioning.

It appears from these results that the ILU(0) and AIBC preconditioners are roughly equivalent from the point of view of the rate of convergence, with ILU(0) having a slight edge. On many problems the two preconditioners give similar results. There are a few cases, like PORES2, for which ILU(0) is much better than AIBC, and others (like MEMPLUS) where the situation is reversed. For some problems it is necessary to allow a relatively high fill in the approximate inverse preconditioner in order to have a convergence rate comparable with that insured by ILU(0) (cf. SAYLR4), but there are cases where a very sparse AIBC gives excellent results (see the ADD or the RAEFSKY matrices). It follows that the timings for the iterative part of the solution process are pretty close, on average, for the two preconditioners.

We also notice that using a more dense approximate inverse preconditioner (obtained with a smaller value of T) nearly always reduces the number of iterations, although this does not necessarily mean a reduced computing time since it takes longer to compute the preconditioner and the cost of each iteration is increased.

Concerning the matrix PORES2, for which our method gives poor results, we observed that fill-in in the $\bar{W}$ factor was very high. We tried to use different drop tolerances for the two factors (the one for $\bar{W}$ being larger than the one used for $\bar{Z}$) but this did not help. It was observed in [31] that finding a sparse right approximate inverse for PORES2 is very hard and a left approximate inverse should be approximated instead. Unfortunately, our method produces exactly the same approximate inverse (up to transposition) for A and A^T, therefore we were not able to cope with this problem effectively. We experienced a similar difficulty with the $\bar{W}$ factor for the matrix SHERMAN2. On the other hand, for SHERMAN3 we did not face any of the problems reported in [30] and convergence with the AIBC preconditioner was smooth.

As for the time required to compute the preconditioners, it is obvious that ILU(0) can be computed more quickly. On the other hand, the computation of the AIBC preconditioner is not prohibitive. There are problems for which computing AIBC is only two to three times more expensive than computing ILU(0). More important, our experiments with AIBC show that the overall solution time is almost always dominated by the iterative part, unless convergence is extremely rapid, in which case the iteration part takes slightly less time than the computation of the preconditioner.

This observation suggests that our approximate inverse preconditioner is much cheaper to construct, in a sequential environment, than approximate inverse preconditioners based

on the Frobenius norm approach described in §3. Indeed, if we look at the results presented in [30] we see that the sequential time required to construct the preconditioner accounts for a huge portion, often in excess of 90%, of the overall computing time. It is worth emphasizing that the approach based on Frobenius norm minimization and the one we propose seem to produce preconditioners of similar quality, in the sense that they are both comparable with ILU(0) from the point of view of fill-in and rates of convergence, at least on average.

As for the different implementations of AIBC, we see from the results in Table 3 that for larger problems, the effect of additional floating-point operations in the SDS implementation is such that the DDS implementation is actually faster. Nevertheless, as already observed the implementation using static data structures may better suited for parallel architectures. Because in this paper we only consider a scalar implementation, in the remaining experiments we limit ourselves to the timings for the DDS implementation of AIBC.

In all the experiments (excluding the ones performed to measure the timings presented in the tables) we monitored also the "true" residual $||b - Ax_k||_2$. In general, we found that the discrepancy between this and the norm of the updated residual was small. However, we found that for some very ill-conditioned matrices in the Harwell-Boeing collection (not included in the tables) this difference may be very large. For instance, for some of the LNS* and WEST* matrices, we found that $||r_k||_2 < 10^{-8}||b - Ax_k||_2$ for the final value of r_k. This happened both with the ILU(0) and with the approximate inverse preconditioner, and we regarded this as a failure of the preconditioned iterative method.

We present in Tables 4 and 5 the results of some experiments on matrices which have been reduced to block lower triangular form. We compared the number of iterations of the preconditioned iterative methods and their timings for the block approximate inverse preconditioner and for the block ILU(0) preconditioner as described in §6. Since some of the matrices have only trivial block lower triangular form (one block, or two blocks with one of the blocks of dimension one for some matrices) we excluded them from our experiments.

In Table 4 we give for each matrix the number NBL of blocks and the results of experiments with ILU(0). In Table 5 we give analogous results for the AIBC preconditioner. The amount of fill-in (denoted by $Fill$) for AIBC is computed as the fill-in in the approximate inverses of the diagonal blocks plus the number of nonzero entries in the off-diagonal blocks.

MATRIX	NBL	P-time	Block ILU – Its			Block ILU – Time		
			BST	QMR	G(20)	BST	QMR	G(20)
JPWH991	146	0.012	11	20	19	0.15	0.36	0.51
SAYLR3	318	0.008	40	65	73	0.45	0.95	1.61
SHERMAN1	318	0.008	40	65	73	0.46	1.00	1.62
SHERMAN3	2111	0.178	101	105	371	9.00	9.39	49.0
SHERMAN4	559	0.009	22	33	59	0.27	0.52	1.46
SHERMAN5	1675	0.048	22	37	60	1.20	2.59	5.57
WATT1	129	0.024	10	26	9	0.31	0.99	0.39
WATT2	65	0.023	5	32	7	0.16	1.20	0.29

Table 4: Time to compute the block ILU preconditioner (P-time), number of iterations and time for Bi-CGSTAB, QMR and GMRES(20) with block ILU(0) preconditioning.

MATRIX	Fill	P-time	Block AIBC – Its			Block AIBC – Time		
			BST	QMR	G(20)	BST	QMR	G(20)
JPWH991	7063	0.46	15	†	25	0.31	†	0.82
	11981	0.57	11	†	19	0.21	†	0.58
SAYLR3	3384	0.36	42	57	75	0.54	0.92	1.78
	9892	0.42	20	33	38	0.33	0.65	1.04
SHERMAN1	5562	0.39	33	49	68	0.47	0.87	1.76
	7842	0.46	23	35	42	0.37	0.67	1.11
SHERMAN3	21821	1.65	93	122	†	7.38	11.8	†
	31540	1.85	80	100	487	6.94	10.5	71.8
SHERMAN4	4356	0.41	31	46	106	0.47	0.86	2.87
	5770	0.42	27	41	93	0.43	0.79	2.58
SHERMAN5	29155	1.88	25	41	81	1.70	3.27	8.52
	36764	2.16	24	38	63	1.77	3.22	6.88
WATT1	10158	0.73	7	18	11	0.22	0.68	0.49
	17494	0.86	6	16	9	0.21	0.91	0.43
WATT2	9194	0.74	6	15	9	0.19	0.55	0.38
	13006	0.76	5	11	8	0.16	0.44	0.34

Table 5: Time to compute the block AIBC preconditioner (P-time) , number of iterations and time in seconds for Bi-CGSTAB, QMR and GMRES(20) with block AIBC preconditioning.

It is clear that in general the reduction to block triangular form does not lead to a noticeable improvement in the timings, at least in a sequential implementation. We observe that when the block form is used, the results for ILU(0) are sometimes worse. This can

probably be attributed to the permutations, which are known to cause in some cases a degradation of the rate of convergence of the preconditioned iterative method [22]. A notable exception is the matrix WATT2, for which the number of iterations is greatly reduced. On the other hand, the results for the block approximate inverse preconditioner are mostly unchanged or somewhat better. Again, matrix WATT2 represents an exception: this problem greatly benefits from the reduction to block triangular form. In any case, permutations did not adversely affect the rate of convergence of the preconditioned iterative method. This fact suggests that perhaps the approximate inverse preconditioner is more robust than ILU(0) with respect to reorderings.

To gain more insight on how permutations of the original matrix can influence the quality of both types of preconditioners, we did some experiments where the matrix A was permuted using the minimum degree algorithm on the structure of $A + A^T$ (see [28]). We applied the resulting permutation to A symmetrically to get PAP^T, in order to preserve the nonzero diagonal. Tables 6 and 7 present the results for the test matrices having trivial block triangular form. The corresponding preconditioners are denoted by ILU(0)-MD and AIBC-MD, respectively.

		ILU-MD – Its			ILU-MD – Time		
MATRIX	P-time	BST	QMR	G(20)	BST	QMR	G(20)
ADD20	0.043	21	30	30	0.73	1.71	1.97
ADD32	0.030	26	43	47	2.18	4.57	6.45
HOR131	0.008	35	59	94	0.31	0.80	1.24
MEMPLUS	0.182	196	254	†	61.3	131	†
ORSIRR1	0.011	128	180	197	2.12	4.65	5.92
ORSIRR2	0.009	138	175	215	1.96	3.92	5.64
ORSREG1	0.024	153	198	237	5.35	10.9	15.9
PORES2	0.015	177	220	†	4.00	8.14	†
RAEFSKY1	2.875	55	70	†	25.1	51.3	†
RAEFSKY5	0.337	2	4	4	0.68	2.04	1.56
SAYLR4	0.042	882	816	†	51.5	76.8	†
SWANG1	0.037	5	10	9	0.29	0.89	0.65

Table 6: Time to compute the ILU(0) preconditioner (P-time) for A permuted according to minimum degree algorithm on $A + A^T$, number of iterations and time for Bi-CGSTAB, QMR and GMRES(20) with ILU(0)-MD preconditioning.

			AIBC-MD – Its			AIBC-MD – Time		
MATRIX	Fill	P-time	BST	QMR	G(20)	BST	QMR	G(20)
ADD20	5173	0.38	27	35	40	0.85	1.93	2.73
	8360	0.49	8	14	13	0.28	0.78	0.77
ADD32	10168	1.61	29	36	49	1.80	3.76	6.55
	11994	1.77	6	12	12	0.38	1.30	1.45
HOR131	5632	0.24	38	50	77	0.37	0.73	1.11
	7152	0.31	35	51	78	0.39	0.88	1.27
MEMPLUS	48375	6.35	16	28	27	4.71	13.5	14.9
	76843	9.00	15	29	25	4.82	15.3	14.8
ORSIRR1	4058	0.30	24	43	48	0.38	1.23	1.38
	10330	0.38	15	27	27	0.29	0.82	0.87
ORSIRR2	5185	0.26	33	41	42	0.47	0.92	1.10
	10340	0.36	14	25	25	0.25	0.68	0.73
ORSREG1	10643	0.56	33	63	54	1.13	3.57	3.60
	15585	0.61	31	51	49	1.17	3.13	3.43
PORES2	19192	0.56	98	97	†	2.75	4.27	†
	19409	0.58	104	95	†	2.91	4.14	†
RAEFSKY1	80544	22.2	46	62	†	14.6	30.8	†
	113160	19.6	41	54	†	14.1	28.8	†
RAEFSKY5	33435	3.74	3	7	6	0.67	2.43	1.67
	98670	5.85	2	4	4	0.58	1.71	1.37
SAYLR4	22766	1.01	51	52	320	3.17	5.26	38.2
	24196	1.02	41	57	412	2.60	5.85	49.3
SWANG1	7737	0.60	8	13	13	0.38	0.99	0.92
	13421	0.94	6	10	10	0.31	0.83	0.67

Table 7: Time to compute the AIBC preconditioner (P-time) for A permuted by the minimum degree algorithm on $A + A^T$, number of iterations and time for Bi-CGSTAB, QMR and GMRES(20) with AIBC-MD preconditioning.

The results in Table 6 show that for some problems, especially those coming from PDEs, minimum degree reordering has a detrimental effect on the convergence of the iterative solvers preconditioned with ILU(0). In some cases we see a dramatic increase in the number of iterations. This is in analogy with the observed fact (see, e.g., [22]) that when the minimum degree ordering is used, the no-fill incomplete Cholesky decomposition of an SPD

matrix is a poor approximation of the coefficient matrix, at least for problems arising from the discretization of 2D PDEs. The convergence of the conjugate gradient method with such a preconditioner (ICCG(0)) is much slower than if the natural ordering of the unknowns was used. Here we observe a similar phenomenon for nonsymmetric linear systems. Note the rather striking behavior of matrix ADD20, which benefits greatly from the minimum degree reordering (this matrix arises from a circuit model and not from the discretization of a PDE).

It was also observed in [22] that the negative impact of minimum degree on the rate of convergence of PCG all but disappears when the incomplete Cholesky factorization of A is computed by means of a drop tolerance rather than by position. It is natural to ask whether the same holds true for the approximate inverse preconditioner AIBC, which is computed using a drop tolerance. The results in Table 7 show that this is indeed the case. For most of the test problems the number of iterations was nearly unaffected (or better) and in addition we note that the minimum degree ordering helps in preserving sparsity in the incomplete inverse factors. While this is usually not enough to decrease the computing times, the fact that it is possible to reduce storage demands for the approximate inverse preconditioner without negatively affecting the convergence rates might become important for very large problems.

We conclude this section with some observations concerning the choice of the drop tolerance T. In all our experiments we used a fixed value of T throughout the incomplete biconjugation process. However, relative drop tolerances, whose value is adapted from step to step, could also be considered (see [57] for a thorough discussion of the issues related to the choice of drop tolerances in the context of ILU). We have observed that the amount of fill-in is distributed rather unevenly in the course of the approximate inverse factorization. A large proportion of nonzeros is usually concentrated in the last several columns of $\bar{Z}$ and $\bar{W}$. For some problems with large fill, it may be preferable to switch to a larger drop tolerance when the columns of the incomplete factors start filling-in strongly. Conversely, suppose we have computed an approximate inverse preconditioner for a certain value of T, and we find that the preconditioned iteration is converging slowly. Provided that enough storage is available, one could then try to recompute at least some of the columns of $\bar{Z}$ and $\bar{W}$ using a smaller value of T. Unfortunately, for general sparse matrices there is no guarantee that this will result in a preconditioner of improved quality. Indeed, allowing more nonzeros in the preconditioner does not always result in a reduced number of iterations.

Finally, it is worthwhile to observe that a dual threshold variant of the incomplete inverse factorization could be adopted, see [51]. In this approach, a drop tolerance is applied but a maximum number of nonzeros per column is specified and enforced during the computation of the preconditioner. In this way, it is possible to control the maximum storage needed by the preconditioner, which is important for an automated implementation. This approach has not been tried yet, but we hope to do so in the near future.

9. Conclusions and future work. In this paper we have developed a sparse approximate inverse preconditioning technique for nonsymmetric linear systems. Our approach is based on a procedure to compute two sets of biconjugate vectors, performed incompletely to preserve sparsity. This algorithm produces an approximate triangular factorization of A^{-1}, which is guaranteed to exist if A is an H-matrix (similar to the ILU factorization).

The factorized sparse approximate inverse is used as an explicit preconditioner for conjugate gradient-type methods. Applying the preconditioner only requires sparse matrix-vector products, which is of considerable interest for use on parallel computers.

The new preconditioner was used to enhance the convergence of different iterative solvers. Based on extensive numerical experiments, we found that our preconditioner can insure convergence rates which are comparable, on average, with those from the standard ILU(0) implicit preconditioner. While the approximate inverse factorization is more time-consuming to compute than ILU(0), its cost is not prohibitive, and is typically dominated by the time required by the iterative part. This is in contrast with other approximate inverse preconditioners, based on Frobenius norm minimization, which produce similar convergence rates but are very expensive to compute.

It is possible that in a parallel environment the situation will be reversed, since the preconditioner construction with the Frobenius norm approach is inherently parallel. However, there is some scope for parallelization also in the inverse factorization on which our method is based: for instance, the approximate inverse factors $\bar{Z}$ and $\bar{W}$ can be computed largely independent of each other. Clearly, this is a point which requires further research, and no conclusion can be drawn until parallel versions of this and other approximate inverse preconditioners have been implemented and tested.

Our results point to the fact that the quality of the approximate inverse preconditioner is not greatly affected by reorderings of the coefficient matrix. This is important in practice because it suggests that we may use permutations to increase the potential for parallelism or to reduce the amount of fill in the preconditioner, without spoiling the rate of convergence.

The theoretical results on fill-in in §5 provide guidelines for the use of pivoting strategies for enhancing the sparsity of the approximate inverse factors, and this is a topic that deserves further research.

Based on the results of our experiments, we conclude that the technique introduced in this paper has the potential to become a useful tool for the solution of large sparse nonsymmetric linear systems on modern high-performance architectures. Work on a parallel implementation of the new preconditioner is currently under way. Future work will also include a dual threshold implementation of the preconditioner computation.

Acknowledgments. We would like to thank one of the referees for helpful comments and suggestions, and Professor Miroslav Fiedler for providing reference [24]. The first author gratefully acknowledges the hospitality and excellent research environment provided by the Institute of Computer Science of the Czech Academy of Sciences.

REFERENCES

[1] E. C. Anderson. *Parallel Implementation of Preconditioned Conjugate Gradient Methods for Solving Sparse Systems of Linear Equations.* M.Sc. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA, 1988. CSRD Report No. 805.

[2] O. Axelsson. *Iterative Solution Methods.* Cambridge University Press, Cambridge, 1994.

[3] R. Barret, 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.* SIAM, Philadelphia, 1994.

[4] M. Benson, J. Krettmann and M. Wright. Parallel algorithms for the solution of certain large sparse linear systems. *Intern. J. Computer Math.*, 16:245–260, 1984.

[5] M. Benzi. *A Direct Row-Projection Method for Sparse Linear Systems.* Ph.D. Thesis, Department of Mathematics, North Carolina State University, Raleigh, NC, USA, 1993.

[6] M. Benzi and C. D. Meyer. A direct projection method for sparse linear systems. *SIAM J. Sci. Comput.*, 16:1159–1176, 1995.

[7] M. Benzi and C. D. Meyer. An explicit preconditioner for the conjugate gradient method. In J. D. Brown, M. T. Chu, D. C. Ellison and R. J. Plemmons, editors, *Proceedings of the Cornelius Lanczos International Centenary Conference*, pages 294–296. SIAM, Philadelphia, 1994.

[8] M. Benzi, C. D. Meyer and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17, 1996 (in press).

[9] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. Research Report No. 653, Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic, 1995.

[10] H. Berryman, J. Saltz, W. Gropp and R. Mirchandaney. Krylov methods preconditioned with incompletely factored matrices on the CM-2. *J. Parall. Distr. Comput.*, 8:186–190, 1990.

[11] E. Chow and Y. Saad. Approximate inverse preconditioners for general sparse matrices. Research Report UMSI 94/101, University of Minnesota Supercomputer Institute, Minneapolis, MN, USA, 1994.

[12] E. Chow and Y. Saad. Approximate inverse techniques for block-partitioned matrices. Research Report UMSI 95/13, University of Minnesota Supercomputer Institute, Minneapolis, MN, USA, 1995.

[13] E. Chu, A. George, J. W.-H. Liu and E. G.-Y. Ng. User's guide for SPARSPAK-A: Waterloo sparse linear equations package. Technical Report CS-84-36, University of Waterloo, Waterloo, Ontario, Canada, 1984.

[14] M. T. Chu, R. E. Funderlic and G. H. Golub. A rank-one reduction formula and its applications to matrix factorizations. *SIAM Review*, 37:512–530, 1995.

[15] P. Concus, G. H. Golub and G. A. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Stat. Comput.*, 6:220–252, 1985.

[16] J. D. F. Cosgrove, J. C. Diaz and A. Griewank. Approximate inverse preconditionings for sparse linear systems. *Intern. J. Computer Math.*, 44:91–110, 1992.

[17] T. Davis. Sparse matrix collection. *NA Digest*, Volume 94, Issue 42, October 1994.

[18] S. Demko, W. F. Moss and P. W. Smith. Decay rates for inverses of band matrices. *Math. Comp.*, 43:491–499, 1984.

[19] I. S. Duff. MA28 - a set of Fortran subroutines for sparse unsymmetric linear equations. Harwell Report AERE R8730, HMSO, London, England. Revised 1980.

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

[21] I. S. Duff, R. G. Grimes and J. G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection. Technical Report RAL-92-086, Rutherford Appleton Laboratory, Chilton, England, 1992.

[22] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29:1–14, 1989.

[23] H. Elman. A stability analysis of incomplete LU factorizations. *Math. Comp.*, 47:191–217, 1986.

[24] M. Fiedler. Inversion of bigraphs and connection with the Gauss elimination. In *Graphs, Hypergraphs and Block Systems*, pages 57–68, Zielona Gora, 1976.

[25] L. Fox. *An Introduction to Numerical Linear Algebra.* Oxford University Press, Oxford, 1964.

[26] L. Fox, H. D. Huskey and J. H. Wilkinson. Notes on the solution of algebraic linear simultaneous equations. *Quart. J. Mech. Appl. Math.*, 1:149–173, 1948.

[27] A. George and J. W.-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[28] A. George and J. W.-H. Liu. The evolution of the minimum degree algorithm. *SIAM Review*, 31:1–19, 1989.

[29] J. R. Gilbert. Predicting structure in sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 15:62–79, 1994.

[30] N. I. M. Gould and J. A. Scott. On approximate-inverse preconditioners. Technical Report RAL-95-026, Rutherford Appleton Laboratory, Chilton, England, 1995.

[31] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, to appear.

[32] M. Grote and H. Simon. Parallel preconditioning and approximate inverses on the Connection Machine. In R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold and D. A. Reed, editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 519–523. SIAM, Philadelphia, 1993.

[33] 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. Num. Math.*, 8:93–115, 1991.

[34] M. R. Hestenes. Inversion of matrices by biorthogonalization and related results. *J. Soc. Indust. Appl. Math.*, 6:51–90, 1958.

[35] M. R. Hestenes and E. Stiefel. Method of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.

[36] A. S. Householder. *The Theory of Matrices in Numerical Analysis*. Blaisdell Publishing Co., New York, 1964. Reprinted by Dover, New York, 1975.

[37] O. G. Johnson, C. A. Micchelli and G. Paul. Polynomial preconditioning for conjugate gradient calculations. *SIAM J. Numer. Anal.*, 20:362–375, 1983.

[38] M. T. Jones and P. E. Plassmann. The efficient parallel iterative solution of large sparse linear systems. In A. George, J. R. Gilbert and J. W.-H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, IMA Vol. 56, pages 229–245. Springer-Verlag, New York, 1994.

[39] I. E. Kaporin. Explicitly preconditioned conjugate gradient method for the solution of unsymmetric linear systems. *Intern. J. Computer Math.*, 40:169–187, 1992.

[40] I. E. Kaporin. Two-level explicit preconditioning of the conjugate gradient method. *Differential Equations*, 28:280–289, 1992.

[41] I. E. Kaporin. New convergence results and preconditioning strategies for the conjugate gradient method. *Num. Lin. Alg. Appl.*, 1:179–210, 1994.

[42] L. Yu. Kolotilina, A. A. Nikishin and A. Yu. Yeremin. Factorized sparse approximate inverse (FSAI) preconditionings for solving 3D FE systems on massively parallel computers II: Iterative construction of FSAI preconditioners. In R. Beauwens and P. de Groen, editors, *Proceedings of the IMACS International Symposium on Iterative Methods in Linear Algebra*, pages 311–312. North-Holland, Amsterdam, 1992.

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

[44] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditioning II: Solution of 3D FE systems on massively parallel computers. *Intern. J. High Speed Comput.*, 7:191–215, 1995.

[45] S. L. Lee. *Krylov Methods for the Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA, 1993. Technical Report UIUCDS-R-93-1814.

[46] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34:473–497, 1980.

[47] 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.*, 31:148–162, 1977.

[48] N. S. Mendelsohn. Some properties of approximate inverses of matrices. *Trans. Roy. Soc. of Canada, Section III*, 50:53–59, 1956.

[49] Y. Saad. Preconditioning techniques for nonsymmetric and indefinite linear systems. *J. Comput. Appl. Math.*, 24:89–105, 1988.

[50] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report CSRD TR 1029, CSRD, University of Illinois at Urbana-Champaign, IL, USA, 1990.

[51] Y. Saad. ILUT: A dual threshold incomplete LU factorization. *Num. Lin. Alg. Appl.*, 1:387–402, 1994.

[52] G. W. Stewart. Conjugate direction methods for solving systems of linear equations. *Numer. Math.*, 21:283–297, 1973.

[53] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1:146–160, 1972.

[54] H. A. van der Vorst. High performance preconditioning. *SIAM J. Sci. Stat. Comput.*, 10:1174–1185, 1989.

[55] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1962.

[56] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput.*, 9:152–163, 1988.

[57] Z. Zlatev. *Computational Methods for General Sparse Matrices*. Kluwer Academic Publishers, Dordrecht, 1991.