SIAM J. SCI. COMPUT.
Vol. 17, No. 5, pp. 1135–1149, September 1996

A SPARSE APPROXIMATE INVERSE PRECONDITIONER FOR THE CONJUGATE GRADIENT METHOD*

MICHELE BENZI†, CARL D. MEYER‡, AND MIROSLAV TŮMA§

Abstract. A method for computing a sparse incomplete factorization of the inverse of a symmetric positive definite matrix A is developed, and the resulting factorized sparse approximate inverse is used as an explicit preconditioner for conjugate gradient calculations. It is proved that in exact arithmetic the preconditioner is well defined if A is an H-matrix. The results of numerical experiments are presented.

Key words. sparse approximate inverses, preconditioned conjugate gradient method, H-matrices, incomplete factorizations

AMS subject classifications. 65F10, 65F35, 65F50, 65Y05

1. Introduction.

In this paper we develop a method for computing an incomplete factorization of the inverse of a symmetric positive definite (SPD) matrix A. The resulting factorized sparse approximate inverse is used as an explicit preconditioner for the solution of $Ax = b$ by the preconditioned conjugate gradient (PCG) method. Due to the fact that an explicit preconditioning step only requires matrix–vector products, explicit preconditioners are of considerable interest for use on parallel computers [7, 9, 17, 19–21]. This is in contrast with more traditional preconditioners based on incomplete factorizations of the coefficient matrix A which necessitate triangular solves (a serial bottleneck) in the preconditioning steps. Sparse incomplete inverses are also useful in the construction of sparse approximate Schur complements for use in incomplete block factorization preconditioners [2, 8]. Furthermore, our preconditioner does not require that A be explicitly stored, a feature which is useful for some problems where A is only implicitly given as an operator.

The paper is organized as follows. In §2 we describe the main idea upon which the preconditioner is based. Section 3 is devoted to a proof of the existence of the incomplete inverse factorization for H-matrices, while in §§4 and 5 implementation details and the results of numerical experiments are discussed. Our experiments indicate that this preconditioning strategy can insure rapid convergence of the PCG iteration with convergence rates comparable with those of the best serial preconditioners. In §6 we draw some conclusions and we indicate some future research directions.

This paper can be viewed as a natural outgrowth of work on a direct sparse linear solver based on oblique projections [3, 4, 28].

---

*Received by the editors July 20, 1994; accepted for publication (in revised form) April 21, 1995.

†Dipartimento di Matematica, Università degli Studi di Bologna, 40127 Bologna, Italy and CERFACS, 42 Ave. G. Coriolis, 31057 Toulouse Cedex, France (benzi@dm.unibo.it and benzi@cerfacs.fr).

‡Mathematics Department, North Carolina State University, Raleigh, NC 27695-8205 (meyer@math.ncsu.edu). The work of this author was supported in part by National Science Foundation grants DMS-9403224 and CCR-9413309 and by computing grants from the North Carolina Super Computing Center.

§Academy of Sciences of the Czech Republic, Institute of Computer Science, 182 07 Prague 8–Libeň, Czech Republic (tuma@uivt.cas.cz). The work of this author was supported in part by grants GA CR 201/93/0067 and GA AS CR 230401.

2. Computing an incomplete inverse factorization. If $A_{n\times n}$ is an SPD matrix, then a factorization of A^{-1} can readily be obtained from a set of conjugate directions $z_1, z_2, \ldots, z_n$ for A. If

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

is the matrix whose ith column is z_i, we have

$$Z^T A Z = D = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n \end{pmatrix} \quad \text{where} \quad p_i = z_i^T A z_i.$$

It follows that

$$A^{-1} = Z D^{-1} Z^T,$$

and a factorization of A^{-1} is obtained. A set of conjugate directions z_i may be constructed by means of a "conjugate Gram–Schmidt" (or A-orthogonalization) process applied to any set of linearly independent vectors $v_1, v_2, \ldots, v_n$. The choice $v_i = e_i$ (the ith unit vector) is computationally convenient. The resulting Z matrix is unit upper triangular; indeed,

$$Z = L^{-T} \quad \text{where} \quad A = LDL^T$$

is the root-free Cholesky factorization of A. Denoting the ith row of A by a_i^T, the inverse factorization algorithm can be written as follows.

THE INVERSE FACTORIZATION ALGORITHM

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

(2) **for** $i = 1, 2, \ldots, n$
  for $j = i, i+1, \ldots, n$
    $p_j^{(i-1)} := a_i^T z_j^{(i-1)}$
  end
  if $i = n$ **go to** (3)
  for $j = i+1, \ldots, n$
    $z_j^{(i)} := z_j^{(i-1)} - \left(\dfrac{p_j^{(i-1)}}{p_i^{(i-1)}} \right) z_i^{(i-1)}$
  end
end

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

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

Notice that the matrix A need not be explicitly stored—only the capability of forming inner products involving the rows of A is required. This is an attractive

feature for cases where the matrix is only implicitly given as an operator. Once Z and D are available, the solution of $Ax = b$ can be computed as

$$x^* = A^{-1}b = ZD^{-1}Z^Tb = \sum_{i=1}^{n} \left(\frac{z_i^T b}{p_i} \right) z_i.$$

A similar algorithm was first proposed in [14]; see also [13, 18]. Further references and a few historical notes can be found in [3, 4]. For a dense matrix this method requires roughly twice as much work as Cholesky. For a sparse matrix the cost can be substantially reduced, but the method is still impractical because the resulting Z tends to be dense. The idea of computing a sparse approximation of Z to construct a preconditioner for the conjugate gradient method was first proposed in [3] (see also [4, 5]). This paper is devoted to developing and testing this idea.

Sparsity is preserved by reducing the amount of fill-in occurring in the computation of the z-vectors (that is, above the main diagonal in the unit upper triangular matrix Z). This can be achieved either by ignoring all fill outside selected positions in Z or by discarding fill whose magnitude falls below a preset drop tolerance (see §4 for details). The motivation for this approach is based upon theoretical results and computer experiments which show that many of the entries in the inverse (or in the inverse Cholesky factor) of a sparse SPD matrix are small in absolute value [2, 10, 25]. Several authors have exploited this fact to construct explicit preconditioners based on sparse approximate inverses [2, 20, 21]. However, the approach taken in this paper is quite different from the previous ones.

If the incomplete inverse factorization process is successfully completed, one obtains a unit upper triangular matrix $\bar{Z}$ and a diagonal matrix $\bar{D}$ with positive diagonal entries such that

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

is a factorized sparse approximate inverse of A. It is shown in the next section that such an incomplete inverse factorization of A exists (in exact arithmetic) for arbitrary values of the drop tolerance and for any choice of the sparsity pattern in $\bar{Z}$ when A is an H-matrix. For general SPD matrices the process may break down due to the occurrence of negative or zero pivots $\bar{p}_i$. Although numerical experiments show that this breakdown is not very likely to occur for reasonably well-conditioned problems, it is necessary to safeguard the computation of the approximate pivots against breakdown in order to obtain a robust procedure (see §4).

In this paper we limit ourselves to SPD matrices, but it is possible to apply the inverse factorization algorithm to arbitrary matrices. In exact arithmetic, the procedure can be carried out provided that all leading principal minors of A are nonzero [3]. The resulting Z and D matrices satisfy

$$AZ = LD$$

where L, a unit lower triangular matrix, is not explicitly computed. Hence, Z is the inverse of U in the LDU factorization of A. The application of such an implicit Gaussian elimination method to the solution of sparse linear systems has been investigated in [3, 4, 28].

3. Existence of the incomplete inverse factorization. The preconditioner based on the incomplete inverse factorization of A exhibits many analogies with the classical incomplete LDU factorization of Meijerink and van der Vorst [24]. These

authors proved that such an incomplete factorization is well defined for arbitrary zero structures of the incomplete factors if A is an M-matrix. In other words, if A is a nonsingular M-matrix, then the incomplete factorization can be carried out (in exact arithmetic) and the computed pivots are strictly positive. Furthermore, the pivots in the incomplete factorization are no smaller than the pivots in the exact factorization. In [23], Manteuffel extended the existence of incomplete LDU factorizations to the class of H-matrices. Recall that $A = [a_{ij}]$ is an H-matrix if $\hat{A} = [\hat{a}_{ij}]$ is an M-matrix where

$$\hat{a}_{ij} = \begin{cases} -|a_{ij}| & \text{when } i \neq j, \\ a_{ii} & \text{when } i = j. \end{cases}$$

Note that a diagonally dominant matrix is an H-matrix.

This result means that if A is a symmetric H-matrix, then the incomplete Cholesky factorization always exists and it can be used to construct an SPD preconditioner for the conjugate gradient method. If A is a general (non-H) SPD matrix, the incomplete factorization may break down due to the occurrence of zero pivots, or the corresponding preconditioner may fail to be positive definite due to the presence of negative pivots.

The same turns out to be true for the incomplete inverse factorization described in the previous section. Here we prove that in exact arithmetic the inverse factorization algorithm given in §2 will never break down provided that A is an H-matrix. In the symmetric case this implies that the approximate inverse $\bar{Z}\bar{D}^{-1}\bar{Z}^T$ is positive definite, so it may be used as preconditioner for the conjugate gradient method. This fact was first proved for M-matrices in [3].

The proof runs as follows. First we show that the incomplete process will never break down if A is an M-matrix. This is a consequence of the fact that dropping a nonzero fill-in in the computation of a vector z_j at step i is equivalent to setting the corresponding entry of the ith row of A to zero. Because the off-diagonal entries of an M-matrix are nonpositive and the entries of $\bar{Z}$ are nonnegative, this shows that the pivots $\bar{p}_i$ produced by the inexact scheme are greater than or equal to the exact pivots p_i. Since these are strictly positive for an M-matrix, no breakdown can occur during the inexact inverse factorization scheme.

Subsequently we show that when A is an H-matrix, the pivots p_i computed by the inverse factorization scheme are no smaller than the pivots $\hat{p}_i$ corresponding to the associated M-matrix.

When combined, these two results will insure the stability of the incomplete procedure for H-matrices. For symmetric matrices this will mean that the factorized approximate inverse is positive definite and can be used as a preconditioner for the conjugate gradient method. However, symmetry is not required in our proof.

PROPOSITION 3.1. *Let A be an M-matrix and let p_i be the pivots produced by the inverse factorization algorithm. If $\bar{p}_i$ are the pivots computed by the incomplete inverse factorization algorithm with any preset zero pattern in the strictly upper triangular part of Z or any value of the drop tolerance, then*

$$\bar{p}_i \geq p_i > 0.$$

Proof. From the identity $AZ = LD$ and the fact that Z and L are unit triangular matrices it follows that the pivots p_i can be expressed in terms of the leading principal minors Δ_i of A as

$$p_i = \frac{\Delta_i}{\Delta_{i-1}} \quad (1 \leq i \leq n; \ \Delta_0 = 1).$$

Because A is an M-matrix, all its leading principal minors are positive and therefore $p_i > 0$ for all i. After $i-1$ steps of the inverse factorization scheme, the column vectors $z_j^{(i-1)}$ $(i \le j \le n)$ are available. Let $z_{kj}^{(i-1)}$ denote the kth entry of $z_j^{(i-1)}$. At step i of the inverse factorization scheme, the following are computed:

$$p_j^{(i-1)} = \sum_{l=1}^{i-1} a_{il} z_{lj}^{(i-1)} + a_{ij} \quad (i \le j \le n). \tag{3.1}$$

Suppose now that a sparsity pattern is imposed on the z-vectors, or that all fill-in in the z-vectors whose magnitude falls below a given drop tolerance is to be dropped. The modified z-vectors will be denoted by $\bar{z}_j^{(i-1)}$, and the pivots are now given by

$$\bar{p}_i^{(i-1)} = \sum_{l=1}^{i-1} a_{il} \bar{z}_{li}^{(i-1)} + a_{ii}. \tag{3.2}$$

We show by induction that $\bar{p}_i^{(i-1)} > 0$ for $1 \le i \le n$. We also show that $\bar{p}_j^{(i-1)} \le 0$ for $i+1 \le j \le n$ and that $\bar{z}_j^{(i-1)} \ge 0$ (componentwise) for $i \le j \le n$ for all i. For $i = 1$ the inequalities are obviously true. Now fix $i \ge 2$ and assume that $\bar{p}_{i-1}^{(i-2)} > 0$, $\bar{p}_j^{(i-2)} \le 0$ for $i \le j \le n$, and $\bar{z}_j^{(i-2)} \ge 0$ for $i-1 \le j \le n$. It follows that in the updates

$$\bar{z}_j^{(i-1)} := \bar{z}_j^{(i-2)} - \left(\frac{\bar{p}_j^{(i-2)}}{\bar{p}_{i-1}^{(i-2)}} \right) \bar{z}_{i-1}^{(i-2)}$$

nonpositive quantities are subtracted from nonnegative quantities. Therefore, even after dropping, no component of $\bar{z}_j^{(i-1)}$ can become negative. That is, $\bar{z}_j^{(i-1)} \ge 0$ for $i \le j \le n$. Using this inequalities and the fact that the off-diagonal entries of an M-matrix are nonpositive, we see from (3.1) that $\bar{p}_j^{(i-1)} \le 0$ for $i+1 \le j \le n$. Finally, it is clear from (3.2) that $\bar{p}_i^{(i-1)} \ge p_i^{(i-1)}$. Thus, the pivots cannot become smaller because of dropping. Since the exact pivots are positive, this proves that the incomplete inverse factorization process will not break down. □

We explicitly observe that when A is an M-matrix, our method is guaranteed to produce a nonnegative approximate inverse.

Now let A be an H-matrix, and apply the inverse factorization scheme to A as well as to the associated M-matrix $\hat{A}$. In the sequel, quantities with hats correspond to the associated process on $\hat{A}$. We need to compare pivots and z-vectors for the original process (on A) and for the associated process (on $\hat{A}$). To do this we also need to introduce intermediate quantities—denoted with tildes—which are constructed with entries from $\hat{A}$ and with pivots from A.

PROPOSITION 3.2. *Let A be an H-matrix and let $\hat{A}$ be the associated M-matrix. If p_i and $\hat{p}_i$ denote the pivots computed by the inverse factorization scheme applied to A and to $\hat{A}$, respectively, then $p_i \ge \hat{p}_i$. Furthermore, if $\bar{p}_i$ denote the pivots computed by the incomplete inverse factorization algorithm applied to A, then $\bar{p}_i \ge \hat{p}_i$.*

Proof. Consider the elements $z_{lj}^{(k)}$ of $z_j^{(k)}$ as rational functions

$$z_{lj}^{(k)} = F_{lj}^{(k)}(a_{11}, \ldots, a_{kn}, p_1, \ldots, p_k)$$

dependent on the elements of A and on the pivots $p_1, \ldots, p_k$. Likewise, we can consider the entries $\hat{z}_{lj}^{(k)}$ as rational functions depending on the entries in the first k rows of $\hat{A}$ and on the corresponding pivots $\hat{p}_1, \ldots, \hat{p}_k$. Let

$$\tilde{z}_{lj}^{(k)} = F_{lj}^{(k)}(\hat{a}_{11}, \ldots, \hat{a}_{kn}, p_1, \ldots, p_k)$$

be computed in the same way as $\hat{z}_{lj}^{(k)}$ using $p_1, \ldots, p_k$ instead of $\hat{p}_1, \ldots, \hat{p}_k$. In the following, it helps to think of the pivots p_i, $\hat{p}_i$ as parameters, ignoring their dependency on the entries of A and $\hat{A}$, respectively. In this way, the entries of $z_j^{(k)}$, $\hat{z}_j^{(k)}$, and $\tilde{z}_j^{(k)}$ can be regarded as polynomials in the entries of A and $\hat{A}$, respectively. We will prove that $p_i \geq \hat{p}_i$ using induction on i. We make the following inductive assumptions for all $k \leq i-1$.

$$p_k \geq \hat{p}_k, \tag{3.3}$$

$$\hat{z}_{lj}^{(k)} \geq \tilde{z}_{lj}^{(k)} \text{ for } l \leq j, j \geq i, \tag{3.4}$$

$$\tilde{z}_{lj}^{(k)} \text{ has all its terms nonnegative (as a polynomial).} \tag{3.5}$$

I. For $i = 1$ we have $p_1 = a_{11} = \hat{a}_{11} = \hat{p}_1 > 0$ and $\hat{z}_{lk}^{(1)} \geq \tilde{z}_{lk}^{(1)} \geq 0$.
II. Using (3.1) for $\hat{p}_i$ we get

$$\hat{p}_i = \sum_{l=1}^{i-1} \hat{a}_{il}\hat{z}_{li}^{(i-1)} + \hat{a}_{ii} \leq \sum_{l=1}^{i-1} \hat{a}_{il}\tilde{z}_{li}^{(i-1)} + \hat{a}_{ii}.$$

This inequality follows from the inductive assumption $\hat{z}_{li}^{(i-1)} \geq \tilde{z}_{li}^{(i-1)}$ and from the fact that the $\hat{a}_{il}$'s are nonpositive (being off-diagonal elements of the associated M-matrix).

Notice that corresponding terms in the expressions of $\tilde{z}_{li}^{(i-1)}$ and $z_{li}^{(i-1)}$ as polynomials have the same absolute value, so they can differ only by the sign. Using the defining identities for the H-matrix, i.e., $\hat{a}_{ii} = a_{ii}$ and $\hat{a}_{ik} = -|a_{ik}|$ for $i \neq k$, we get

$$\sum_{l=1}^{i-1} \hat{a}_{il}\tilde{z}_{li}^{(i-1)} + \hat{a}_{ii} \leq \sum_{l=1}^{i-1} a_{il}z_{li}^{(i-1)} + a_{ii} = p_i.$$

All terms in $\hat{a}_{il}\tilde{z}_{li}^{(i-1)}$ on the left-hand side, when considered as a polynomial in elements of $\hat{A}$, are nonpositive. On the right-hand side, some of the terms in the expression for $z_{li}^{(i-1)}$ can be positive. But corresponding terms of these polynomials have the same absolute value, so they differ only by the sign. Hence the inequality.

Using the updating formula—given in the inverse factorization algorithm—for $\hat{z}_j^{(i)}$ we have

$$\hat{z}_j^{(i)} = \hat{z}_j^{(i-1)} - \frac{\sum_{l=1}^{i-1} \hat{a}_{il}\hat{z}_{lj}^{(i-1)} + \hat{a}_{ij}}{\hat{p}_i}\hat{z}_i^{(i-1)}.$$

Since the off-diagonal elements of the M-matrix are nonpositive, and since $p_i \geq \hat{p}_i > 0$, we obtain

$$\hat{z}_j^{(i)} \geq \tilde{z}_j^{(i-1)} - \frac{\sum_{l=1}^{i-1} \hat{a}_{il}\tilde{z}_{lj}^{(i-1)} + \hat{a}_{ij}}{p_i}\tilde{z}_i^{(i-1)} \equiv \tilde{z}_j^{(i)}.$$

This follows from the set of inequalities

$$-\hat{a}_{il}\hat{z}_{lj}^{(i-1)} \geq -\hat{a}_{il}\tilde{z}_{lj}^{(i-1)},$$

$$\hat{z}_{i}^{(i-1)} \geq \tilde{z}_{i}^{(i-1)},$$

$$\hat{p}_{i}^{-1} \geq p_{i}^{-1}.$$

Assembling everything together we have

$$-\frac{\sum_{l=1}^{i-1}\hat{a}_{il}\hat{z}_{lj}^{(i-1)}+\hat{a}_{ij}}{\hat{p}_i}\hat{z}_{i}^{(i-1)} \geq -\frac{\sum_{l=1}^{i-1}\hat{a}_{il}\tilde{z}_{lj}^{(i-1)}+\hat{a}_{ij}}{p_i}\tilde{z}_{i}^{(i-1)}.$$

This inequality is added to the inequality from the assumption

$$\hat{z}_{j}^{(i-1)} \geq \tilde{z}_{j}^{(i-1)}$$

to arrive at

$$\hat{z}_{j}^{(i)} \geq \tilde{z}_{j}^{(i)}.$$

In addition, all the terms of $\tilde{z}_{j}^{(i)}$ are nonnegative.

Using the inductive assumption and defining identities for the H-matrix it can also be seen that (3.5) is true for $k = i$.

Concerning the second statement, it is easily seen that the same inequality for the pivots holds when the inverse factorization algorithm is applied to A incompletely, as the same argument for the polynomial terms can be applied even when some elements of $\hat{z}_{j}^{(i)}$ are set to zero. □

It follows from Propositions 3.1 and 3.2 that the incomplete inverse factorization process will never break down (in exact arithmetic) when A is an H-matrix. This is true for arbitrary zero patterns in the strictly upper triangular part of $\bar{Z}$ and for arbitrary choices of the drop tolerance.

The pivots produced by the incomplete inverse factorization of an H-matrix are no smaller than the pivots produced by the incomplete inverse factorization of the associated M-matrix. However, they are not necessarily larger than the pivots produced by the exact inverse factorization of A, contrary to what happens in the M-matrix case. For example, consider the H-matrix

$$\begin{pmatrix} 4 & -1 & -\epsilon \\ -1 & 4 & 1 \\ -\epsilon & 1 & 4 \end{pmatrix}.$$

If $0 < \epsilon < 1/4$, the incomplete inverse factorization algorithm with drop tolerance $Tol = 1/16$ returns a pivot $\bar{p}_3$ which is smaller than the pivot p_3 produced by the exact inverse factorization scheme. This is in perfect analogy with incomplete Cholesky factorizations (see [23, p. 479]).

If A is not an H-matrix, the incomplete inverse factorization algorithm may break down. For instance, applying the algorithm with a drop tolerance $Tol = 0.06$ to the SPD matrix

$$\begin{pmatrix} 2.00 & 0.40 & 0.10 \\ 0.40 & 1.08 & 2.00 \\ 0.10 & 2.00 & 3.96 \end{pmatrix}$$

results in $\tilde{p}_3 = 0$ (a breakdown).

In finite precision computations, zero or negative pivots may occur even for H-matrices, due to round-off errors. Also, trouble can be expected in the presence of extremely small pivots. Indeed, this is one way for severe ill conditioning to manifest itself. Furthermore, there are many applications leading to SPD matrices which are not H-matrices—typically, finite element analysis. It is therefore desirable to incorporate some safeguard mechanism in the incomplete algorithm which guarantees that the computation of the preconditioner will run to completion and that it will always produce a symmetric positive definite approximate inverse factorization. Similar techniques have been implemented in connection with incomplete Cholesky factorization preconditioning and with approximate Hessian modifications [23, 16, 27].

4. Notes on implementation. We have implemented the PCG algorithm with our approximate inverse preconditioner—hereafter referred to as AINV—based on the inverse factorization algorithm of §2 as well as with a standard incomplete Cholesky (IC) preconditioner. The purpose of this comparison is to explore some characteristic algorithmic properties of the explicit preconditioner and to get a feeling for the convergence rate for the explicit PCG method as compared with one of the best implicit preconditioners.

We first describe the IC preconditioner used in our comparison. It was computed by a standard column algorithm with symbolic and numeric phases (see [15, 22]). During the decomposition we removed all the elements of the factor less than a prescribed drop tolerance Tol. Necessary working space was thus dominated by the size (number of nonzero entries) of the lower triangular factor of the IC preconditioner.

This decomposition, which could break down for general (non-H) matrices, was modified by a standard stabilization; see [16]. The algorithm insures that all diagonal elements of D in the LDL^T decomposition are strictly positive and the absolute values of the elements of L satisfy a uniform upper bound in order to preserve numerical stability and to prevent excessively large elements in the factors. We refer to [16] for details.

Computing the AINV preconditioner is a slightly more complicated process. In the Cholesky case we can use an elimination tree structure to minimize symbolic integer overhead and working storage. Due to the complicated rules governing fill-in in the AINV case, it is not clear how to realize an analogous symbolic process. Therefore, we used a submatrix type of algorithm which updates at each step all the remaining z-vectors by a rank-one modification. We adopted dynamic data structures similar to those used in submatrix formulations of sparse unsymmetric Gaussian elimination (see [11,28,29]). This requires the user to provide an estimate for the number of nonzeros allowed in the preconditioner. Additionally, some elbow space is needed for the data structures. In our implementation the elbow space was four times the estimated space for storing the nonzeros in the preconditioner—similar to the implementation of sparse Gaussian elimination in the widely used packages MA28 [11] and Y12M [29].

However, there are some differences in the use of such data structures in Gaussian elimination and in the AINV procedure. For instance, these data structures are used in AINV for the matrix Z but not for A, now stored in static data structures. During the AINV process, A is delivered into the cache by rows since we need at each step only one row of A. Recall that in some cases it may even be possible to avoid storage of A altogether—e.g., when a routine is available to compute the action of A on a vector (we did not take advantage of this option in our implementation).

The amount of fill-in created during the computation of the AINV preconditioner

in most of the first steps is very small and thus the integer overhead and CPU time spent in these initial stages is very small. This is in contrast with sparse Gaussian elimination (as represented, for instance, by MA28), where the proportion of integer overhead and CPU time is distributed more uniformly over the algorithmic steps.

The sizes of the data structures in the AINV case which are necessary in the top level of the memory hierarchy (cache and registers) were found to be small, often much smaller than the size of the preconditioner. This fact can strongly influence performance, especially on workstation equipment. Nevertheless, working storage for the implementation of AINV is larger than for the implementation of IC.

Sparsity was preserved on the basis of value rather than on the positions of fill-in. For capturing the relevant entries in the inverse Cholesky factor of A, this is a better strategy than imposing a preset sparsity pattern on Z. Consistency suggested that drop tolerances be used with IC as well.

Skipping some z-vector updates in step (2) of the inverse factorization algorithm when the coefficients $p_j^{(i-1)}/p_i^{(i-1)}$ were in some sense "small" produced bad numerical results, so no skipping was done.

In the AINV case we also implemented an algorithmic modification to avoid breakdown for general SPD (non-H) matrices. When some computed diagonal element $\bar{p}_i$ was too small—in our case, less than $\sqrt{\epsilon_M}$ where ϵ_M is the machine precision—we replaced it by

$$\bar{p}_i \longleftarrow \max\{\sqrt{\epsilon_M},\ \mu\sigma\theta\} \tag{4.1}$$

where

$$\mu = 0.1 \text{ (a relaxation parameter)},$$
$$\sigma = \max_{i \le k \le n-1} \left\{ p_k^{(i-1)} \right\},$$
$$\theta = \|z_i^{(i-1)}\|_\infty \neq 0.$$

The rule (4.1) was chosen to avoid breakdowns due to very small or negative diagonal elements $\bar{p}_i$. It also has the effect of constraining the growth of elements in the z-vectors. This avoids break downs, but, just as in the IC case, there is no guarantee that we will get a good preconditioner after this regularization.

5. Numerical experiments. The following experiments illustrate some properties of the two preconditioners when applied within the PCG algorithm to SPD matrices. Nine test matrices were taken from the Harwell–Boeing collection [12] and the remaining two were kindly provided by G. Zilli (Padua University).

All experiments were run on a SGI Crimson computer with RISC processor R4000. Codes were written in Fortran 77 and compiled with the optimization level –O4. CPU time was measured using the standard function *dtime*. We also experimented with the compiling option $-MIPS2$ which enables double word loads and stores. It is known that this can substantially enhance the performance of floating point arithmetic of some codes. This option led to slightly improved timings in only a few cases, mostly for ICCG. The timings reported in the tables are the best between those obtained with the two compiling options.

All matrices were rescaled by dividing their elements by their largest nonzero entry, but no preordering of their elements was used. The right-hand side of each system was computed using the solution vector composed of ones. The PCG iteration was terminated when the 2-norm of the unpreconditioned residual had been reduced

to less than 10^{-9}. The matrices used in the experiments correspond to finite element approximations to problems in structural engineering (NOS3, NOS5, PADUA1, PADUA2), finite difference approximations to elliptic PDEs (NOS6, NOS7, GR3030) and to modelling of power system networks (BUS matrices).

TABLE 1
Behavior of the unpreconditioned CG algorithm.

test matrix	n	density	time	iterations
NOS3	960	8,402	1.76	266
NOS5	468	2,820	0.88	468*
NOS6	675	1,965	1.66	675*
NOS7	729	2,673	1.89	729*
494BUS	494	1,080	0.91	494*
662BUS	662	1,568	1.30	614
685BUS	685	1,967	1.44	556
1138BUS	1,138	2,596	3.83	1,138*
GR3030	900	4,322	0.28	45
PADUA1	812	3,135	3.06	812*
PADUA2	1,802	13,135	24.11	1,802*

The listings in Table 1 are for the unpreconditioned CG algorithm. Column 1 is the name of the test matrix; column 2 lists the size (n) of the matrix; column 3 (density) gives the number of nonzeros in the lower triangular part including the diagonal of the test matrix; column 4 (time) reports the execution time in seconds; and column 5 (iterations) gives the number of iterations. An asterisk (*) in column 5 indicates that the algorithm failed to converge after n steps using the above mentioned stopping criterion, and computations were terminated.

From the description of our implementation it can be expected that the CPU times for computing the two preconditioners IC and AINV will be different because the IC computation has very small integer overhead. Of course, this difference may become negligible if a sequence of linear systems with the same coefficient matrix (or a slightly modified one) and different right-hand sides has to be solved, since the time for computing the preconditioners is then only a small fraction of the time required for the overall computation.

Drop tolerances parameterize IC and AINV in different ways. That is, using the same Tol value will produce very different results in the two cases. It is preferable to compare the two preconditioners in terms of amount of fill-in rather than to compare two preconditioners obtained using the same value of Tol. The key role in the experiments is played by the fill-in allowed in the preconditioners. Allowing more fill-in results in less PCG iterations, though not always in less overall CPU time. The relation between preconditioner size (measured by fill-in) and number of PCG iterations for AINV and IC is one of the objectives of our comparison.

Table 2 lists the results of applying PCG with different sizes (measured by fill-in) of IC and AINV on the 675×675 Harwell–Boeing test matrix NOS6 which is derived from Poisson's equation in an L-shaped region with mixed boundary conditions. The timings in this table do not include the time required to compute the preconditioner.

TABLE 2
Behavior of PCG using IC versus AINV on H–B test matrix NOS6.

IC			AINV		
fill-in	**iterations**	**time**	**fill-in**	**iterations**	**time**
675	87	0.33	743	76	0.32
897	53	0.18	780	74	0.32
912	51	0.18	1,135	54	0.26
1,204	38	0.14	1,208	47	0.18
1,439	32	0.14	1,300	40	0.21
1,520	28	0.10	1,502	37	0.16
1,565	24	0.10	3,654	22	0.14
1,918	8	0.03	17,387	6	0.09

The results in Table 2 indicate that by using preconditioners of restricted size (obtained by adjusting the drop tolerances), the iteration counts as well as the timings for IC and AINV preconditioning are comparable, even in scalar mode allowing slightly more fill-in for the AINV preconditioner. For preconditioners of comparable size, slightly more iterations are needed by AINV preconditioning.

If we keep the size of the preconditioners "moderate," we usually decrease overall CPU time. What moderate means here is strongly problem and architecture (CPU, memory hierarchy) dependent.

The AINV method tends to generate more fill-in than IC, and for small drop tolerances the fill-in for AINV can be so high on some structured problems that we can no longer talk of sparse approximate inverse preconditioning thus making the comparison with IC not very meaningful (a preconditioner can be considered sparse if it contains about the same number of nonzeros as the original matrix or less). However, as discussed in [3, 4], problems having irregular sparsity patterns seem to be well suited for the AINV preconditioner because fill-in is often reasonably low.

TABLE 3
Iteration counts and timings for the IC preconditioner in PCG.

Test Matrix	Fill-in	PCG steps	IC time	PCG time	Fill-in	PCG steps	IC time	PCG time
NOS3	2,290	150	0.08	1.45	10,875	32	0.08	0.38
NOS5	744	76	0.04	0.25	1,967	57	0.05	0.21
NOS6	912	51	0.04	0.19	1,439	32	0.05	0.14
NOS7	902	51	0.09	0.21	938	40	0.09	0.14
494BUS	532	197	0.02	0.51	807	114	0.01	0.31
662BUS	694	159	0.02	0.58	1,090	103	0.02	0.42
685BUS	719	184	0.03	0.69	1,554	77	0.04	0.31
1138BUS	1,183	316	0.07	1.82	2,084	121	0.07	0.63
GR3030	900	45	0.07	0.28	4,322	26	0.06	0.22
PADUA1	1,228	104	0.06	0.54	1,644	70	0.06	0.45
PADUA2	5,305	143	0.23	2.71	7,777	84	0.20	1.65

Table 3 shows iteration counts and timings for PCG using the IC preconditioner, and Table 4 gives the same information for PCG using the AINV preconditioner. These tables also report the time required to compute each preconditioner. Again, the timings for PCG refer to the iterative part only. The IC and AINV preconditioners were computed with similar restricted sizes up to about the original number of nonzeros. Drop tolerances for IC were taken between 0.0001 and 0.01, and drop tolerances for AINV were in the range 0.1 to 0.6. For each test matrix two sparse preconditioners were computed—the first being very sparse while the second contains roughly the same number of nonzeros as the test matrix being used.

TABLE 4
Iteration counts and timings for the AINV preconditioner in PCG.

Test Matrix	Fill-in	PCG steps	AINV time	PCG time	Fill-in	PCG steps	AINV time	PCG time
NOS3	1,946	139	0.27	1.53	6,213	89	0.27	1.14
NOS5	889	87	0.09	0.29	2,086	67	0.12	0.30
NOS6	743	76	0.11	0.32	1,502	37	0.10	0.17
NOS7	903	55	0.07	0.28	2,727	30	0.13	0.19
494BUS	683	173	0.07	0.48	899	110	0.10	0.35
662BUS	893	147	0.13	0.68	1,008	125	0.10	0.56
685BUS	808	178	0.09	0.74	1,836	90	0.11	0.46
1138BUS	1,808	205	0.15	1.22	2,013	156	0.21	0.86
GR3030	900	45	0.12	0.29	13,541	26	0.35	0.37
PADUA1	1,274	63	0.14	0.38	1,459	48	0.14	0.31
PADUA2	5,269	159	0.25	3.14	11,095	80	0.42	1.92

Our results indicate that implicit and explicit sparse preconditioners can have similar behavior—even in the scalar case. The fact that a somewhat higher fill-in is required by the AINV preconditioner in order to achieve the same reduction in the number of PCG iterations as with IC is only natural, since in AINV we are approximating the inverse Cholesky factor (usually a dense matrix), whereas IC is a sparse approximation to the Cholesky factor L itself. If $\bar{L}$ is an incomplete Cholesky factor of A and $\bar{Z}$ is an incomplete inverse Cholesky factor, and if the amount of nonzeros in these two matrices is about the same, then one can expect that $\bar{L}^{-1}$ will be substantially denser than $\bar{Z}$.

In other words, for the same amount of fill-in in the preconditioners, IC yields a better approximation to A^{-1} than AINV. But this comes at a price—namely that two triangular solves are needed at each PCG iteration. On the other hand, the price to pay for the explicitness afforded by the AINV preconditioner is the increased size of the preconditioner, so, on a scalar computer, IC has a slight edge over AINV. However, the situation could be reversed in a parallel computing environment thanks to the explicit nature of AINV. This is a point which warrants further research, and no firm conclusion can be drawn until a parallel version of AINV-PCG has been actually implemented and compared with recent work on parallel solution of sparse triangular systems (see, for instance, [1]). In any event, we observe that even in scalar mode there are problems for which AINV is superior to IC—e.g., PADUA1.

It should be observed that all test matrices used are M-matrices except for NOS3, NOS5, PADUA1, and PADUA2, and these are not even H-matrices. In no case was safeguarding necessary during the computation of the AINV preconditioners, whereas in a few cases IC shifted positive pivots away from zero by a very small amount. This did not adversely affect the convergence of PCG.

6. Conclusions and future work. Our study involved a novel approach to approximate inverse preconditioning for conjugate gradient calculations. One interesting feature of this technique is the fact that the entries of the coefficient matrix A are not explicitly needed, which may be useful for problems where A is only implicitly given as an operator. It was proven that the computation of the preconditioner has the same robustness as standard IC factorization, and numerical evidence was given to make the point that the new preconditioner is competitive with IC—even in scalar mode. The results presented in this paper suggest that our approximate inverse preconditioner can be a useful tool for the solution of large sparse SPD linear systems on modern high-performance architectures.

Future research will focus on efficient parallel implementations and on the extension to unsymmetric problems. A sparse approximate inverse preconditioner for a nonsymmetric matrix A may be obtained by constructing a set of approximate A-biconjugate directions. This can be achieved by applying the inverse factorization algorithm to both A and A^T together with suitable sparsity-preserving strategies. The resulting factorized sparse approximate inverse, which is guaranteed to exist when A is an H-matrix, is an explicit preconditioner which can be used to enhance the convergence of conjugate gradient-like methods for the solution of $Ax = b$.

A different approach, applicable to general sparse matrices, is based on the normal equations $A^TAx = A^Tb$. The solution of this system by the preconditioned conjugate gradient method (PCGNR) is an effective strategy for problems which are unsymmetric and strongly indefinite; see [26, 29]. Also, the PCGNR method is attractive for solving large sparse linear least squares problems. Some of the most effective preconditioners for PCGNR are based on incomplete orthogonal factorizations of A and do not require explicitly forming the matrix A^TA. These procedures compute a sparse approximation to the upper triangular factor R in the QR decomposition of A. It is known that this approach is more robust than computing an incomplete Cholesky factorization of A^TA (notice that R is the transpose of the Cholesky factor of A^TA).

A natural idea is to compute an approximate inverse preconditioner for A^TA based on the inverse factorization scheme of §2. At step i of the algorithm the ith row of A^TA is computed and used and then discarded. The resulting $\bar{Z}$ is a sparse approximation to R^{-1}. The PCGNR scheme can be carried out free of triangular solves in the preconditioning steps. This approach was found to be effective for problems in which A^TA enjoys some form of diagonal dominance but in general was not competitive with more traditional schemes based on variants of the Gram–Schmidt orthogonalization process. More important, there exist other methods which can be used to compute R^{-1} directly from A. A description of some incomplete orthogonalization methods for approximating R and R^{-1}, together with the results of numerical experiments on a variety of general sparse matrices (including rectangular ones), can be found in [6].

REFERENCES

[1] F. L. Alvarado, A. Pothen, and R. Schreiber, *Highly parallel sparse triangular solution,* in

Graph Theory and Sparse Matrix Computations, IMA Vol. 56, A. George, J. R. Gilbert, and J. W. H. Liu, eds., Springer, New York, 1994, pp. 141–157.

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

[3] M. BENZI, *A Direct Row-Projection Method For Sparse Linear Systems,* Ph.D. thesis, Department of Mathematics, North Carolina State University, Raleigh, NC, 1993.

[4] M. BENZI AND C. D. MEYER, *A direct projection method for sparse linear systems,* SIAM J. Sci. Comput., 16 (1995), pp. 1159–1176.

[5] ———*An explicit preconditioner for the conjugate gradient method,* Proceedings of the Cornelius Lanczos International Centenary Conference, J. D. Brown et al., eds., Society of Industrial and Applied Mathematics, Philadelphia, 1994, pp. 294–296.

[6] M. BENZI AND M. TŮMA, *A comparison of some preconditioning techniques for general sparse matrices,* in Proc. of the Second IMACS International Symposium on Iterative Methods in Linear Algebra, S. Margenov and P. Vassilevski, eds., 1995, to appear.

[7] E. CHOW AND Y. SAAD, *Approximate inverse preconditioners for general sparse matrices,* in Proc. of the Colorado Conference on Iterative Methods, April 5–9, 1994.

[8] P. CONCUS, G. H. GOLUB, AND G. MEURANT, *Block preconditioning for the conjugate gradient method,* SIAM J. Sci. Statis. Comput., 6 (1985), pp. 220–252.

[9] J. D. F. COSGROVE, J. C. DIAZ, AND A. GRIEWANK, *Approximate inverse preconditioning for sparse linear systems,* Internat. J. Computer Math., 44 (1992), pp. 91–110.

[10] S. DEMKO, W. F. MOSS, AND P. W. SMITH, *Decay rates for inverses of band matrices,* Math. Comp., 43 (1984), pp. 491–499.

[11] I. S. DUFF, MA28 - *A Set of Fortran Subroutines for Sparse Unsymmetric Linear Equations,* Harwell Report AERE - R.8730, Harwell Labortories, 1980.

[12] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems,* ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[13] D. K. FADDEEV AND V. N. FADDEEVA, *Computational Methods of Linear Algebra,* W. H. Freeman and Co., San Francisco, London, 1963.

[14] 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), pp. 149–173.

[15] A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems,* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[16] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization,* Academic Press, London, 1981.

[17] M. GROTE AND H. SIMON, *Parallel preconditioning and approximate inverses on the Connection Machine,* in Proc. of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, R. Sincovec et al., eds., Society of Industrial and Applied Mathematics, Philadelphia, 1993, pp. 519–523.

[18] A. S. HOUSEHOLDER, *The Theory of Matrices in Numerical Analysis,* Blaisdell Publishing Co., New York, 1964.

[19] T. HUCKLE AND M. GROTE, *A new approach to parallel preconditioning with sparse approximate inverses,* Manuscript SCCM-94-03, Scientific Computing and Computational Mathematics Program, Stanford University, Stanford, CA, May 1994.

[20] L. YU. KOLOTILINA AND A. YU. YEREMIN, *Factorized sparse approximate inverse preconditioning I. Theory,* SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.

[21] E. A. LIPITAKIS AND D. J. EVANS, *Explicit semi-direct methods based on approximate inverse matrix techniques for solving boundary-value problems on parallel processors,* Math. Comput. Simulation, 29 (1987), pp. 1–17.

[22] J. W. H. LIU, *The role of elimination trees in sparse factorization,* SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.

[23] T. A. MANTEUFFEL, *An incomplete factorization technique for positive definite linear systems,* Math. Comp., 34 (1980), pp. 473–497.

[24] 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), pp. 148–162.

[25] G. MEURANT, *A review of the inverse of symmetric tridiagonal and block tridiagonal matrices,* SIAM J. Matrix Anal. Appl., 13 (1992), pp. 707–728.

[26] Y. SAAD, *Preconditioning techniques for nonsymmetric and indefinite linear systems,* J. Comput. Appl. Math., 24 (1988), pp. 89–105.

[27] R. B. SCHNABEL AND E. ESKOW, *A new modified Cholesky factorization,* SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1136–1158.

[28] M. TŮMA, *Solving Sparse Unsymmetric Sets of Linear Equations Based on Implicit Gauss Projection,* Technical rep. 556, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, April 1993.

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