SIAM J. SCI. COMPUT. © 1998 Society for Industrial and Applied Mathematics
Vol. 19, No. 2, pp. 605–625, March 1998 017

SPARSE APPROXIMATE-INVERSE PRECONDITIONERS USING NORM-MINIMIZATION TECHNIQUES*

NICHOLAS I. M. GOULD† AND JENNIFER A. SCOTT†

Abstract. We investigate the use of sparse approximate-inverse preconditioners for the iterative solution of unsymmetric linear systems of equations. We consider the approximations proposed by Cosgrove, Diaz, and Griewank [*Internat. J. Comput. Math.*, 44 (1992), pp. 91–110] and Huckle and Grote [*A New Approach to Parallel Preconditioning with Sparse Approximate Inverses*, Tech. report SCCM-94-03, Stanford University, 1994] which are based on norm-minimization techniques. Such methods are of particular interest because of the considerable scope for parallelization. We propose a number of enhancements which may improve their performance. When run in a sequential environment, these methods can perform unfavorably when compared with other techniques. However, they can be successful when other methods fail and simulations indicate that they can be competitive when considered in a parallel environment.

Key words. iterative methods, unsymmetric systems, preconditioners, sparse approximate inverses

AMS subject classifications. 65F05, 65F10, 65F15, 65F50, 65K05, 90C30

PII. S1064827595288425

1. Introduction.

Suppose that $\boldsymbol{A}$ is a real $n \times n$ unsymmetric matrix whose columns are $\boldsymbol{a}_j$, $1 \le j \le n$. We are principally concerned with the solution of large, sparse systems of linear equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.1}$$

using iterative techniques (see, for example, [11]). Such methods invariably require one or more matrix–vector products per iteration; convergence is normally accelerated using a preconditioner $\boldsymbol{P}$, in which case the required matrix–vector products involve $\boldsymbol{AP}$ or $\boldsymbol{PA}$, not $\boldsymbol{A}$.

The solution of such problems remains a considerable challenge, especially if we are interested in robust methods for general problems. The difficulties are twofold. First, the theory of iterative methods for (1.1) is, at best, incomplete. In particular, it is difficult to be confident, when faced with a new problem, that a given algorithm will converge in a reasonable time, if at all. Second, while it is recognized that preconditioning the system often improves the convergence of a particular method, this is not always so. In particular, a successful preconditioner for one class of problems may prove ineffective on another class. Thus, it has long been recognized that the construction of successful general purpose preconditioners is unlikely to be possible.

In this paper, we are interested in constructing approximations $\boldsymbol{M}$ to the inverse of $\boldsymbol{A}$ for which $\|\boldsymbol{AM} - \boldsymbol{I}\|$ is small—here and elsewhere $\boldsymbol{I}$ is the $n \times n$ identity matrix, $\boldsymbol{e}_i$ is its ith column, and $\boldsymbol{e} = \sum_{i=1}^{n} \boldsymbol{e}_i$. So long as matrix–vector products involving $\boldsymbol{M}$ are inexpensive, the matrix $\boldsymbol{M}$ may then be a suitable preconditioner. Notice that even if $\boldsymbol{A}$ is singular, it may still be possible to choose a suitable preconditioner for a consistent system (1.1) by approximately minimizing some norm of $\boldsymbol{AM} - \boldsymbol{I}$. Recently there has been a lot of interest in such preconditioners (see [5], [6], [9], [16],

*Received by the editors June 28, 1995; accepted for publication (in revised form) May 25, 1996. http://www.siam.org/journals/sisc/19-2/28842.html

†Department for Computation and Information, Atlas Centre, Rutherford Appleton Laboratory, Oxfordshire OX11 0QX, UK (n.gould@rl.ac.uk., sct@letterbox.rl.ac.uk).

[18], and [19], and the references contained therein), and there is some evidence that they can be effective in practice. However, we have been somewhat surprised that there has been no proper assessment of their effectiveness in comparison with other preconditioners, and little on the correctness of the assumptions on which they are based.

A common assumption made in the construction of approximate inverses is that the approximation should be sparse. This is later convenient from the point of forming matrix–vector products and of storage, but such an assumption is at variance with the form of the true inverse which is almost always dense (see, for instance, [12, section 12.6]). Nevertheless, for certain classes of problems (see, for instance, [10]), the elements of the inverse decay rather rapidly to zero away from the diagonal and thus may reasonably be approximated by zeros.

The paper is organized as follows. We describe the basis of sparse approximate-inverse preconditioners in section 2. As these methods heavily depend on our ability to solve sequences of least-squares problems, we consider the generic overdetermined least-squares problem in section 3. In section 4, we examine the efficiencies which are then possible when the data is sparse. We then present numerical results and compare sparse approximate-inverse preconditioners with incomplete LU factorization (ILU) preconditioners in section 5, mention other alternatives in section 6, and draw broad conclusions in section 7.

2. Sparse approximate-inverse preconditioners.

The methods proposed in [6] and [18] impose specified sparsity patterns on the approximate inverse. To be specific, let $\mathcal{N} \stackrel{\text{def}}{=} \{1, \ldots, n\}$, let $\mathcal{S}$ be a given set of index pairs (i, j), with $i, j \in \mathcal{N}$, and let $\mathcal{G}_{\mathcal{S}}$ to be the space of all $n \times n$ matrices which have entries in the positions indexed by $\mathcal{S}$. Similarly, define $\mathcal{S}_j \stackrel{\text{def}}{=} \{i : (i,j) \in \mathcal{S}\}$ and let $\mathcal{G}^j_{\mathcal{S}}$ be the space of all n-vectors which have entries in the positions indexed by $\mathcal{S}_j$. The entries in the approximate inverse $\boldsymbol{M}$ are calculated by solving the problem

$$\underset{\boldsymbol{M} \in \mathcal{G}_{\mathcal{S}}}{\text{minimize}} \;\; \|\boldsymbol{AM} - \boldsymbol{I}\|_F^2 \equiv \sum_{j=1}^{n} \underset{\boldsymbol{m}_j \in \mathcal{G}^j_{\mathcal{S}}}{\text{minimize}} \;\; \|\boldsymbol{A m}_j - \boldsymbol{e}_j\|_2^2, \tag{2.1}$$

where $\boldsymbol{m}_j$ is the jth column of $\boldsymbol{M}$ and where $\|\cdot\|_F$ and $\|\cdot\|_2$ are, respectively, the Frobenius- and two-norms (see also [2]). Thus, each of the columns of $\boldsymbol{M}$ may be calculated independently, and, if required, in parallel, by solving the least-squares problem

$$\underset{\boldsymbol{m}_j \in \mathcal{G}^j_{\mathcal{S}}}{\text{minimize}} \;\; \|\boldsymbol{A m}_j - \boldsymbol{e}_j\|_2^2. \tag{2.2}$$

The differences between the proposals in [6] and [18] are primarily concerned with the specification of $\mathcal{S}$. The principal ingredients are as follows:

(a) the least-squares problems are cheap to solve so long as $\mathcal{S}$ is sparse,

(b) the residuals may be improved by enlarging $\mathcal{S}$,

(c) there are effective means of choosing how to enlarge $\mathcal{S}$, and

(d) the process may be enhanced by updating rather than recomputing as $\mathcal{S}$ enlarges.

Our principal concern, in this paper, is improving components (c) and (d).

3. The generic least-squares problem.

Since we require that the jth column $\boldsymbol{m}_j$ of the sparse approximate inverse $\boldsymbol{M}$ belongs to $\mathcal{G}^j_{\mathcal{S}}$, the only columns of $\boldsymbol{A}$

involved in the least-squares problem (2.2) are those flagged by $\mathcal{G}^j_{\mathcal{S}}$. Thus, in this section, we consider the generic overdetermined least-squares problem

$$\sigma = \underset{\boldsymbol{z}}{\text{minimize}} \quad \|\boldsymbol{C}\boldsymbol{z} - \boldsymbol{d}\|_2^2, \tag{3.1}$$

where $\boldsymbol{C}$ is a full-rank matrix of order $n \times l$ ($n \geq l$). In our case, the columns of $\boldsymbol{C}$ are those of $\boldsymbol{A}$ indexed by $\mathcal{G}^j_{\mathcal{S}}$, $\boldsymbol{d}$ is $\boldsymbol{e}_j$, and $l = |\mathcal{G}^j_{\mathcal{S}}|$.

There is further structure in our particular least-squares problem which should be exploited. As we are assuming that $\boldsymbol{A}$ is sparse, in general its columns $\boldsymbol{a}_j$ have only a few nonzeros. If we define the *shadow* of $\boldsymbol{A}$ from $\mathcal{G}^j_{\mathcal{S}}$ as the intersection of the row indices of the columns $\boldsymbol{a}_j$, $j \in \mathcal{G}^j_{\mathcal{S}}$, it is clear that, if $\boldsymbol{m}_j \in \mathcal{G}^j_{\mathcal{S}}$, the nonzero components of the residual $\boldsymbol{A}\boldsymbol{m}_j - \boldsymbol{e}_j$ from (2.2) can only occur within the intersection of the shadow of $\boldsymbol{A}$ from $\mathcal{G}^j_{\mathcal{S}}$ and row j — the remaining rows may be disregarded. Thus, both the *true* row and column dimensions of (2.2) are actually likely to be modest, especially when $|\mathcal{G}^j_{\mathcal{S}}|$ is small. This reduction in dimension is crucial to the effectiveness of the methods we shall consider.

3.1. Solving least-squares problems. Let $\boldsymbol{z}^*$ denote the minimizer of (3.1) and let $\boldsymbol{r}^*$ denote the residual $\boldsymbol{d} - \boldsymbol{C}\boldsymbol{z}^*$. Then $\boldsymbol{z}^*$ and $\boldsymbol{r}^*$ satisfy the *augmented system*

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{C} \\ \boldsymbol{C}^T & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{r}^* \\ \boldsymbol{z}^* \end{pmatrix} = \begin{pmatrix} \boldsymbol{d} \\ \boldsymbol{0} \end{pmatrix}. \tag{3.2}$$

The solution to (3.2) may be formally expressed as

$$\begin{pmatrix} \boldsymbol{r}^* \\ \boldsymbol{z}^* \end{pmatrix} = \begin{pmatrix} \boldsymbol{P}_C \boldsymbol{d} \\ (\boldsymbol{C}^T\boldsymbol{C})^{-1}\boldsymbol{C}^T\boldsymbol{d} \end{pmatrix}, \tag{3.3}$$

where the projection matrix

$$\boldsymbol{P}_C = \boldsymbol{I} - \boldsymbol{C}(\boldsymbol{C}^T\boldsymbol{C})^{-1}\boldsymbol{C}^T \tag{3.4}$$

projects into the null-space of $\boldsymbol{C}^T$. It then follows that

$$\sigma = \boldsymbol{d}^T\boldsymbol{P}_C^2\boldsymbol{d} = \boldsymbol{d}^T\boldsymbol{P}_C\boldsymbol{d}. \tag{3.5}$$

There are a variety of numerical methods for solving (3.1) (see, for instance, [20] or [4]). The most convenient in our case is to use a dense QR factorization of $\boldsymbol{C}$. A QR factorization of $\boldsymbol{C}$ is a decomposition of the form

$$\boldsymbol{C} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} = (\boldsymbol{Y} \;\; \boldsymbol{Z})\begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} = \boldsymbol{Y}\boldsymbol{R}, \tag{3.6}$$

where $\boldsymbol{Q}$ is orthogonal and $\boldsymbol{R}$ is upper triangular. It then follows that

$$\boldsymbol{P}_C = \boldsymbol{I} - \boldsymbol{Y}\boldsymbol{Y}^T = \boldsymbol{Z}\boldsymbol{Z}^T. \tag{3.7}$$

If there are relatively few columns in $\boldsymbol{C}$, it is probably better to use the Gram–Schmidt YR factorization $\boldsymbol{C} = \boldsymbol{Y}\boldsymbol{R}$ suggested by (3.6), since then we only require the $n \times l$ portion $\boldsymbol{Y}$ of $\boldsymbol{Q}$. An effective, stable scheme for computing this latter factorization, one column at a time, is given by [7]. This then gives the alternative representation

$$\sigma = \|\boldsymbol{Z}^T\boldsymbol{d}\|_2^2. \tag{3.8}$$

3.2. Adding a column of data. The residual $\|\boldsymbol{A}\boldsymbol{m}_j - \boldsymbol{e}_j\|_2$ may be reduced by allowing extra nonzeros in a column $\boldsymbol{m}_j$ of the sparse approximate inverse $\boldsymbol{M}$. The augmented $\boldsymbol{M}$ should then be a better approximation to the true inverse. Of course, each extra nonzero introduced to $\boldsymbol{m}_j$ increases the number of columns in the least-squares problem (2.2) by one and the shadow of incoming columns may increase its true row dimension.

In this section we investigate how the solution to the least-squares problem (3.1) changes if we introduce an extra column $\boldsymbol{c}$ to the data — obviously, a candidate column $\boldsymbol{c}$ will only make an impact if $\boldsymbol{c}$ intersects the shadow of the nonzeros of $(\boldsymbol{C}\ \ \boldsymbol{d})$. We denote the solution to the resulting expanded problem as $\sigma_{+\boldsymbol{c}}$, where

$$\sigma_{+\boldsymbol{c}} = \underset{\boldsymbol{z},\,\zeta}{\text{minimize}} \ \ \|\boldsymbol{C}\boldsymbol{z} + \zeta\boldsymbol{c} - \boldsymbol{d}\|_2^2. \tag{3.9}$$

Clearly, once we have decided which column to introduce, we could calculate the required minimizer, residual, and residual sum-of-squares by recomputing the QR or YR factorization of the extended matrix $(\boldsymbol{C}\ \boldsymbol{c})$. However, this is inefficient and it will normally be better to update the existing QR or YR factorization to accommodate the extra column. Good schemes to accomplish this have been proposed by [7], [15], and [20]. Since we would like to introduce nonzeros to $\boldsymbol{m}_j$ to reduce $\|\boldsymbol{A}\boldsymbol{m}_j - \boldsymbol{e}_j\|_2$ by as much as possible, it is of interest to know a priori the effect of introducing a nonzero on the residual sum-of-squares.

Huckle and Grote [18] propose the following approximation. They consider the univariate minimization problem of calculating

$$\sigma_{+\boldsymbol{c}}^{\text{approx}} = \underset{\zeta}{\text{minimize}} \ \ \|\zeta\boldsymbol{c} - \boldsymbol{r}^*\|_2^2. \tag{3.10}$$

Thus, $\sigma_{+\boldsymbol{c}}^{\text{approx}}$ indicates the gain to be made by keeping the existing components $\boldsymbol{z}^*$ fixed and minimizing solely with respect to the new component ζ. We have the following result.

LEMMA 3.1. *The solution to the least-squares problem* (3.10) *is*

$$\sigma_{+\boldsymbol{c}}^{approx} = \sigma - \frac{(\boldsymbol{c}^T\boldsymbol{r}^*)^2}{\|\boldsymbol{c}\|_2^2}, \tag{3.11}$$

and this solution satisfies the inequality

$$\sigma_{+\boldsymbol{c}} \le \sigma_{+\boldsymbol{c}}^{approx} \le \sigma. \tag{3.12}$$

Proof. The result follows by direct calculation; see [18]. □

The basis of Huckle and Grote's method is to find the smallest $\sigma_{+\boldsymbol{c}}^{\text{approx}}$ over all candidate columns $\boldsymbol{c}$, and to then introduce this column. There are many refinements possible which we shall come to shortly, but Huckle and Grote indicate that such a method is effective in practice in picking good columns to introduce.

We now return to the exact value σ_{+c} of the expanded problem once column $\boldsymbol{c}$ has been introduced. The following result shows that it may be beneficial to augment a given sparsity pattern of $\boldsymbol{M}$.

LEMMA 3.2. *The solution to the least-squares problem* (3.9) *is*

$$\sigma_{+\boldsymbol{c}} = \sigma - \frac{(\boldsymbol{c}^T\boldsymbol{P}_C\boldsymbol{d})^2}{\|\boldsymbol{P}_C\boldsymbol{c}\|_2^2} = \sigma - \frac{(\boldsymbol{c}^T\boldsymbol{r}^*)^2}{\|\boldsymbol{P}_C\boldsymbol{c}\|_2^2}. \tag{3.13}$$

Proof. The result follows, by analogy with (3.3), by observing that the matrix

$$\boldsymbol{P}_{(\boldsymbol{C}\ \boldsymbol{c})} = \boldsymbol{P}_{\boldsymbol{C}} - \frac{\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\boldsymbol{c}^T\boldsymbol{P}_{\boldsymbol{C}}}{\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|_2^2} \tag{3.14}$$

projects into the null-space of $(\boldsymbol{C}\ \ \boldsymbol{c})^T$. See also [6]. $\square$

From Lemma 3.2 we see that it is possible to calculate the potential decrease in the residual sum-of-squares following the introduction of an extra column of $\boldsymbol{C}$ *without* actually computing the corresponding residual. In particular, using the QR factors (3.6) we have

$$\sigma_{+\boldsymbol{c}} = \sigma - \frac{(\boldsymbol{c}^T\boldsymbol{r}^*)^2}{\|\boldsymbol{Z}^T\boldsymbol{c}\|_2^2} = \sigma - \frac{(\boldsymbol{c}^T\boldsymbol{r}^*)^2}{\|\boldsymbol{c}\|_2^2 - \|\boldsymbol{Y}^T\boldsymbol{c}\|_2^2}. \tag{3.15}$$

Thus, to compute $\sigma_{+\boldsymbol{c}}$ it suffices to know $\boldsymbol{Z}^T\boldsymbol{c}$ or $\boldsymbol{Y}^T\boldsymbol{c}$. However, the direct calculation of these quantities for all candidate columns $\boldsymbol{c}$ is still potentially a considerable overhead.

Fortunately, there is a more efficient way of calculating the potential decrease. Suppose that we already know $\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|_2^2$. Then we see from (3.13) that, once we have formed $\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{d}$, $\sigma_{+\boldsymbol{c}}$ may be calculated merely by forming a dot product with $\boldsymbol{c}$ — the relationship (3.3) shows that $\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{d}$ is readily available as $\boldsymbol{r}^*$. Thus, the dominant cost of computing $\sigma_{+\boldsymbol{c}}$ is in computing $\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|_2^2$. But rather than compute this quantity directly, it is simple to *update* its value following a change in $\boldsymbol{C}$. For, suppose we have augmented $\boldsymbol{C}$ by introducing the new column $\boldsymbol{c}_{\text{new}}$. Then, from (3.14),

$$\boldsymbol{P}_{(\boldsymbol{C}\ \boldsymbol{c}_{\text{new}})} = \boldsymbol{P}_{\boldsymbol{C}} - \frac{\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}_{\text{new}}\boldsymbol{c}_{\text{new}}^T\boldsymbol{P}_{\boldsymbol{C}}}{\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}_{\text{new}}\|_2^2}. \tag{3.16}$$

Thus,

$$\|\boldsymbol{P}_{(\boldsymbol{C}\ \boldsymbol{c}_{\text{new}})}\boldsymbol{c}\|_2^2 = \boldsymbol{c}^T\boldsymbol{P}_{(\boldsymbol{C}\ \boldsymbol{c}_{\text{new}})}\boldsymbol{c} = \|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|_2^2 - \frac{(\boldsymbol{c}^T\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}_{\text{new}})^2}{\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}_{\text{new}}\|_2^2}, \tag{3.17}$$

and the value of $\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|_2^2$ following a change in $\boldsymbol{C}$ may be updated rather than recomputed provided the product $\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}_{\text{new}}$ is known. But this latter term is available as a byproduct of the update to the YR factorization of $(\boldsymbol{C}\ \boldsymbol{c}_{\text{new}})$. For, we may write

$$(\boldsymbol{C}\ \ \boldsymbol{c}_{\text{new}}) = (\boldsymbol{Y}\ \ \boldsymbol{y})\begin{pmatrix} \boldsymbol{R} & \boldsymbol{r} \\ \boldsymbol{0} & \rho \end{pmatrix}. \tag{3.18}$$

In exact arithmetic, the new components $\boldsymbol{r}$, ρ, and $\boldsymbol{y}$ satisfy

$$\boldsymbol{r} = \boldsymbol{Y}^T\boldsymbol{c}_{\text{new}}, \quad \rho = \|\boldsymbol{c}_{\text{new}} - \boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{c}_{\text{new}}\|_2, \quad \text{and} \quad \boldsymbol{y} = \frac{\boldsymbol{c}_{\text{new}} - \boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{c}_{\text{new}}}{\|\boldsymbol{c}_{\text{new}} - \boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{c}_{\text{new}}\|_2}; \tag{3.19}$$

the $\boldsymbol{r}$, ρ, and $\boldsymbol{y}$ computed using the algorithm of [7] are good approximations to (3.19). But then

$$\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}_{\text{new}} = \boldsymbol{c}_{\text{new}} - \boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{c}_{\text{new}} = \rho\boldsymbol{y} \tag{3.20}$$

and

$$\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}_{\text{new}}\|_2^2 = \rho^2 \tag{3.21}$$

are readily available. Hence, we may rewrite (3.17) as

$$\|\boldsymbol{P}_{(\boldsymbol{C}\ \boldsymbol{c}_{\text{new}})}\boldsymbol{c}\|_2^2 = \|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|_2^2 - (\boldsymbol{y}^T\boldsymbol{c})^2. \tag{3.22}$$

The cost of performing the update (3.17) is that of performing the dot product between the sparse vector $\boldsymbol{c}$ and $\boldsymbol{y}$, and is comparable to computing the numerator $\boldsymbol{c}^T\boldsymbol{r}^*$ in (3.13).

As before, if there are a number of candidate columns $\boldsymbol{c}_i$ which we might add to $\boldsymbol{C}$, a good strategy is to add the column which maximizes the decrease in the residual sum-of-squares. The above schemes allow us to calculate the decrease for each candidate column. While such a scheme is perfectly feasible, it still involves an overhead compared to the method of Huckle and Grote [18]. Huckle and Grote reject using the exact decreases as they perceive this as being expensive, but, as we have shown, this is not the case. As our numerical experiments indicate (see section 5.3), the small overhead often pays handsome dividends in reducing the overall number of columns required.

Lemma 3.2 also indicates that $\sigma_{+\boldsymbol{c}}^{\text{approx}}$ may be a bad approximation to $\sigma_{+\boldsymbol{c}}$ if $\|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|$ is small relative to $\|\boldsymbol{c}\|$, that is, if $\boldsymbol{c}$ is predominantly in the null-space of the columns of $\boldsymbol{C}$.

4. Practical approximate-inverse preconditioners.

We now return to the calculation of sparse approximate-inverse preconditioners introduced in section 2.

4.1. The least-squares problem in hand.

We have already shown that solving the least-squares problem (2.2) is equivalent to solving a least-squares problem of the form (3.1). The structure of $\boldsymbol{A}$ has a significant impact on the update of the YR factorization following the introduction of an additional component in $\mathcal{G}_{\mathcal{S}}^j$ and in the computation of the approximate or exact costs, (3.10) or (3.9). Clearly, adding a column $\boldsymbol{a}_+$ of $\boldsymbol{A}$ introduces an extra column to $\boldsymbol{C}$; moreover, $\boldsymbol{C}$ has additional rows corresponding to nonzeros in $\boldsymbol{a}_+$ which do not lie in the shadow of $\boldsymbol{A}$ from $\mathcal{G}_{\mathcal{S}}^j$. The values of $\sigma_{+\boldsymbol{a}_j}^{\text{approx}}$ or $\sigma_{+\boldsymbol{a}_j}$ need only be calculated for columns $\boldsymbol{a}_j$ which have nonzeros in the shadow of $\boldsymbol{A}$ from $\mathcal{G}_{\mathcal{S}}^j$, as all other columns offer no (local) improvement to the residual.

An approximation to each column of the inverse may be calculated, independently, by introducing nonzeros one at a time. Each successive nonzero $m_{i,j}$ is chosen to minimize $\sigma_{+\boldsymbol{a}_i}^{\text{approx}}$ or $\sigma_{+\boldsymbol{a}_i}$ as appropriate, and the process is stopped when either

$$\|\boldsymbol{A}\boldsymbol{m}_j - \boldsymbol{e}_j\| \le \epsilon \tag{4.1}$$

or a predefined maximum limit $m_{\max}$ on the number of allowed nonzeros in a column is exceeded. Huckle and Grote [18] suggest that an initial sparsity pattern $\mathcal{S}$ be given, but we view this as unnecessary, preferring to start with $\mathcal{S} = \emptyset$ and allowing each nonzero i to enter on the basis of the size of $\sigma_{+\boldsymbol{a}_i}^{\text{approx}}$ or $\sigma_{+\boldsymbol{a}_i}$. Of course, if we are solving a sequence of closely related linear systems, it may be useful to use a previously successful sparsity pattern to initialize the current calculation. However, as a large part of the cost of finding a good pattern is in forming the YR factorization, the gain from such a strategy is unlikely to be as significant as it might first seem.

Huckle and Grote also propose a potentially useful saving by introducing more than one nonzero at once. They choose nonzeros which give the smallest $\sigma_{+\boldsymbol{a}_i}^{\text{approx}}$; it is just as appropriate to choose those which minimize $\sigma_{+\boldsymbol{a}_i}$. At most a user-specified number s nonzeros may enter at any time, but those which enter are required to correspond to a $\sigma^{\text{approx}}_{+\boldsymbol{a}_i}$ which is no larger than the mean such value. The reader should note, however, that such a scheme may be far from optimal, as there is no guarantee that the value of (3.1) following a multiple update is any smaller than that following a single update corresponding to the smallest $\sigma_{+\boldsymbol{a}_i}$. Note that (3.22) may be generalized to handle multiple updates since

$$\|\boldsymbol{P}_{(\boldsymbol{C}\ \boldsymbol{C}_{\text{new}})}\boldsymbol{c}\|_2^2 = \|\boldsymbol{P}_{\boldsymbol{C}}\boldsymbol{c}\|_2^2 - \|\boldsymbol{Y}^T_{\text{new}}\boldsymbol{c}\|_2^2, \tag{4.2}$$

where $\boldsymbol{C}_{\text{new}}$ columns are appended to $\boldsymbol{C}$ and $\boldsymbol{Y}_{\text{new}}$ are the resulting new columns of $\boldsymbol{Y}$.

Finally, Huckle and Grote prefer the Householder QR factorization (3.6) to the Gram–Schmidt factorization in which $\boldsymbol{Q}$ is stored as a product of Householder matrices. The Householder factorization may be updated slightly more efficiently as extra columns of data are introduced, but we prefer the YR factorization as it allows us direct access to the rows and columns of $\boldsymbol{Y}$. This is important when forming the residual

$$\boldsymbol{r}^* = \boldsymbol{d} - \boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{d}, \tag{4.3}$$

needed by (3.9) and (3.10), when, as in our application (2.2), $\boldsymbol{d}$ is a column of the identity matrix. For then $\boldsymbol{Y}^T\boldsymbol{d}$ is merely a row of $\boldsymbol{Y}$; if $\boldsymbol{Q}$ is stored as a product of Householder matrices, the required row may be recovered as a sequence of Householder products.[1] A further possibility that we have not investigated is that the residual $\boldsymbol{r}^*_{+\boldsymbol{c}_{\text{new}}}$ after the column $\boldsymbol{c}_{\text{new}}$ is appended to $\boldsymbol{C}$ is given by

$$\boldsymbol{r}^*_{+\boldsymbol{c}_{\text{new}}} = \boldsymbol{r}^* - (\boldsymbol{y}^T\boldsymbol{d})\boldsymbol{y}. \tag{4.4}$$

This recurrence leads to an efficient method for calculating $\boldsymbol{r}^*$ so long as $\boldsymbol{y}$ is directly available, as it is when the YR factorization is used.

4.2. Row-inverse methods. It may sometimes be preferable to compute an approximate-inverse preconditioner by rows rather than by columns. For instance, in an extreme case, the true inverse might be of the form

$$\begin{pmatrix} \times & & & & \\ \times & \times & & & \\ \times & & \times & & \\ \cdot & & & \cdot & \\ \times & & & & \times \end{pmatrix}. \tag{4.5}$$

Here, the first column is completely dense while each of the remaining columns have a single nonzero. Thus, the computation of the approximate inverse will be completely dominated by the cost of computing an approximation to the first column. If, on the other hand, the inverse had been found by rows, each row of the inverse involves at most two nonzeros which leads to a considerably better-balanced computation.

We may construct such an inverse by considering the alternative

$$\underset{\boldsymbol{M}\in\mathcal{G}_{\mathcal{S}}}{\text{minimize}}\ \|\boldsymbol{M}\boldsymbol{A}-\boldsymbol{I}\|_F^2 \equiv \sum_{j=1}^{n} \underset{\boldsymbol{m}_j\in\mathcal{G}^j_{\mathcal{S}}}{\text{minimize}}\ \|\boldsymbol{m}_j^T\boldsymbol{A}-\boldsymbol{e}_j^T\|_2^2 \tag{4.6}$$

[1] In fact, it would be sensible to update the existing $\boldsymbol{Q}^T\boldsymbol{d}$ whenever a new Householder matrix is created.

to (2.1), where now $\mathcal{G}_{\mathcal{S}}^{j}$ is the space of all n-vectors which have entries in the positions indexed by the set $\{i : (j,i) \in \mathcal{S}\}$ and $\boldsymbol{m}_j^T$ are the rows of $\boldsymbol{M}$ (see [18]). This is equivalent to finding the matrix $\boldsymbol{M}^T$ to

$$\underset{\boldsymbol{M} \in \mathcal{G}_{\mathcal{S}}}{\text{minimize}} \quad \|\boldsymbol{A}^T\boldsymbol{M}^T - \boldsymbol{I}\|_F^2, \tag{4.7}$$

and thus all of the discussions in our preceding sections are appropriate so long as one replaces $\boldsymbol{A}$ and $\boldsymbol{M}$ by their transposes. We shall refer to a preconditioner chosen by (2.1) as a *right* preconditioner, while that chosen by (4.6) is a *left* preconditioner.

4.3. Block methods. Matrices which arise in practice may often be reduced to block (upper) triangular form. It is straightforward to derive approximate-inverse preconditioners which exploit this structure.

Suppose that we can find permutation matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ for which

$$\boldsymbol{PAQ} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \vdots & \boldsymbol{A}_{1l} \\ \boldsymbol{0} & \boldsymbol{A}_{22} & \vdots & \boldsymbol{A}_{2l} \\ \cdots & \cdots & \cdot & \cdots \\ \boldsymbol{0} & \boldsymbol{0} & \vdots & \boldsymbol{A}_{ll} \end{pmatrix}. \tag{4.8}$$

Such a permutation may be found efficiently, for instance, using the algorithm of [22] (see [12, Chapter 6], for details). Then, letting $\bar{\boldsymbol{b}} = \boldsymbol{Pb}$ and $\boldsymbol{x} = \boldsymbol{Q}\bar{\boldsymbol{x}}$, and partitioning

$$\bar{\boldsymbol{b}} = \begin{pmatrix} \bar{\boldsymbol{b}}_1 \\ \bar{\boldsymbol{b}}_2 \\ \cdot \\ \bar{\boldsymbol{b}}_l \end{pmatrix} \quad \text{and} \quad \bar{\boldsymbol{x}} = \begin{pmatrix} \bar{\boldsymbol{x}}_1 \\ \bar{\boldsymbol{x}}_2 \\ \cdot \\ \bar{\boldsymbol{x}}_l \end{pmatrix}, \tag{4.9}$$

we may solve (1.1) by block back-substitution of the form

$$\bar{\boldsymbol{x}}_i = \boldsymbol{A}_{ii}^{-1}\left(\bar{\boldsymbol{b}}_i - \sum_{j=i+1}^{l} \boldsymbol{A}_{ij}\bar{\boldsymbol{x}}_j\right) \quad \text{for} \quad i = l, \ldots, 1. \tag{4.10}$$

Now suppose that we find an approximation $\boldsymbol{M}_{ii}$ to the inverse of each diagonal block $\boldsymbol{A}_{ii}^{-1}$. Then we might instead choose to use the approximation

$$\bar{\boldsymbol{x}}_i = \boldsymbol{M}_{ii}\left(\bar{\boldsymbol{b}}_i - \sum_{j=i+1}^{l} \boldsymbol{A}_{ij}\bar{\boldsymbol{x}}_j\right) \quad \text{for} \quad i = l, \ldots, 1 \tag{4.11}$$

to (4.10). We thus propose using the matrix

$$\boldsymbol{M}_B = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{M}_{11}^{-1} & \boldsymbol{A}_{12} & \vdots & \boldsymbol{A}_{1l} \\ \boldsymbol{0} & \boldsymbol{M}_{22}^{-1} & \vdots & \boldsymbol{A}_{2l} \\ \cdots & \cdots & \cdot & \cdots \\ \boldsymbol{0} & \boldsymbol{0} & \vdots & \boldsymbol{M}_{ll}^{-1} \end{pmatrix}^{-1} \boldsymbol{P} \tag{4.12}$$

as a block preconditioner, where $\boldsymbol{M}_{ii}^{-1}$ should be interpreted here as a generalized inverse of $\boldsymbol{M}_{ii}$. A matrix–vector product $\boldsymbol{M}_B\boldsymbol{b}$ is formed via (4.11); the product $\boldsymbol{y} = \boldsymbol{M}_B^T\boldsymbol{z}$ is given by

$$\bar{\boldsymbol{y}}_i = \boldsymbol{M}_{ii}^T \left(\bar{\boldsymbol{z}}_i - \sum_{j=1}^{i-1} \boldsymbol{A}_{ji}^T \bar{\boldsymbol{y}}_j \right) \quad \text{for} \quad i = 1, \ldots, l, \tag{4.13}$$

where $\bar{\boldsymbol{z}}_i$ and $\bar{\boldsymbol{y}}_i$ are the obvious partitions of the vectors $\bar{\boldsymbol{z}} = \boldsymbol{Q}^T\boldsymbol{z}$ and $\boldsymbol{y} = \boldsymbol{P}^T\bar{\boldsymbol{y}}$.

Clearly, we are particularly interested in picking the approximate inverse of $\boldsymbol{A}_{ii}$ using the techniques discussed thus far in this paper. There are a number of advantages in using the block form. First, each diagonal block may be significantly smaller than $\boldsymbol{A}$, and thus one would expect to find the required approximate inverses more cheaply. Second, the quality of the preconditioner may improve. For example, if the original matrix could be permuted to the form

$$\begin{pmatrix} \alpha & 0 \\ \boldsymbol{a} & \boldsymbol{A} \end{pmatrix},$$

a nonblocked sparse approximate-inverse method may expend considerable effort approximating the first column of its inverse

$$\begin{pmatrix} 1/\alpha & 0 \\ \boldsymbol{u} & \boldsymbol{A}^{-1} \end{pmatrix}, \quad \text{where} \quad \boldsymbol{u} = -(1/\alpha)\boldsymbol{A}^{-1}\boldsymbol{a};$$

if the block form is exploited, there is no need to approximate $\boldsymbol{u}$ directly. The SHERMAN problems in section 5.5 are examples for which the block form has large blocks but for which considerable savings are possible. Finally, each block may be processed independently. The disadvantage is that some of the parallelism in forming matrix–vector products is lost.

5. Numerical experiments.

5.1. The test problems.

In this section we present results for a range of problems arising from real scientific and industrial applications. Our aim is to illustrate, and thus to get a better understanding of, the practical behavior of the methods we have considered in this paper. The test examples are all taken either from the widely used Harwell–Boeing collection of sparse matrices [13] or the recent collection of Davis [8]. The matrices in both these collections are available via anonymous ftp.[2] A brief description of each of the test problems is given in Table 5.1. The problems which we have chosen from these test sets were (with the exception of problem BP200) used by either Chow and Saad [5] or Huckle and Grote [18]. All our numerical experiments were performed on each of the examples listed in Table 5.1. Since giving full results for each test problem would present the reader with an indigestible amount of data, we feel it is more helpful to only give comprehensive results for a subset of the main test set and, where appropriate, to augment these with results for other matrices. The matrices in the selected subset are marked with a $*$ in Table 5.1. The subset was chosen to represent different application areas and does not include matrices for which our methods proved unsuitable (see section 5.8).

[2]Available via anonymous ftp from 130.246.8.22 and the directory pub/harwell_boeing for the Harwell–Boeing matrices, and from ftp.cis.ufl.edu and the directory pub/umfpack/matrices for the Davis collection.

TABLE 5.1
The test problems (n =order of matrix, nz =number of entries in matrix, $*$ indicates a problem in the selected subset).

Identifier	n	nz	Description/discipline
ORSREG1$*$	2205	14133	Oil reservoir simulation. Jacobian matrix $21 \times 21 \times 5$ grid
ORSIRR1	1030	6858	As ORSREG1 but unnecessary cells coalesced to give a coarser mesh.
ORSIRR2$*$	886	5970	As ORSIRR1 but further coarsening of grid.
PORES2$*$	1224	9613	Reservoir simulation.
PORES3	532	3474	Reservoir simulation.
SHERMAN1$*$	1000	3750	Oil reservoir simulation. $10 \times 10 \times 10$ grid.
SHERMAN2$*$	1080	23094	Oil reservoir simulation. $6 \times 6 \times 5$ grid.
SHERMAN3	5005	20033	Oil reservoir simulation. $35 \times 11 \times 13$ grid.
SHERMAN4	1104	3786	Oil reservoir simulation. $16 \times 23 \times 3$ grid.
SHERMAN5	3312	20793	Oil reservoir simulation. $16 \times 23 \times 3$ grid.
SAYLR4$*$	3564	22316	3D reservoir simulation.
RAEFSKY1	3242	294276	Incompressible flow in pressure-driven pipe.
BP200	822	3802	Basis matrix from application of simplex method to a linear programming problem.
WEST0497	497	1727	Modeling of chemical engineering plant.
GRE216A	216	876	Simulation studies in computer systems.
GRE512	512	2192	Simulation studies in computer systems.
GRE1107	1107	5664	Simulation studies in computer systems.
NNC261	261	1500	Nuclear reactor core modeling.

The numerical experiments were performed on a SUN Sparc station 10 using double-precision arithmetic, which gives roughly 16 decimal digits of accuracy. CPU timings are all given in seconds. In all our reported experiments, preconditioning was on the right except for problem PORES2 for which, following [18], preconditioning was on the left. The iterative methods employed during the experiments were the conjugate gradient squared method (CGS), the restarted generalized minimum residual method (GMRES(m), where m is the number of iterations between restarts), the biconjugate gradient method (BiCG), and the biconjugate gradient stabilized method (BiCGSTAB). A description of these methods is given, for example, by [1]. When using GMRES(m), for each problem we tried setting $m = 20$ and 50; we present results for whichever of these values gave the most rapid convergence. The codes we used to implement the iterative methods are included in Release 12 of the Harwell Subroutine Library [17] as routines `MI03`, `MI04`, `MI05`, and `MI06`, respectively. Each of these routines uses reverse communication so that every time a matrix–vector product $\boldsymbol{Ax}$ or a preconditioning operation $\boldsymbol{Mx}$ is required, control is returned to the user. This allows the user to take full advantage of the sparsity and structure of $\boldsymbol{A}$ and $\boldsymbol{M}$ and of vectorization or parallelism.

The stopping criterion used for the iterative methods was $\|\boldsymbol{b} - \boldsymbol{Ax}\| < 10^{-8}\|\boldsymbol{r}_0\|$. The initial guess for the iterative solvers was always $\boldsymbol{x}_0 = \boldsymbol{0}$, $\tilde{\boldsymbol{r}}_0 = \boldsymbol{r}_0 = \boldsymbol{b}$. These choices were made for consistency with the experiments reported on in [18]. If the right-hand-side vector $\boldsymbol{b}$ was not supplied, a random vector was generated.

5.2. An implementation. The methods outlined in sections 2–4 form the basis of a new code, `MI12`, in the Harwell Subroutine Library [17]. This implementation

TABLE 5.2
Cost of computing the approximate-inverse preconditioner via σ_{+c}^{approx} *versus* σ_{+c}. † *indicates that not all columns satisfy* (4.1).

			$nz(\boldsymbol{M})/nz(\boldsymbol{A})$		CPU time	
Matrix	ϵ	$m_{\max}$	σ_{+c}^{approx}	σ_{+c}	σ_{+c}^{approx}	σ_{+c}
ORSREG1	0.4	50	0.681	0.648	15.84	15.94
ORSIRR2	0.36	50	0.846	0.661	4.78	4.24
PORES2	0.2	150	5.018†	2.226	373.39	62.46
SHERMAN1	0.4	100	1.474†	0.722	7.40	2.57
SHERMAN2	0.4	200	1.585†	0.926	1112.39	318.27
SAYLR4	0.2	150	4.469†	2.555†	384.70	188.94
RAEFSKY1	0.3	50	0.087	0.084	665.45	1067.42

allows the columns of the sparse approximate-inverse preconditioner, or its block form, to be formed independently. Options include using either the exact (3.9) or approximate (3.10) gains to predict which entry to introduce next, allowing more than one entry to be introduced at once into the inverse, and using the method in either single-block or multiple-block mode—the block form is determined efficiently by `MI12`. Additionally, the maximum number of nonzeros allowed in a column of the approximate inverse and the required convergence tolerance ϵ in (4.1) are under the user's control. All the numerical experiments reported in the following sections were performed using `MI12`.

5.3. Approximate versus exact improvement. We first report the results of experiments to assess how well approximating the improvement that is made by introducing an extra nonzero into a column of the approximate inverse performs in practice. In these experiments, except for problem SHERMAN2, values for the stopping criteria ϵ for the columns of the inverse and for the maximum number $m_{\max}$ of allowed nonzeros in a column of the inverse were those used by Huckle and Grote [18]. For SHERMAN2, Huckle and Grote take $\epsilon = 0.4$ and $m_{\max} = 50$, but with these values they fail to achieve convergence to the requested accuracy when $\boldsymbol{M}$ is used as preconditioner for the iterative solvers. We found it necessary to increase $m_{\max}$ to 200 to get convergence for this problem.

In Table 5.2 the ratios of nonzeros $nz(\boldsymbol{M})/nz(\boldsymbol{A})$ and the CPU times (in seconds) for computing $\boldsymbol{M}$ are presented using σ_{+c}^{approx}, given by (3.10), and σ_{+c}, defined by (3.9). A † in Table 5.2 indicates that some of the columns of $\boldsymbol{M}$ did not satisfy the termination test (4.1). Table 5.3 gives the number of iterations and time taken by the iterative schemes to achieve convergence using the computed $\boldsymbol{M}$ as a preconditioner. Here, and elsewhere, ‡ indicates convergence was not attained in 1000 iterations.

For some of the test problems (including ORSREG1 and ORSIRR2) we found there was little to choose between using σ_{+c}^{approx} and σ_{+c}, but, in general, our experience was that using σ_{+c}^{approx} was considerably more expensive than using σ_{+c}, and it frequently led to the matrix $\boldsymbol{M}$ having a greater number of nonzeros. For only one of the test problems, RAEFSKY1, were significant savings made by using σ_{+c}^{approx}. This matrix is more dense that the other matrices we considered (see Table 5.1) and the sparse approximate-inverse preconditioner has far fewer entries than the original matrix. In this case, the overhead involved in computing the exact decrease (3.9) for each candidate column is significantly more than in using the approximation (3.10) of Huckle and Grote. For matrix SHERMAN3 with $\epsilon = 0.2$ and $m_{max} = 100$ (the values used by Huckle and Grote), `MI12` failed to compute $\boldsymbol{M}$ and returned an error message that the matrix was so ill conditioned that the sparse approximate inverse

TABLE 5.3
Convergence results for σ_{+c}^{approx} *versus* σ_{+c}. ‡ *indicates convergence was not attained in* 1000 *iterations.*

		σ_{+c}^{approx}		σ_{+c}	
Matrix	Method	Iterations	CPU time	Iterations	CPU time
ORSREG1	BiCG	84	3.48	86	3.64
	CGS	49	2.09	48	2.00
	BiCGSTAB	54	2.34	56	2.30
	GMRES(20)	82	3.01	85	3.04
ORSIRR2	BiCG	69	1.21	69	1.13
	CGS	47	0.83	40	0.68
	BiCGSTAB	48	0.88	39	0.66
	GMRES(20)	90	1.13	75	0.98
PORES2	BiCG	71	4.20	57	2.10
	CGS	70	3.99	40	1.47
	BiCGSTAB	48	2.74	44	1.60
	GMRES(50)	239	9.71	114	3.05
SHERMAN1	BiCG	100	1.72	80	1.22
	CGS	85	1.43	55	0.85
	BiCGSTAB	71	1.22	51	0.75
	GMRES(20)	203	4.23	113	1.45
SHERMAN2	BiCG	286	16.64	29	1.28
	CGS	‡		14	0.61
	BiCGSTAB	‡		10	0.44
	GMRES(20)	‡		16	0.48
SAYLR4	BiCG	108	13.58	110	10.33
	CGS	83	10.70	96	9.32
	BiCGSTAB	71	9.10	87	8.39
	GMRES(50)	312	40.02	519	57.77
RAEFSKY1	BiCG	111	28.94	110	29.00
	CGS	99	25.98	97	25.38
	BiCGSTAB	86	22.64	86	22.64
	GMRES(50)	438	82.51	376	70.07

was likely to be worthless. Because of this, SHERMAN3 was omitted from the rest of our experiments.

5.4. Multiple updates. As we discussed in section 4.1, Huckle and Grote [18] suggest that useful savings may be achieved by introducing more than one new entry at once into a column of the approximate inverse. In particular, in their reported numerical results, they allow up to five entries to be introduced simultaneously. We denote by $\boldsymbol{M}_s$ the sparse approximate-inverse preconditioner computed by introducing a maximum of s entries at once. In Tables 5.4 and 5.5 we compare computing and using $\boldsymbol{M}_1$ with $\boldsymbol{M}_5$ (σ_{+c} is used in these and all subsequent numerical experiments). We found, in general, that $\boldsymbol{M}_5$ had more nonzeros than $\boldsymbol{M}_1$ and took longer to compute. The quality of the resulting preconditioners when used with the iterative solvers was comparable. The only exceptions to these general findings were again for problem RAEFSKY1. For this problem, the CPU time for computing $\boldsymbol{M}_5$ was significantly less than for $\boldsymbol{M}_1$. Experiments with other values of s revealed similar findings, with a general deterioration in performance as s increased.

5.5. The block method versus the single-block method. In Tables 5.6 and 5.7 we give some results which compare treating the matrix as a single block with using the block form discussed in section 4.3. Test problems ORSREG1, ORSIRR2, PORES2, and SAYLR4 were found to be irreducible and for these problems the differences in the computed results for the two methods are caused by tiebreaking. For

Table 5.4
Cost of computing the approximate-inverse preconditioner $\boldsymbol{M}_1$ *versus* $\boldsymbol{M}_5$. † *indicates that not all columns satisfy* (4.1).

			$nz(\mathbf{M}s)/nz(\boldsymbol{A})$		CPU time	
Matrix	ϵ	$m_{\max}$	$s=1$	$s=5$	$s=1$	$s=5$
ORSREG1	0.4	50	0.648	0.817	15.94	18.00
ORSIRR2	0.36	50	0.661	0.750	4.24	4.35
PORES2	0.2	150	2.226	2.488	62.46	71.31
SHERMAN1	0.4	100	0.722	0.938	1.59	3.04
SHERMAN2	0.4	200	0.926	1.232	318.27	359.50
SAYLR4	0.2	150	2.555†	3.740†	188.94	193.69
RAEFSKY1	0.3	50	0.084	0.087	1067.42	796.07

Table 5.5
Convergence results for $\boldsymbol{M}_1$ *versus* $\boldsymbol{M}_5$.

		$\boldsymbol{M}_1$		$\boldsymbol{M}_5$	
Matrix	Method	Iterations	CPU time	Iterations	CPU time
ORSREG1	BiCG	86	3.64	83	3.60
	CGS	48	2.00	46	2.04
	BiCGSTAB	56	2.30	53	2.39
	GMRES(20)	85	3.04	82	3.18
ORSIRR2	BiCG	69	1.13	69	1.19
	CGS	40	0.68	40	0.67
	BiCGSTAB	39	0.66	40	0.70
	GMRES(20)	75	0.98	75	0.98
PORES2	BiCG	57	2.10	64	2.56
	CGS	40	1.47	41	1.62
	BiCGSTAB	44	1.60	40	1.57
	GMRES(50)	114	3.05	104	2.78
SHERMAN1	BiCG	80	1.22	77	1.21
	CGS	55	0.85	54	0.83
	BiCGSTAB	51	0.75	50	0.81
	GMRES(50)	113	1.45	112	1.42
SHERMAN2	BiCG	29	1.26	29	1.41
	CGS	14	0.61	13	0.64
	BiCGSTAB	10	0.44	13	0.61
	GMRES(20)	16	0.48	19	0.60
SAYLR4	BiCG	110	10.33	115	13.56
	CGS	96	9.32	96	11.59
	BiCGSTAB	87	8.39	95	11.44
	GMRES(50)	519	57.77	529	64.17
RAEFSKY1	BiCG	110	29.00	110	29.88
	CGS	97	25.38	110	29.18
	BiCGSTAB	86	22.64	65	22.27
	GMRES(50)	376	70.07	405	78.60

problems SHERMAN1 and SHERMAN2, the block form (4.8) has one large block and the remaining blocks are all of order 1. For such problems we found the two approaches computed comparable preconditioners, but there were considerable time-savings in using the block form. For test examples BP200 and WEST0497, the block form (4.8) had a large number of blocks of order greater than 1, and for these examples using the block form was not only much faster than treating the matrix as a single block but also gave a much improved preconditioner. For example, for BP200, treating the matrix as a single block, the preconditioner took 27.54 seconds to compute while using the block form took only 0.35 seconds. Moreover, when used with the iterative solvers, for this problem the preconditioner computed using the single-block method failed

Table 5.6

Cost of computing the approximate-inverse preconditioner treating the matrix as a single block versus using the block form. † *indicates that not all columns satisfy* (4.1).

			$nz(\boldsymbol{M})/nz(\boldsymbol{A})$		CPU time	
Matrix	ϵ	$m_{\max}$	*single*	*block*	*single*	*block*
ORSREG1	0.4	50	0.648	0.648	15.94	16.64
ORSIRR2	0.36	50	0.662	0.662	4.24	4.25
PORES2	0.2	150	2.226	2.226	62.46	62.46
SHERMAN1	0.4	100	0.722	0.722	2.57	1.59
SHERMAN2	0.4	200	0.926	0.916	318.27	208.19
SAYLR4	0.2	150	2.556†	2.556†	188.94	186.21
BP200	0.4	50	2.459†	1.133	27.54	0.35
WEST0497	0.4	100	2.577†	1.260	19.66	1.36

Table 5.7

Convergence results for the preconditioner obtained by treating the matrix as a single block versus the preconditioner obtained using the block form. ‡ *indicates convergence was not attained in* 1000 *iterations.*

		Single block		Block form	
Matrix	Method	Iterations	CPU time	Iterations	CPU time
ORSREG1	BiCG	86	3.64	86	3.61
	CGS	48	2.00	48	2.00
	BiCGSTAB	56	2.30	56	2.30
	GMRES(20)	85	3.04	85	3.04
ORSIRR2	BiCG	69	1.13	69	1.12
	CGS	40	0.68	41	0.68
	BiCGSTAB	39	0.66	42	0.70
	GMRES(20)	75	0.98	79	1.00
PORES2	BiCG	57	2.10	57	2.10
	CGS	40	1.47	40	1.42
	BiCGSTAB	44	1.60	44	1.55
	GMRES(50)	114	3.05	114	3.03
SHERMAN1	BiCG	80	1.22	82	1.47
	CGS	55	0.85	55	0.99
	BiCGSTAB	51	0.75	50	0.89
	GMRES(20)	113	1.45	113	1.49
SHERMAN2	BiCG	29	1.26	27	1.34
	CGS	14	0.61	9	0.45
	BiCGSTAB	10	0.44	9	0.43
	GMRES(50)	16	0.48	15	0.47
SAYLR4	BiCG	110	10.33	110	10.68
	CGS	96	9.32	96	9.69
	BiCGSTAB	87	8.39	87	8.51
	GMRES(50)	519	57.77	519	57.19
BP200	BiCG	‡		27	0.55
	CGS	‡		19	0.35
	BiCGSTAB	‡		27	0.30
	GMRES(50)	‡		26	0.37
WEST0497	BiCG	‡		21	0.20
	CGS	‡		14	0.13
	BiCGSTAB	‡		13	0.12
	GMRES(50)	‡		21	0.15

to give convergence within the imposed limit of 1000 iterations while that computed using the block form gave convergence in less than 30 iterations.

5.6. ILU(0) versus sparse approximate-inverse preconditioners. A class of preconditioners which is frequently used when solving sparse unsymmetric linear

TABLE 5.8
Cost of ILU(0) versus sparse approximate-inverse preconditioner. † *indicates that not all columns satisfy* (4.1).

	ILU(0)	Approximate-inverse preconditioner			
Matrix	CPU time	ϵ	$m_{\max}$	$nz(\boldsymbol{M})/nz(\boldsymbol{A})$	CPU time
ORSREG1	2.53	0.3	50	1.257	29.60
ORSIRR2	0.41	0.3	50	1.154	9.28
PORES2	0.85	0.2	150	2.226	62.46
SHERMAN1	0.35	0.3	25	1.144	3.13
SHERMAN2	0.99	0.4	200	0.916	206.51
SAYLR4	6.32	0.15	300	5.460	2667.49†
RAEFSKY1	23.08	0.3	50	0.084	1067.42
BP200	0.22	0.4	50	1.133	0.35
WEST0497	0.11	0.4	100	1.260	1.36
GRE512	0.11	0.4	100	7.498	27.03†

systems is based on ILUs of the matrix $\boldsymbol{A}$. We now present some results which compare our proposed sparse approximate-inverse preconditioner with an ILU preconditioner. The code we used in our numerical experiments to generate an ILU preconditioner is included in Release 12 of the Harwell Subroutine Library [17] as routine `MI11`. The code first finds a permutation matrix $\boldsymbol{Q}$ so that the matrix $\boldsymbol{QA}$ has nonzeros on the diagonal. If no such permutation can be found, the method breaks down. Assuming such a permutation exists, an incomplete factorization of the permuted matrix $\boldsymbol{QA} = \boldsymbol{LU} + \boldsymbol{E}$, where $\boldsymbol{E}$ is some error matrix, and $\boldsymbol{L}$ and $\boldsymbol{U}$ have the same nonzero structure as the lower and upper parts of $\boldsymbol{A}$, respectively, is formed. The ILU(0) preconditioner is then $\boldsymbol{P} = (\boldsymbol{LU})^{-1}\boldsymbol{Q}$.

The incomplete factorization of $\boldsymbol{QA}$ will breakdown if, at any stage k of the elimination process, the kth diagonal entry is zero. To prevent this breakdown, `MI11` checks the size of the diagonal entry against the entries in its row and column and, if it is too small, the value of the diagonal entry is increased using a user-defined control parameter.

In Tables 5.8 and 5.9 results are given for the ILU(0) preconditioner computed using `MI11` and for the sparse approximate-inverse preconditioner $\boldsymbol{M}$ (computed using the block form and σ_{+c}). For problems ORSREG1, ORSIRR2, PORES2, SHERMAN1, and SAYLR4, the stopping criteria ϵ for the columns of the sparse inverse preconditioner was chosen so that the numbers of iterations required by the iterative schemes using $\boldsymbol{M}$ were generally comparable to those required using the ILU(0) preconditioner. We see that, for these problems, the time required to generate $\boldsymbol{M}$ is substantially more than is needed by the ILU(0) factorization.

For some of the test problems, including PORES2 and SAYLR4, the number of nonzeros in $\boldsymbol{M}$ is significantly greater than the number in $\boldsymbol{A}$ (and hence in the ILU(0) preconditioner). This can mean that each preconditioning operation with $\boldsymbol{M}$ is much more expensive on a scalar machine than with the ILU(0) preconditioner. Thus, even if the number of iterations required using the two preconditioners is similar, the convergence times using the sparse approximate-inverse preconditioner may exceed those for the ILU(0) preconditioner. The only test problem for which $\boldsymbol{M}$ was much sparser than the ILU(0) preconditioner was problem RAEFSKY1. For this problem, the number of iterations required using $\boldsymbol{M}$ as a preconditioner was significantly larger than for the ILU(0) preconditioner, but for each of the iterative methods apart from GMRES, the time needed for convergence using $\boldsymbol{M}$ was less than that using the ILU(0) preconditioner.

TABLE 5.9
Convergence results for ILU(0) *versus approximate-inverse preconditioner.* ‡ *indicates convergence was not attained in* 1000 *iterations.*

		ILU(0)		$\boldsymbol{M}$	
Matrix	Method	Iterations	CPU time	Iterations	CPU time
ORSREG1	BiCG	67	3.59	69	3.33
	CGS	40	2.15	39	1.87
	BiCGSTAB	37	2.00	45	2.16
	GMRES(20)	60	2.63	62	3.61
ORSIRR2	BiCG	52	1.19	55	1.06
	CGS	32	0.65	32	0.59
	BiCGSTAB	31	0.63	32	0.62
	GMRES(20)	58	0.86	56	0.80
PORES2	BiCG	50	1.58	57	2.10
	CGS	38	1.13	40	1.47
	BiCGSTAB	37	1.40	44	1.60
	GMRES(50)	42	1.44	114	3.05
SHERMAN1	BiCG	49	0.92	60	1.00
	CGS	39	0.72	39	0.64
	BiCGSTAB	35	0.62	38	0.62
	GMRES(20)	61	0.92	76	1.05
SHERMAN2	BiCG	84	4.09	27	1.34
	CGS	44	1.98	9	0.45
	BiCGSTAB	256	11.24	9	0.43
	GMRES(50)	48	1.70	15	0.46
SAYLR4	BiCG	60	5.25	64	9.31
	CGS	50	4.25	48	7.08
	BiCGSTAB	41	3.51	42	6.15
	GMRES(50)	70	6.45	89	11.60
RAEFSKY1	BiCG	36	15.90	103	12.14
	CGS	31	12.26	85	10.30
	BiCGSTAB	27	10.71	72	8.66
	GMRES(50)	35	8.71	265	25.14
BP200	BiCG	74	1.16	27	0.50
	CGS	94	1.39	19	0.35
	BiCGSTAB	99	1.49	17	0.30
	GMRES(50)	39	0.66	26	0.37
WEST0497	BiCG	‡		21	0.20
	CGS	‡		14	0.13
	BiCGSTAB	655	5.33	13	0.12
	GMRES(50)	129	1.05	21	0.15
GRE512	BiCG	‡		144	2.75
	CGS	‡		126	2.41
	BiCGSTAB	‡		122	2.26
	GMRES(20)	‡		363	4.35

For problem GRE512, the ILU(0) factorization was successfully computed using `MI11`, without the need to increase any diagonal entries to prevent breakdown. However, when used as a preconditioner, the ILU(0) factorization gave much poorer results for this problem than were achieved using the sparse approximate-inverse preconditioner. For SHERMAN2, BP200, and WEST0497, `MI11` issued a warning that the value of one or more diagonal entries had been increased and the resulting ILU(0) factorization gave poor convergence (or no convergence) when used with the iterative solvers.

We note that more sophisticated ILU preconditioners are available (see, for example, [21]). These methods aim to trade the speed advantages of ILU(0) with the reliability of exact factorization methods. A full assessment of these methods would be welcomed.

5.7. Parallel preconditioning. A significant advantage of the sparse approximate-inverse preconditioner $\boldsymbol{M}$ over the ILU(0) preconditioner is that the computation of $\boldsymbol{M}$ is inherently parallel, since its columns are calculated independently of one another. If there are k processors available, an obvious way of distributing the work between the processors is to assign the calculation of columns 1 to k/n to processor 1, that for columns $k/n+1$ to $2k/n$ to processor 2, and so on. A disadvantage of this strategy is that load balancing may be very poor since we have observed in practice that $\boldsymbol{M}$ often contains a small number of columns with many more nonzeros than the remaining columns, and computing these few columns accounts for most of the computational effort in generating $\boldsymbol{M}$. For example, for the matrix SAYLR4, $\boldsymbol{M}$ has only 28 columns with more than 100 nonzeros and the time taken to compute these columns is 85.90 seconds, compared with 188.94 seconds for the whole matrix.

An alternative strategy is to hold the columns in a queue and as soon as a processor becomes free, the next column in the queue is assigned to it. This has the advantage that considerable load balancing can be achieved even if the processors are not all identical. We have performed some experiments to simulate this strategy for the case of identical processors. Let *stime* denote the time required to start the computation (this is on a single processor). Let *coltime*(i) be the time taken by a processor to compute column i of $\boldsymbol{M}$, and let S_l be the set of column indices which are assigned to processor l $(l = 1, 2, ..., k)$. Define

$$Ideal = stime + \left(\sum_{i=1}^{n} coltime(i) \right) / k$$

and

$$Actual = stime + \max_{l} \left(\sum_{i \in S_l} coltime(i) \right).$$

Ideal is the time to compute $\boldsymbol{M}$ on k processors assuming that, after the initial start-up, each processor performs the same amount of work. *Actual* is the time which will actually elapse before all the processors have finished. Comparing *Actual* with *Ideal* gives an indication of efficiency of computing $\boldsymbol{M}$ in parallel.

We see from Table 5.10 that, for a small number of processors (up to about 16), the *Actual* and *Ideal* times generally differ by only a small amount. As the number of processors is increased, the difference between *Actual* and *Ideal* remains small provided most of the columns of $\boldsymbol{M}$ are of a similar length (for example, problems ORSREG1 and RAEFSKY1). However, if a few columns of $\boldsymbol{M}$ have many more nonzeros than the remaining columns, the speedup which can be achieved by increasing the number of processors is much less. For example, for problem PORES2, increasing the number of processors from 16 to 64 only reduces the *Actual* time by a factor of approximately 2.6. If a column is assigned to a single processor, the *Actual* time cannot be reduced beyond $stime + \max_l coltime(l)$. This is illustrated by problems SHERMAN2 and SAYLR4. The only hope for improvement in these cases is to allocate more than one processor for the calculation of difficult columns.

We note that, for most of our test problems, provided sufficiently many processors are available and provided that each processor is capable of holding the entire matrix, the potential parallel processing times are quite competitive with the ILU(0) times. Encouraged by these predictions, we intend to implement a parallel version of `MI12`.

Table 5.10

Ideal time versus actual time (in seconds) to compute the sparse approximate-inverse preconditioner in parallel.

Matrix	ϵ	$m_{\max}$	*stime*	$\max_l$ *coltime*(l)	Number of processors	*Ideal* time	*Actual* time
ORSREG1	0.3	50	0.00	0.15	1	29.60	29.60
					8	3.69	3.70
					16	1.88	1.89
					64	0.52	0.53
					256	0.18	0.21
ORSIRR2	0.3	50	0.00	0.15	1	9.28	9.28
					8	1.12	1.12
					16	0.56	0.56
					64	0.14	0.23
					256	0.03	0.16
PORES2	0.2	150	0.00	0.81	1	64.26	64.26
					8	7.71	7.71
					16	3.85	4.55
					64	0.96	1.72
					256	0.24	0.96
SHERMAN1	0.3	25	0.00	0.03	1	3.13	3.13
					8	0.39	0.40
					16	0.20	0.20
					64	0.05	0.05
					256	0.01	0.03
SHERMAN2	0.4	200	0.12	6.22	1	208.19	208.19
					8	26.10	26.10
					16	13.11	13.11
					64	3.37	6.34
					256	0.93	6.34
RAEFSKY1	0.3	50	0.96	1.55	1	1069.59	1069.59
					8	134.39	134.61
					16	67.67	67.94
					64	17.64	17.93
					256	5.13	5.51
SAYLR4	0.15	300	0.12	26.45	1	2667.49	2667.49
					8	333.42	333.42
					16	166.78	166.78
					64	41.78	41.79
					256	10.54	26.57

5.8. Limitations of the sparse approximate-inverse preconditioner. The results which we have presented so far have shown that the sparse approximate-inverse preconditioner can be effective when used with the standard iterative methods BiCG, CGS, BiCSTAB, and GMRES, although it can be very expensive to compute for large and difficult problems, particularly in a sequential environment, and it may not give any improvement over an ILU(0) preconditioner. During our experiments we also found matrices for which the sparse approximate-inverse preconditioner failed to converge when used with the iterative methods. In particular, we failed to get convergence for problems NNC261 and GRE1107 (the ILU(0) preconditioner also failed to converge for these problems). To try and understand why the sparse approximate-inverse preconditioner was failing for these problems, we used the Harwell Subroutine Library [17] routine `MA48` [14] to compute the exact inverses. `MA48` is a direct solver and, once the LU factors of $\boldsymbol{A}$ have been determined, it allows repeated calls to the solve routine `MA48C` to solve for different right-hand-sides. By choosing the right-hand-side vector to be each of the columns of the identity matrix in turn, we were able to compute $\boldsymbol{A}^{-1}$.

Applying MA48 in this way, we found that the inverse of the NNC261 matrix is dense, having 60136 entries. Furthermore, the absolute values of many of these entries are not small, so that it is not possible to approximate the inverse well by a sparse matrix. Similarly, the inverse of the GRE1107 matrix is found to be completely dense, again with many entries with large absolute values.

Thus, in hindsight, it is easy to see why a sparse approximate-inverse method has trouble with these matrices. Of course, it is difficult to know a priori whether such a method is likely to succeed. However, the disadvantage of not knowing if the method will succeed is shared by its competitors.

6. Other methods.

In the methods considered so far, the sparsity pattern for a particular column directly defines its numerical values. In [5], Chow and Saad propose a different strategy in which an approximate solution to the system

$$\boldsymbol{A}\boldsymbol{m}_j = \boldsymbol{e}_j \tag{6.1}$$

is sought using a few iterations of a suitable iterative method (in their case, GMRES is used). Once an adequate approximation to the required solution is found, a subset of its components are reset to zero (dropped). The authors suggest a number of dropping rules. Of course, solving each equation (6.1) is hardly easier than solving the original problem (1.1) if high accuracy is required, so the authors accept very low accuracy. The authors acknowledge that achieving even low accuracy can be difficult if $\boldsymbol{A}$ is ill conditioned. They therefore propose using the columns of the preconditioner computed so far to precondition the remaining equations, although they warn that this reduces the scope for parallelism. Alternatively, they suggest improving the preconditioner using refinement. That is, given a preconditioner $\boldsymbol{M}^{(k-1)}$, they construct $\boldsymbol{M}^{(k)}$ by approximately solving each equation (6.1), using an iterative method preconditioned by $\boldsymbol{M}^{(k-1)}$, and then dropping appropriate values.

Chow and Saad indicate that their approach is effective but give little indication as to the cost of their methods. We would be surprised if these methods were cheaper than the methods considered elsewhere in this paper, and it is not obvious to us how the more sophisticated variants may be parallelized effectively.

Recently, Benzi and Tůma [3] have proposed a method for constructing approximate inverse preconditioners based not on norm minimization techniques but on constructing two sets of biconjugate vectors, performed incompletely to preserve sparsity. Their algorithm results in an approximate sparse triangular factorization of $\boldsymbol{A}^{-1}$. Being an incomplete matrix factorization method, the procedure resembles classical ILU-type techniques, but it has the advantage that applying the preconditioner requires only sparse matrix–vector products. Furthermore, the cost of computing the preconditioner is reported as being only two or three times that of computing the ILU(0) preconditioner and, when used with an iterative solver, it gives rates of convergence which are, in general, comparable with those for the ILU(0) preconditioner. However, it is not clear how much scope there is for constructing the preconditioner in parallel. This method will require further investigation.

7. Conclusions.

We have considered sparse approximate-inverse preconditioners for the iterative solution of nonsymmetric linear systems of equations. We have proposed a number of enhancements to existing methods which appear to improve their performance significantly. In comparison with ILU preconditioner methods, the sparse approximate-inverse methods are significantly more expensive to use on a single processor machine. However, the sparse approximate-inverse methods can be successful when ILU preconditioners fail, and we have indicated that the imbalance in the computation times can be redressed when the methods are used in parallel. We intend to provide a full parallel implementation in due course.

8. Availability of the codes. The codes `MI03`, `MI04`, `MI05`, `MI06`, `MI11`, and `MI12` are all written in standard FORTRAN 77. The codes are included in Release 12 of the Harwell Subroutine Library [17]. Anyone interested in using the codes should contact the Harwell Subroutine Library Manager: Dr. S. Roberts, AEA Technology, Building 552 Harwell, Didcot, Oxfordshire, OX11 0RA, United Kingdom; telephone (+44) 1235 434714; fax (+44) 1235 434340; e-mail scott.roberts@aeat.co.uk, who will provide license details.

Acknowledgments. We should like to thank Iain Duff and John Reid, of the Rutherford Appleton Laboratory, and two anonymous referees for their helpful comments on earlier drafts of this paper.

REFERENCES

[1] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[2] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Math., 22 (1982), pp. 127–140.

[3] M. Benzi and M. Tůma, *A Sparse Approximate Inverse Preconditioner for Nonsymmetric Linear Systems*, Tech. report 653, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1995.

[4] A. Björck, *Least squares methods*, in Handbook of Numerical Analysis 1, P. G. Ciarlet and J. L. Lions, eds., Elsevier–North Holland, Amsterdam, 1989.

[5] E. Chow and Y. Saad, *Approximate Inverse Preconditioners for General Sparse Matrices*, Research report UMSI 94/101, University of Minnesota Supercomputer Institute, Minneapolis, MN, 1994.

[6] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank, *Approximate inverse preconditionings for sparse linear systems*, Internat. J. Comput. Math., 44 (1992), pp. 91–110.

[7] J. W. Daniel, W. B. Gragg, L. C. Kaufman, and G. W. Stewart, *Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization*, Math. Comput., 30 (1976), pp. 772–795.

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

[9] E. de Doncker and A. K. Gupta, *Coarse grain preconditioned conjugate gradient solver for large sparse systems*, in Proc. Seventh SIAM Conference on Parallel Processing for Scientific Computing, D. H. Bailey, ed., SIAM, Philadelphia, PA, 1995, pp. 472–477.

[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] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, PA, 1991.

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

[13] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release* 1*)*, Tech. report RAL-92-086, Rutherford Appleton Laboratory, Chilton, UK, 1992.

[14] I. S. Duff and J. K. Reid, *The design of MA*48*, a code for the direct solution of sparse unsymmetric linear systems of equations*, ACM Trans. Math. Software, 22 (1996), pp. 187–226.

[15] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.

[16] M. Grote and T. Huckle, *Effective parallel preconditioning with sparse approximate inverses*, in Proc. Seventh SIAM Conference on Parallel Processing for Scientific Computing, D. H. Bailey, ed., SIAM, Philadelphia, PA, 1995, pp. 466–471.

[17] Harwell Subroutine Library, *A Catalogue of Subroutines (Release* 12*)*, Advanced Computing Department, Harwell Laboratory, Harwell, UK, 1995.

[18] T. Huckle and M. Grote, *A New Approach to Parallel Preconditioning with Sparse Approximate Inverses*, Tech. report SCCM-94-03, School of Engineering, Stanford University, Stanford, CA, 1994.

[19] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.

[20] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice–Hall, Englewood Cliffs, NJ, 1974.

[21] Y. Saad, *ILUT: A dual threshold incomplete LU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.

[22] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.