SIAM J. SCI. COMPUT.
Vol. 36, No. 2, pp. A609–A633

ON POSITIVE SEMIDEFINITE MODIFICATION SCHEMES FOR INCOMPLETE CHOLESKY FACTORIZATION*

JENNIFER SCOTT† AND MIROSLAV TŮMA‡

Abstract. Incomplete Cholesky factorizations have long been important as preconditioners for use in solving large-scale symmetric positive-definite linear systems. In this paper, we focus on the relationship between two important positive semidefinite modification schemes that were introduced to avoid factorization breakdown, namely, the approach of Jennings and Malik and that of Tismenetsky. We present a novel view of the relationship between the two schemes and implement them in combination with a limited memory approach. We explore their effectiveness using extensive numerical experiments involving a large set of test problems arising from a wide range of practical applications.

Key words. sparse matrices, sparse linear systems, positive-definite symmetric systems, iterative solvers, preconditioning, incomplete Cholesky factorization

AMS subject classifications. 65F05, 65F50

DOI. 10.1137/130917582

1. Introduction.

Iterative methods are widely used for the solution of large sparse symmetric linear systems of equations $Ax = b$. To increase their robustness, the system matrix A generally needs to be transformed by preconditioning. For positive-definite systems, an important class of preconditioners is represented by incomplete Cholesky (IC) factorizations, that is, factorizations of the form LL^T in which some of the fill entries (entries that were zero in A) that would occur in a complete factorization are ignored. Over the last 50 years or so, many different algorithms for computing incomplete factorizations have been proposed and used to solve problems from a wide range of application areas. A brief historical overview of some of the key developments may be found in [44].

An important step in the practical use of algebraic preconditioning based on incomplete factorizations came with the 1977 paper of Meijerink and van der Vorst [35]. They proved the existence of the IC factorization, for arbitrary choices of the sparsity pattern, for the class of symmetric M-matrices; this property was later also proved for H-matrices with positive diagonal entries [34, 49]. However, for a general symmetric positive-definite A (including many examples that arise from practical applications) the incomplete factorization can break down because of the occurrence of zero or negative pivots.

The problem of breakdown is well known and various approaches have been employed to circumvent it. Indeed, shortly after the work of Meijerink and van der Vorst, Kershaw [29] showed that the IC factorization of a general symmetric positive-definite matrix from a laser fusion code can suffer seriously from breakdowns. To

*Submitted to the journal's Methods and Algorithms for Scientific Computing section April 18, 2013; accepted for publication (in revised form) December 13, 2013; published electronically April 8, 2014.

http://www.siam.org/journals/sisc/36-2/91758.html

†Scientific Computing Department, Rutherford Appleton Laboratory, Harwell Oxford, Oxfordshire, OX11 0QX, UK (jennifer.scott@stfc.ac.uk). This author's work was supported by EPSRC grant EP/I013067/1.

‡Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodarenskou Vezi 2, Czech Republic (tuma@cs.cas.cz). This author's work was partially supported by Grant Agency of the Czech Republic project 13-06684 S.

complete the factorization, Kershaw locally replaced nonpositive diagonal entries by a small positive number. The hope was that if only a few of the diagonal entries had to be replaced, the resulting factorization would still yield a useful preconditioner. The use of perturbations helped popularize incomplete factorizations, although local perturbations with no relation to the overall matrix can lead to large growth in the entries and the subsequent Schur complements and hence to unstable preconditioners. Heuristics have been proposed for how to choose the perturbations, and a discussion of the possible effects on more general incomplete factorizations (and that can occur even in the symmetric positive-definite case) can be found in [9].

Manteuffel [34] introduced an alternative strategy that involved the notion of a shifted factorization. He proposed factorizing the diagonally shifted matrix $A+\alpha I$ for some positive α. (Note that provided α is large enough, the incomplete factorization always exists.) Diagonal shifts were used in some implementations even before Manteuffel (see [36, 37]) and, although currently the only way to find a suitable global shift is by trial and error, provided an α can be found that is not too large, the approach is surprisingly effective and remains well used. One of the best implementations of IC factorization is the ICFS code of Lin and Moré [30]. It uses an efficient loop for changing the shift within a prescribed memory approach. The use of prescribed memory builds on the work of Jones and Plassman [25, 26] and of Saad [41]. Given $p \geq 0$, the ICFS code retains the $n_j + p$ largest entries in the lower triangular part of the jth column of L (where n_j is the number of entries in the lower triangular part of column j of A) and it uses only memory as the criterion for dropping entries. Reported results for large-scale trust region subproblems indicate that allowing additional memory can substantially improve performance on difficult problems.

A very different technique to avoid factorization breakdown is that of using a semidefinite modification scheme. As in any incomplete factorization, entries are discarded as the factorization progresses but the idea here is to use the discarded entries to guarantee preservation of positive definiteness of the reduced matrices so that the method is breakdown free. In this paper we consider two such schemes. The first is that of Jennings and Malik [23, 24], who, in the mid 1970s, introduced a modification strategy to prevent factorization breakdown for symmetric positive-definite matrices arising from structural engineering. Jennings and Malik were motivated only by the need to compute a preconditioner without breakdown and not by any consideration of the conditioning of the preconditioned system. Their work was extended by Ajiz and Jennings [1], who discussed dropping (rejection) strategies as well as implementation details. Variations of the Jennings–Malik scheme were adopted by engineering communities and it is recommended in, for example, [39] and used in experiments in [6] to solve some hard problems, including the analysis of structures and shape optimization.

The second scheme we consider is that proposed in the early 1990s by Tismenetsky [48], with significant later improvements by Suarjana and Law [47] and Kaporin [27]. The Tismenetsky approach has been used to provide robust preconditioners for some real-world problems; see, for example, [2, 3, 31, 32]. However, despite the fact that the computed preconditioners frequently outperform those from other known incomplete factorization techniques, as Benzi remarks in his authoritative survey paper [4], Tismenetsky's idea "has unfortunately attracted surprisingly little attention." Benzi also highlights a serious drawback of the scheme, which is that its memory requirements can be prohibitively large. (In some cases, more than 70 percent of the storage required for a complete Cholesky factorization is needed; see also [7].)

In this paper, we seek to gain a better understanding of the Jennings–Malik and Tismenetsky semidefinite modifications schemes and to explore the relationship between them. In section 2, we introduce the schemes and then, in section 3, we present new theoretical results that compare the 2-norms of the modifications to A that each approach makes. In section 4, we propose a memory-efficient variant of the Tismenetsky approach, optionally combined with the use of drop tolerances and the Jennings–Malik modifications to reduce factorization breakdowns. In section 5, we report on extensive numerical experiments in which we aim to isolate the effects of the modifications so as to assess their usefulness in the development of robust algebraic incomplete factorization preconditioners. Finally, we draw some conclusions in section 6.

2. Positive semidefinite modification schemes.

2.1. The Jennings–Malik scheme. The scheme proposed by Jennings and Malik [23, 24] can be interpreted as modifying the factorization dynamically by adding to A simple symmetric positive semidefinite matrices, each having just four nonzero entries. At each stage, we compute a column of the factorization and then modify the subsequent Schur complement. For $j = 1$, we consider the matrix A and, for $1 < j < n$, we obtain the jth Schur complement by applying the previous $j - 1$ updates and possible additional modifications. Throughout our discussions, we denote the Schur complement of order $n - j + 1$ that is computed on the jth step by $\hat{A}$ and let $\hat{A}_j$ be the first column of $\hat{A}$ (corresponding to column j of A). The jth column of the incomplete factor L is obtained from $\hat{A}_j$ by dropping some of its entries (for example, using a drop tolerance or the sparsity pattern). The Jennings–Malik scheme modifies the corresponding diagonal entries every time an off-diagonal entry is discarded. Specifically, if the nonzero entry $\hat{a}_{ij}$ of $\hat{A}_j$ is to be discarded, the Jennings–Malik scheme adds to A a modification (or cancellation) matrix of the form

$$E_{ij} = e_i e_i^T \gamma |\hat{a}_{ij}| + e_j e_j^T \gamma^{-1} |\hat{a}_{ij}| - e_i e_j^T \hat{a}_{ij} - e_j e_i^T \hat{a}_{ij}. \tag{2.1}$$

Here the indices i, j are global indices (that is, they relate to the original matrix A). E_{ij} has nonzero entries $\gamma|\hat{a}_{ij}|$ and $\gamma^{-1}|\hat{a}_{ij}|$ in the ith and jth diagonal positions, respectively, and entry $-\hat{a}_{ij}$ in the (i, j) and (j, i) positions. The scalar γ may be chosen to keep the same percentage change to the diagonal entries $\hat{a}_{ii}$ and $\hat{a}_{jj}$ that are being modified (see [1]). Alternatively, γ may be set to 1 (see [19]) and this is what we employ in our numerical experiments (but see also the weighted strategy in [14]). A so-called relaxed version of the form

$$E'_{ij} = \omega e_i e_i^T \gamma |\hat{a}_{ij}| + \omega e_j e_j^T \gamma^{-1} |\hat{a}_{ij}| - e_i e_j^T \hat{a}_{ij} - e_j e_i^T \hat{a}_{ij}, \tag{2.2}$$

with $0 \le \omega \le 1$, was proposed by Hladík, Reed, and Swoboda [19].

It is easy to see that the modification matrix E_{ij} is symmetric positive semidefinite (while for $0 \le \omega < 1$, E'_{ij} is indefinite). We use $\omega = 1$. The sequence of these dynamic changes leads to a breakdown-free factorization that can be expressed in the form

$$A = LL^T - E,$$

where L is the computed incomplete factor and E is a sum of positive semidefinite matrices with nonpositive off-diagonal entries and is thus positive semidefinite.

2.2. Tismenetsky scheme. The second modification scheme we wish to consider is that of Tismenetsky [48]. A matrix-based formulation with significant improvements and theoretical foundations was later supplied by Kaporin [27] (see also [28]).

The Tismenetsky scheme is a matrix decomposition of the form

$$A = (L + R)(L + R)^T - E, \tag{2.3}$$

where L is a lower triangular matrix with positive diagonal entries that is used for preconditioning, R is a strictly lower triangular matrix with small entries that is used to stabilize the factorization process, and the error matrix E has the structure

$$E = RR^T. \tag{2.4}$$

Consider the decomposition locally. At the jth step, the first column of the computed Schur complement $\hat{A}_j$ can be decomposed into a sum of two vectors each of length $n - j + 1$,

$$l_j + r_j,$$

such that $l_j^T r_j = 0$ (with the first entry in l_j nonzero), where l_j (respectively, r_j) contains the entries that are retained (respectively, not retained) in the incomplete factorization. At step $j + 1$ of the standard factorization, the Schur complement of order $n - j$ is updated by subtracting the outer product of the pivot row and column. That is, by subtracting

$$(l_j + r_j)(l_j + r_j)^T.$$

The Tismenetsky incomplete factorization does not compute the full update as it does not subtract

$$E_j = r_j r_j^T. \tag{2.5}$$

Thus, the positive semidefinite modification E_j is implicitly added to A. Note that we can also regard LR^T and RL^T as error matrices because R is not part of the computed preconditioner and such an approach led to a successful condition number analysis of the decomposition process in [27]. However, we feel that to get an insight into the updates it is better to consider the error locally in the form (2.3), and so we denote the error matrix at the jth step as E_j.

The obvious choice for r_j (which was proposed in the original paper [48]) is the vector of the smallest off-diagonal entries in the column (those that are smaller in magnitude than a chosen drop tolerance). Then in a right-looking formulation,[1] at each stage implicitly adding E_j is combined with the standard steps of the Cholesky factorization, with entries dropped from the incomplete factor *after* the updates have been applied to the Schur complement. The approach is naturally breakdown free because the only modification of the Schur complement that is used in the later steps of the factorization is the addition of the positive semidefinite matrices E_j.

The fill in L can be controlled by choosing the drop tolerance to limit the size of $|l_j|$. However, it is important to note that this does not limit the memory required to compute L. A right-looking implementation of a sparse factorization is generally very demanding from the point of view of memory as it is necessary to store all the fill-in for column j until the modification is applied in the step j, as

[1] A right-looking formulation directly updates the approximate Schur complement, while a left-looking column-oriented approach computes the same decomposition but the operations are performed in a different order. Full details may be found, for example, in [38].

follows from (2.5). Hence, a left-looking implementation (or, as in [27], an upward-looking implementation) might be thought preferable. But to compute column $\hat{A}_j$ in a left-looking implementation and to apply the modification (2.5) correctly, all the vectors l_k and r_k for $k = 1, \ldots, j-1$ have to be available. Therefore, the dropped entries have to be stored throughout the left-looking factorization and the r_k may be discarded only once the factorization is finished (and similarly for an upward-looking implementation). These vectors thus represent intermediate memory. Note the need for intermediate memory is caused not just by the fill in the factorization: it is required because of the structure of the positive semidefinite modification that forces the use of the r_k. Sparsity may allow some of the r_k to be discarded before the factorization is complete. For example, if A can be preordered to have a narrow bandwidth, the updates to a column involve only a limited number of the immediately preceding columns. The columns r_k can then be successively discarded with a delay determined by the matrix bandwidth. However, in general, the total memory is as for a complete factorization, without the other tools that direct methods offer. This memory problem was discussed by Kaporin [27], who proposed using two drop tolerances $droptol1 > droptol2$. Only entries of magnitude at least $droptol1$ are kept in L and entries smaller than $droptol2$ are dropped from R; the larger $droptol2$ is, the closer the method becomes to that of Jennings and Malik. The local error matrix E then has the structure

$$E = RR^T + F + F^T,$$

where F is a strictly lower triangular matrix that is not computed, while R is used in the computation of L but is then discarded.

When drop tolerances are used, the factorization is no longer guaranteed to be breakdown free. To avoid breakdown, modifications (as in the Jennings–Malik scheme) for the entries that are dropped from R may be used. Kaporin coined the term *second order incomplete Cholesky factorization* to denote this combined strategy (but note an earlier proposal of virtually the same strategy by Suarjana and Law [47]).

Finally, consider again the left-looking implementation of the Jennings–Malik scheme where the column $\hat{A}_j$ that is computed at stage j is based on the columns computed at the previous $j-1$ stages. Standard implementations perform the updates using the previous columns of L (without the dropped entries). But the dropped entries may also be used in the computation and, if we do this, the only difference between the Jennings–Malik and Tismenetsky schemes lies in the way in which the factorization is modified to safeguard against breakdown.

2.3. Related research. A discussion of the Tismenetsky approach and a modification with results for finite-element modeling of linear elasticity problems is given in [28]. In [54], Yamazaki et al. use the Kaporin approach combined with the global diagonal shift strategy of Manteuffel [34]. In contrast to Kaporin [27], who uses the upward-looking factorization motivated by Saad [41], Yamazaki et al. employ a left-looking implementation based on the pointer strategy from Eisenstat et al. [15, 16]; moreover, they do not compensate the diagonal entries fully as in the Jennings–Malik diagonal modification strategy. There is an independent derivation of the Tismenetsky approach for IC factorizations in [53], which emphasizes the relation with the special case of incomplete QR factorization (see also [50]). In fact, this variant of the incomplete QR factorization of A in which there is no dropping in Q is equivalent to the Tismenetsky approach applied to $A^T A$. For completeness, note that a related incomplete QR factorization was introduced earlier by Jennings and Ajiz [22]. From

a theoretical point of view, the authors of [51, 52] show some additional structural properties of this type of IC factorization.

An interesting application of both the Jennings–Malik and Tismenetsky ideas to multilevel Schur complement evaluations was proposed by Janna, Ferronato, and Gambolati [21]. A block incomplete factorization preconditioner with automatic selection and tuning of the factorization parameters was recently presented by Gupta and George [18]. Rather than using a modification scheme or a global shift, Gupta and George propose switching from an IC factorization to an LDLT factorization if a pivot becomes negative. This removes the requirement that the preconditioner is positive definite. Another recent paper of interest here is that of Maclachlan, Osei-Kuffuor, and Saad [33]. Although devoted to nonsymmetric incomplete factorizations, the authors point out that a quantification of the norms of the updates could be further developed theoretically. Note that [33] discusses modifications and errors from the local point of view, as we do.

The Tismenetsky approach has been used to provide a robust preconditioner for some real-world problems; see, for example, the comparison for tasks in linear elasticity in [2], emphasizing reduced parametrization in the upward-looking implementation of Kaporin [27], diffusion equations in [31, 32], and the Stokes problem in [3]. We note, however, that there are no reported comparisons with other approaches that take into account not only iteration counts but also the size of the preconditioner.

3. Theoretical results. The importance of the size of the modifications to the matrix was emphasized by Duff and Meurant in [13]. In particular, the modifications should not be large in terms of the norm of the matrix. In this section, we consider the Jennings–Malik and Tismenetsky modifications from this point of view. We have the following simple lemma for the size of the Jennings–Malik modification.

LEMMA 3.1. *The* 2*-norm of the modification in the Jennings–Malik approach based on the fill-in entry* $\hat{a}_{ij}$ *and the update formula* (2.1) *is equal to*

$$\gamma|\hat{a}_{ij}| + \gamma^{-1}|\hat{a}_{ij}|.$$

Proof. The modification (2.1) can be written as the outer product matrix

$$E_{ij} = vv^T,$$

where $v \in \mathcal{R}^n$ has entries

$$v_k = \begin{cases} (|\hat{a}_{ij}|\gamma)^{1/2} & \text{if } k = i, \\ -sgn(\hat{a}_{ij})(|\hat{a}_{ij}|/\gamma)^{1/2} & \text{if } k = j, \\ 0 & \text{otherwise.} \end{cases}$$

The result follows from the fact that the 2-norm of vv^T is equal to v^Tv. ∎

In the following, we assume in the Jennings–Malik modification the parameter $\gamma = 1$. For the Tismenetsky approach, we introduce some further notation. Here and elsewhere we denote the reduced vector of nonzero entries in l_j by $\bar{l}_j$ with $|\bar{l}_j| = n_j + lsize = lsize'$ (where n_j is the number of entries in the lower triangular part of column j of A) and, similarly, we denote the reduced vector of nonzero entries in r_j by $\bar{r}_j$ with $|\bar{r}_j| = rsize$; in the remainder of our discussion, we use bars for the quantities corresponding to these reduced vectors. Further, let us assume that the entries in both these reduced vectors are in descending order of their magnitudes and that the magnitude of each entry in $\bar{l}_j$ is at least as large as the magnitude of the largest entry in $\bar{r}_j$. We have the following result.

LEMMA 3.2. *The 2-norm of the jth modification in the Tismenetsky approach* (2.5) *is equal to*

$$(\bar{r}_{1,j},\ldots,\bar{r}_{rsize,j})(\bar{r}_{1,j},\ldots,\bar{r}_{rsize,j})^T = \sum_{k=1}^{rsize} \bar{r}_{k,j}^2, \tag{3.1}$$

where $\bar{r}_{k,j}$ *is the kth entry of the vector* $\bar{r}_j$.

It is natural to ask how these modifications are related. To compare them, we assume that the Jennings–Malik modification is applied only to the Schur complement that corresponds to the truncated update, that is, to the submatrix

$$(\bar{r}_{1,j},\ldots,\bar{r}_{rsize,j})^T(\bar{r}_{1,j},\ldots,\bar{r}_{rsize,j}).$$

Setting $\gamma = 1$ in (2.1), the two previous simple lemmas imply the following result that explains why the Tismenetsky modification should often be considered preferable in practice. A comparison of norms in Theorem 3.3 indicates that the Jennings–Malik modification may give better results, particularly if the decomposition does not generate too much fill-in.

THEOREM 3.3. *Assume* $\hat{A}_j$ *has been computed and all but the* $lsize'$ *entries of largest magnitude are dropped from column* j *of* L. *Denote the 2-norm of the Jennings–Malik modification* (2.1) *that compensates for all the dropped entries in* $r_j r_j^T$ *by* $|JM|$ *and denote the 2-norm of the Tismenetsky modification* (2.5) *related to the remaining* $|\hat{A}_j| - lsize \equiv rsize > 1$ *entries by* $|T|$. *Then* $|JM| \le (rsize-1)|T|$.

Proof. From Lemma 3.2, the 2-norm of the Tismenetsky modification in (3.1) is given by the squared diagonal entries of the matrix $r_j r_j^T$. Each of the modifications in the Jennings–Malik approach is of the form of a submatrix (2.1) with the off-diagonal entry $\bar{r}_{k,j}\bar{r}_{l,j}$ for some $1 \le k,l \le rsize, k \ne l$. Note that the off-diagonal entry corresponds to moving the fill-in from a product of entries of the vector r_j to the diagonal of $r_j r_j^T$. The sum of the 2-norms of these modifications is equal to the overall 2-norm of the modifications and is, by Lemma 3.1 (with $\gamma = 1$), equal to

$$|JM| = 2 \times \sum_{(k,l)\in\mathcal{Z}_j, k<l} |\bar{r}_{k,j}\bar{r}_{l,j}|,$$

where $\mathcal{Z}_j$ denotes the set of pairs (i,j) of off-diagonal positions in $\hat{A}_j$ corresponding to the dropped entries for which the Jennings–Malik modification was used. Note that $\mathcal{Z}_j$ does not include the positions that are present in the final decomposition. Further, we have

$$|T| = \sum_{k=1}^{rsize} \bar{r}_{k,j}^2.$$

Using the fact that $n\sum_{i=1}^n a_i^2 \ge \left(\sum_{i=1}^n a_i\right)^2$ for any real numbers $a_1,\ldots,a_n$, we obtain the result

$$|T| + |JM| = \sum_{k=1}^{rsize} \bar{r}_{k,j}^2 + 2 \times \sum_{(k,l)\in\mathcal{Z}_j, k<l} |\bar{r}_{k,j}\bar{r}_{l,j}| \tag{3.2}$$

$$\le \sum_{k=1}^{rsize} \bar{r}_{k,j}^2 + 2 \times \sum_{1\le k<l\le rsize} |\bar{r}_{k,j}\bar{r}_{l,j}| \tag{3.3}$$

$$= \left(\sum_{k=1}^{rsize} |\bar{r}_{k,j}|\right)^2 \le rsize \times |T|,$$

and $|JM| \leq (rsize - 1)|T|$ follows. ∎

The key to understanding the relationship between the two updates is the last part of the proof. If the gap between (3.2) and (3.3) is large, the Jennings–Malik modification can be better than the Tismenetsky one. A simple example is a dense submatrix $r_j^T r_j$ that does need to be updated in the Jennings–Malik scheme but the original Tismenetsky strategy still comes with the error $r_j r_j^T$. Similarly, the Jennings–Malik modification can be beneficial for sparse matrices for which the factorization generates only a small amount of fill-in [5]. As soon as the amount of fill-in grows, the Jennings–Malik approach becomes less attractive. Note that if at stage j, we use the Jennings–Malik scheme for all the entries of $r_j r_j^T$, the number of modifications is equal to $|\mathcal{Z}_j| \leq (|\hat{A}_j| - lsize')(|\hat{A}_j| - lsize' - 1)/2$. Such a potentially large number of modifications may result in the preconditioner being far from the original matrix (and hence of poor quality). Theorem 3.3 also shows how the two modification schemes are in some sense complementary and that their norms can be far apart once $rsize$ is large. A natural consequence of the result is that we can possibly make the Cholesky decomposition with the Tismenetsky update more precise by also updating the entries that are nonzero in the actual Schur complement. These are the entries that may provide potential advantage for the Jennings–Malik modifications since they do not need to induce a diagonal modification and, similarly, they do not need to be a part of the error matrix E_j in the Tismenetsky scheme. This offers the potential for additional improvements inside this submatrix based on its structural pattern, as we will discuss in the numerical experiments section.

While we discuss here dropping in the Schur complement, our implementation of the Jennings–Malik modifications differs since we use a left-looking implementation (see section 4), whereas Theorem 3.3 describes and compares both modification approaches cumulatively for a major step j of the decomposition and suggests other possible ways of improving both approaches. A natural idea is to incorporate the Jennings–Malik approach on top of the Tismenetsky update. This may appear to be an obvious idea because it can make the Tismenetsky update sparser. However, it follows from the Courant–Fischer theorem (e.g., in [17]; see also a nice overview of properties of the sums of positive semidefinite matrices in [8]) that if a 2×2 Jennings–Malik modification of the form (2.1) is added on top of a symmetric positive semidefinite submatrix that may represent the Tismenetsky update (3.1), then both eigenvalues of the resulting matrix can only increase.

Thus using the Tismenetsky approach and then the Jennings–Malik modification to nullify some off-diagonal entries does not appear helpful. However, as Theorem 3.3 points out, the two schemes can be combined differently. Indeed, the new unified explanation of the Tismenetsky and Jennings–Malik modifications indicates two ideas that we will report on in section 5: (1) the norm of the matrix modification during the Tismenetsky update can be decreased by including some entries of RR^T and (2) the remaining off-diagonal entries of RR^T can be compensated for using the Jennings–Malik scheme. As we will see, the strategy that is the best theoretically with unlimited memory may not be the best when solving practical problems using limited memory.

An important consideration for any practical strategy for computing incomplete factorizations is limiting the number of parameters that must be set by the user while still getting an acceptably robust preconditioner. As we shall see in section 5, we have observed in our numerical experiments that the intermediate memory can, to some extent, replace the memory used for the preconditioner and this experimentally based fact motivates the following discussion. Our experiments show that if the sum

$lsize + rsize = tsize$ is kept constant (and $lsize$ is not too small in relation to $rsize$), the performance of the preconditioner is maintained. Therefore, we could require a single input parameter $tsize$ and choose $lsize$ and $rsize$ internally. Consider stage j of the factorization and the Tismenetsky update restricted to the reduced submatrix determined by the first $n_j + tsize$ components of the reduced column $\bar{A}_j$ (with the nonzero entries $\bar{a}_{k,j}$, $j \le k \le n_j + tsize$, of $\bar{A}_j$ in descending order of their magnitudes). The modification restricted to this submatrix is the "error" block

$$(\bar{a}_{n_j+lsize+1,j}, \ldots, \bar{a}_{n_j+tsize,j})^T (\bar{a}_{n_j+lsize+1,j}, \ldots, \bar{a}_{n_j+tsize,j}),$$

where $lsize$ is to be determined. Consequently, we have to find a splitting of $tsize$ into $lsize$ and $rsize$ such that the off-diagonal block that is used in the updates,

$$(\bar{a}_{n_j+lsize+1,j}, \ldots, \bar{a}_{n_j+tsize,j})^T (\bar{a}_{n_j+1,j}, \ldots, \bar{a}_{n_j+lsize,j}),$$

is not small with respect to the error block measured in a suitable norm. Recall that this diagonal error block is excluded from the actual updates. Assuming $lsize, rsize \ge 1$, a possible strategy is to choose $lsize$ such that

$$1/lsize \sum_{k=n_j}^{n_j+lsize} |\bar{a}_{kj}| \ge \beta/(tsize - lsize) \sum_{k=n_j+lsize+1}^{n_j+tsize} |\bar{a}_{kj}|$$

for some modest choice of $\beta \ge 1$. (That is, the average magnitude of the entries n_j to $n_j + lsize$ in $\bar{A}_j$ is compared to β times the average magnitude of the remaining entries.) If there is no such $lsize$ (which occurs if there is insufficient "block diagonal dominance" in the considered part of the column), $lsize$ is set to $tsize$; if there is reasonable diagonal dominance in L, $lsize$ should be smaller than $tsize$. Such a strategy provides a dynamic splitting of $tsize$ since it is determined separately for each column of L. Using the following result, $lsize$ can be found by a simple search through part of the reduced column $\bar{A}_j$.

LEMMA 3.4. *The function $f(s)$ defined as*

$$1/s \sum_{k=n_j}^{n_j+s} |\bar{a}_{kj}| - \beta/(tsize - s) \sum_{k=n_j+s+1}^{n_j+tsize} |\bar{a}_{kj}|$$

is nondecreasing for $s = 0, \ldots, tsize - 1$.

The proof follows from the fact that the nonzero entries in $\bar{A}_j$ are ordered in descending order of their magnitudes. □

4. Algorithm outline. For an algebraic preconditioner to be practical it needs to have predictable and reasonable memory demands. All implementations of the Tismenetsky–Kaporin approach known to us drop entries based only on their magnitude but this does not provide practical predictable memory demands. As in the ICFS code of Lin and Moré [30], the memory predictability in our IC implementation depends on specifying a parameter $lsize$ that limits the maximum number of nonzero off-diagonal fill entries in each column of L. In addition, we retain at most $rsize$ entries in each column of R. At each step j of the factorization, the candidate entries for inclusion in the jth column of L are sorted and the largest $n_j + lsize$ off-diagonal entries plus the diagonal are retained; the next $rsize$ largest entries form the jth column of R and all other entries are discarded. Following Kaporin, the use of drop tolerances can be included. In this case, entries in L are retained only if they are at least $droptol1$ in magnitude, while those in R must be at least $droptol2$.

ALGORITHM 4.1. Memory-limited IC decomposition.

Absolute dropping, left-looking column formulation, limited memory, Tismenetsky approach combined with Jennings–Malik modifications
Input: *Symmetric and positive definite* $A \in R^{n\times n}$*; lsize, rsize, droptol*1*, droptol*2
Initialize: $L, R \in R^{n\times n}$, $L = I, R = 0$, $w \in R^n, w = 0$
for *j=1:n*
  $w = A_{:,j}$ *! Store original sparsity pattern of column j*
  for $k < j$ *and* $L_{j,k} \neq 0$ **do**
    $A_{j:n,j} \Leftarrow A_{j:n,j} - L_{j:n,k} * L_{j,k}$ *! LL^T updates*
    $A_{j:n,j} \Leftarrow A_{j:n,j} - R_{j:n,k} * L_{j,k}$ *! RL^T updates*
  end
  for $k < j$ *and* $R_{j,k} \neq 0$ **do**
    $A_{j:n,j} \Leftarrow A_{j:n,j} - L_{j:n,k} * R_{j,k}$ *! LR^T updates*
  end
  while $k < j$ *and* $R_{j,k} \neq 0$ **do**
    while $i >= j$ *and* $R_{i,k} \neq 0$ **do**
      if $A_{i,j} \neq 0$ *! Handle RR^T entries by allowing those that cause no fill.*
        $A_{i,j} \Leftarrow A_{i,j} - R_{i,k} * R_{j,k}$
      else *! JM modification for all other off-diagonal entries.*
        $A_{j,j} \Leftarrow A_{j,j} + |R_{i,k} * R_{j,k}|$
        $A_{i,i} \Leftarrow A_{i,i} + |R_{i,k} * R_{j,k}|$
      end
    end
  end
  Put into $L_{:,j}$ *the* $\min\{lsize + n_j, n - j\}$ *entries of* $A_{:,j}$ *of largest magnitude, provided they are at least droptol*1

  Put into $R_{:,j}$ *the* $\min\{rsize, n - j\}$ *entries of* $A_{:,j}$ *that are next largest in magnitude, provided they are at least droptol*2

  Denote the entries $A_{s_1,j}, \ldots, A_{s_j,j}$ *that are not in* $L_{:,j}$ *or* $R_{:,j}$ *by* S_j
  Perfom JM modification on these entries
  for each $A_{s_k,j} \in S_j$ *with* $w_{s_k} = 0$ **do**
    $A_{j,j} \Leftarrow A_{j,j} + |L_{s_k,j}|$
    $A_{k,k} \Leftarrow A_{k,k} + |L_{s_k,j}|$
  end

  Scale $L_{j+1:n,j} \Leftarrow L_{j+1:n,j}/\sqrt{A_{j,j}}$, $R_{j+1:n,j} \Leftarrow R_{j+1:n,j}/\sqrt{A_{j,j}}$
  Set $L_{j,j} = \sqrt{A_{j,j}}$ *and reset* $w = 0$

end

Algorithm 4.1 presents an outline of our memory-limited IC factorization. It shows the basic steps but, for simplicity, omits details of our sparse implementation. Because the limited memory approach is not guaranteed to be breakdown free, in practice we combine it with using a global diagonal shift using a strategy similar to that of [30] (see [43] for details). For clarity, we omit this from the outline. The user is required to provide the memory parameters $lsize$ and $rsize$ plus the drop tolerances *droptol*1 and *droptol*2.

In our experiments (section 5.6), we will consider applying Jennings–Malik modifications in a number of different ways. They can be applied to all the entries of L and R that are smaller than the drop tolerances *droptol*1 and *droptol*2, respectively; this corresponds to lines 30–35 in the algorithm. They can also be used for the entries that correspond to the off-diagonal entries of RR^T. This use corresponds to lines 18–21. The (limited memory) Tismenetsky approach discards all entries of RR^T and corresponds to deleting lines 14–23.

5. Numerical experiments.

5.1. Test environment. All the numerical results reported on in this paper are performed (in serial) on our test machine that has two Intel Xeon E5620 processors with 24 GB of memory. Our software is written in Fortran and the ifort Fortran compiler (version 12.0.0) with option -O3 is used. The implementation of the conjugate gradient algorithm offered by the HSL routine `MI22` is employed, with starting vector $x_0 = 0$, the right-hand side vector b computed so that the exact solution is $x = 1$, and stopping criteria

$$\|A\hat{x} - b\|_2 \leq 10^{-10}\|b\|_2, \tag{5.1}$$

where $\hat{x}$ is the computed solution. In addition, for each test we impose a limit of 2000 iterations. In all the tests, we order and scale the matrix A prior to computing the incomplete factorization. Based on numerical experiments (see [43]), we use a profile reduction ordering based on a variant of the Sloan algorithm [40, 45, 46]. We also use l_2 scaling, in which the entries in column j of A are normalized by the 2-norm of column j; this scaling is chosen as it is used by Lin and Moré [30] in their ICFS incomplete factorization code and it is inexpensive and simple to apply. Note that it is important to ensure the matrix A is prescaled before the factorization process commences; other scalings are available and yield comparable results, but if no scaling is used, the effectiveness of the incomplete factorization algorithms used in this paper can be significantly affected (see [43] for some results).

We define the *iteration count* for an incomplete factorization preconditioner for a given problem to be the number of iterations required by the iterative method using the preconditioner to achieve the requested accuracy and we define the *preconditioner size* to be the number of entries $nz(L)$ in the incomplete factor L.

While we are well aware that the number of entries in the preconditioner may increase but its effectiveness decreases, in many practical situations, the mutual relation between the iteration count and preconditioner size provides an important insight into the usefulness of an incomplete factorization preconditioner if we assume that the following two important conditions are fulfilled:

1. the preconditioner is sufficiently *robust* with respect to changes to the parameters of the decomposition, such as the limit on the number of entries in a column of L and of R;
2. the time required to compute the preconditioner grows *slowly* with the problem dimension n.

We define the *efficiency* of the preconditioner P to be

$$iter \times nz(L), \tag{5.2}$$

where *iter* is the iteration count for $P = (LL^T)^{-1}$ (see [42]). Assuming the IC preconditioners $P_q = (L_q L_q^T)^{-1}$ $(q = 1, \ldots, r)$ each satisfy the above conditions, we say that for solving a given problem, P_i is the *most efficient* of the r preconditioners if

$$iter_i \times nz(L_i) \leq \min_{q \neq i}(iter_q \times nz(L_q)). \tag{5.3}$$

We use this measure of efficiency in our numerical experiments.

A weakness of this measure is that it does not taken into account the number of entries in R. We anticipate that with the number of entries in each column of L fixed, increasing the permitted number of entries in each column of R will lead to a more efficient preconditioner. However, this improvement will be at the cost of additional work in the construction of the preconditioner. Thus we record the time to compute the preconditioner together with the time for convergence of the iterative method: the sum of these will be referred to as the *total time* and will also be used to assess the quality of the preconditioner. Note that the efficiency (5.2) is independent of $nz(A)$. During the application of the iterative solver, the operations with A can be implemented by the user in various ways. (`MI22` is a reverse communication code so that it is the user's responsibility to decide how to implement matrix-vector products.) In this study, a simple matrix-vector product routine is used with the lower triangular part of A held in compressed sparse column format: we have not attempted to perform either the matrix-vector products or the application of the preconditioner in parallel and all times are serial times.

Our test problems are real positive-definite matrices of order at least 1000 taken from the University of Florida Sparse Matrix Collection [10]. Many papers on preconditioning techniques and iterative solvers select a small set of test problems that are somehow felt to be representative of the applications of interest. However, our interest is more general and we want to test the different ideas and approaches on as wide a range of problems as we can. Thus we took all such problems and then discarded any that were diagonal matrices and, where there was more than one problem with the same sparsity pattern, we chose only one representative problem. This resulted in a test set of 153 problems of order up to 1.5 million. Following initial experiments, 8 problems were removed from this set as we were unable to achieve convergence to the required accuracy within our limit of 2000 iterations without allowing a large amount of fill. The same set of tests problems is used in each of the experiments reported on in this paper. To assess performance on our test set, we use performance profiles [12]. A performance profile measures the relative performance of two or more preconditioners on a set $\mathcal{S}$ of problems. Let $e_{k,P}$ be the efficiency of using preconditioner P to solve problem k and define the efficiency performance ratio to be $ratio_{k,P} = e_{k,P}/\min\{e_{k,P_i} : \text{for all } P_i\}$. If the number of problems in $\mathcal{S}$ is N, the efficiency performance profile for P

$$\rho_P(\tau) = (1/N)\,|\,\{k \in \mathcal{S} : ratio_{k,P} \leq \tau\}|$$

is the probability that an efficiency performance ratio $ratio_{k,P}$ is within a factor τ of the best possible ratio. For instance, $\rho_P(1)$ gives the fraction of the test problems for which P is the most efficient preconditioner and $\rho_P(2)$ gives how often P can get results with an efficiency that is within twice that of the best preconditioner. The closer ρ_P is to 1, the greater the probability that preconditioner P can solve all problems from $\mathcal{S}$. By plotting the curves $\rho_{P_i}(\tau)$ on a single plot we can easily compare them and deduce information about the relative performance of the respective preconditioners.

5.2. No intermediate memory, without Jennings–Malik modifications. We first present results for *lsize* varying and $rsize = 0$, without Jennings–Malik modifications for the discarded entries. We set the drop tolerance *droptol*1 to zero. This is very similar to the ICFS code of Lin and Moré [30]. The efficiency and

iteration performance profiles are given in Figure 1. Note that the asymptotes of the performance profile provide a statistic on reliability and as the curve for $lsize = 5$ lies below the others on the right-hand axis of the profiles in Figure 1, this indicates poorer reliability for small $lsize$. We see that increasing $lsize$ improves reliability and reduces the number of iterations for convergence but the efficiency is not very sensitive to the choice of $lsize$ (although, of course, the time to compute the factorization and the storage for L increase with $lsize$). This suggests that when performing numerical experiments it is not necessarily appropriate to consider just one of the counts used in (5.2) without also checking its relation to the other. If the preconditioner is to be used in solving a sequence of problems (so that the time to compute the incomplete factorization becomes an insignificant part of the total time compared to the case of a single problem), the user may want to choose the parameter settings to reduce either the iteration count or the preconditioner size, depending on which they consider to be the most important.

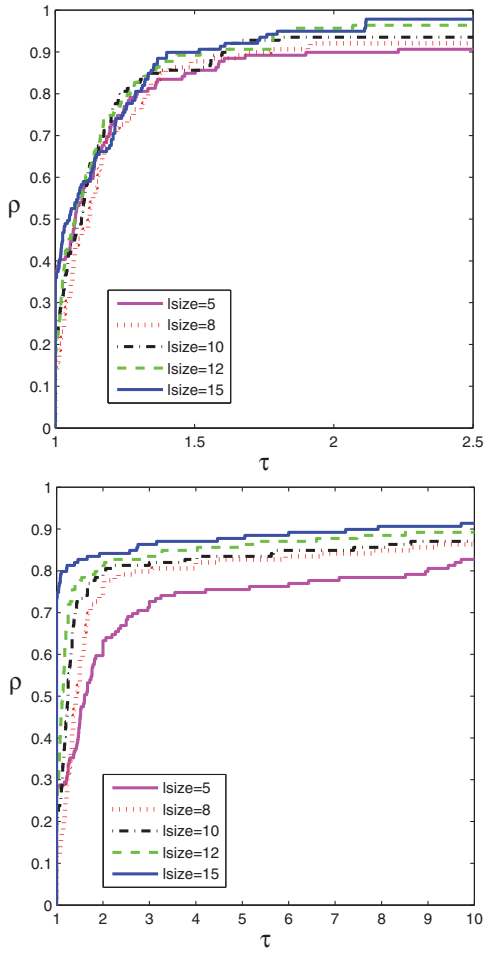

FIG. 1. *Efficiency (top) and iteration (bottom) performance profiles for lsize varying and* $rsize = 0$.

5.3. No intermediate memory, with Jennings–Malik modifications. We now explore the effects of Jennings–Malik modifications (still with $rsize = 0$). We first consider the structure-based approach originally proposed by Jennings and Malik that compensates for *all* the entries that are not retained in the factor. (Only $lsize$ fill entries are retained, with no tolerance-based dropping.) Our findings are presented in Figure 2. We see that the best results are without using standard Jennings–Malik modifications (SJM = F) and that if it is used (SJM = T), increasing $lsize$ has little effect on the efficiency (but improves the reliability). The advantage of using the modifications is that for the majority of the test problems, the factorization is breakdown free. However, a closer look at the results shows that the penalty for this breakdown-free property can be a poor quality preconditioner. Whereas using a diagonal shift led to convergence failure for just two of our test problems, with Jennings–Malik modifications there were 11 convergence failures ($lsize = 10$). In Table 1, we present detailed results for some of our test problems that used a nonzero diagonal shift. We report the number of shifts used, the number of iterations of CG

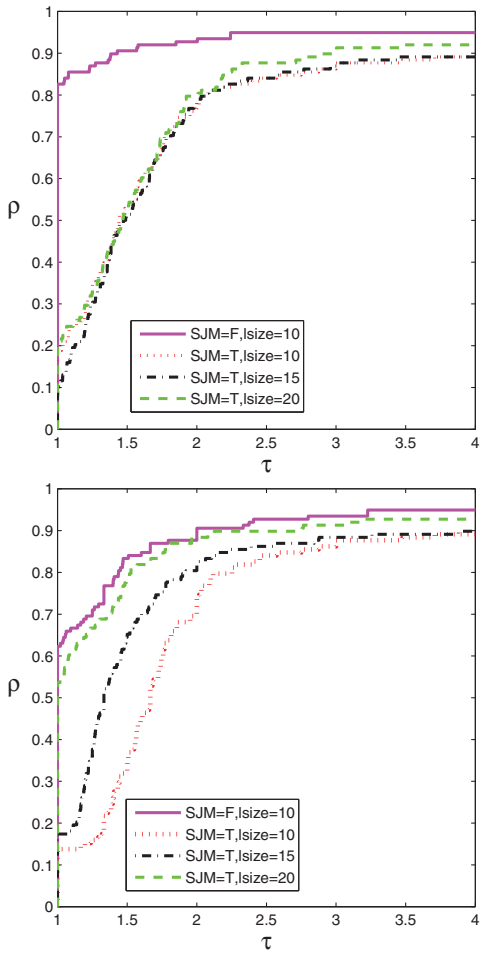

FIG. 2. *Efficiency (top) and iteration (bottom) performance profiles with (SJM = T) and without (SJM = F) Jennings–Malik modifications for* $rsize = 0$.

TABLE 1

A comparison of using a global diagonal shift (SJM = F) with the Jennings–Malik strategy (SJM = T) (rsize = 0, lsize = 10). The figures in parentheses are the number of diagonal shifts used and the final shift; times are in seconds.

Problem	Iterations		Factor time		Total time	
	F	T	F	T	F	T
HB/bcsstk28	232 (2, 2.0×10^{-3})	468	0.025	0.026	0.120	0.221
Cylshell/s3rmq4m1	648 (2, 2.0×10^{-3})	838	0.027	0.026	0.381	0.459
Rothberg/cfd2	550 (4, 3.2×10^{-2})	791	0.541	0.442	5.85	6.42
GHS_psdef/ldoor	434 (3, 8.0×10^{-3})	643	6.63	4.04	66.4	91.5
GHS_psdef/audikw_1	708 (2, 2.0×10^{-3})	1442	11.3	8.66	157	303

required for convergence, the time to compute the preconditioner, and the total time. (The reported times includes the time taken to restart the factorization following breakdown.) In each case, using a diagonal shift leads to a reduction in the iteration count, and this can be by more than a factor of two. This, in turn, reduces the time for the conjugate gradient algorithm and this reduction generally more than offsets the additional time taken for the factorization as a result of restarting. We remark, however, that Benzi [5] reports that for some highly sparse matrices (where few modifications are necessary) Jennings–Malik modifications can work well (see also [6]) and this is fully consistent with Theorem 3.3 and the comments following its proof.

We now consider a dropping-based Jennings–Malik strategy in which off-diagonal entries that are less than a chosen tolerance *droptol*1 in absolute value are dropped from L and added to the corresponding diagonal entries. The largest (in absolute value) entries in each column j (up to $lsize + n_j$ entries) are retained in the computed factor. Figure 3 presents an efficiency performance profile for a range of values of *droptol*1. We include *droptol*1= 0 (no dropping and no modifications). Although not given here, the total time performance profile is very similar. For these experiments, we use $lsize = 10$. We see that as *droptol*1 increases, the efficiency steadily deteriorates and the robustness of the preconditioner decreases. In Figure 4, we compare dropping small entries without modifications (denoted by JM = F) with using Jennings–Malik modifications (JM = T). It is clear that in terms of efficiency, it is better not to use the Jennings–Malik modifications. However, an advantage of the latter is that taken over the complete test set, it reduces the number of breakdowns and subsequent restarts. Furthermore, the computed incomplete factor is generally sparser when Jennings–Malik modifications are used, potentially reducing its application time.

5.4. Results for *rsize* varying, without Jennings–Malik modifications. We have seen that increasing $lsize$ with $rsize = 0$ does little to improve the efficiency of the preconditioner. We now consider fixing the incomplete factor size ($lsize = 5$) and varying the amount of intermediate memory (controlled by $rsize$). We run with no intermediate memory, $rsize = 2, 5$, and 10, and with unlimited intermediate memory. (All entries in R are retained, which we denote by $rsize = -1$.) Note that the latter is the original Tismenetsky approach with the memory limit $lsize$ used to determine L. Figure 5 presents the efficiency (top left), time to compute the preconditioner (top right), and total time (bottom left) performance profiles. Since $lsize$ is the same for all runs, the fill in L is essentially the same in each case and thus comparing the efficiency here is equivalent to comparing the iteration counts. For many of our test

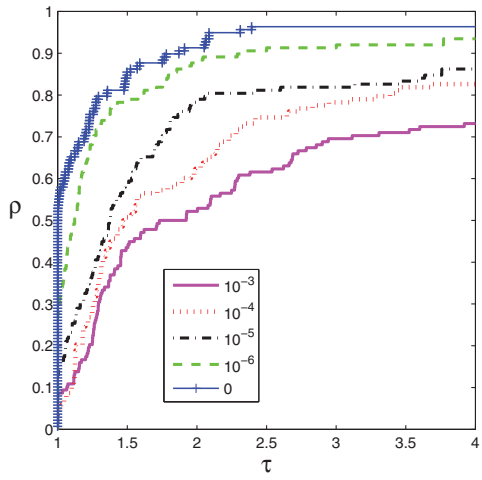

FIG. 3. *Efficiency performance profile for the Jennings–Malik strategy based on a drop tolerance for* $rsize = 0$ *and a range values of the drop tolerance droptol*1.

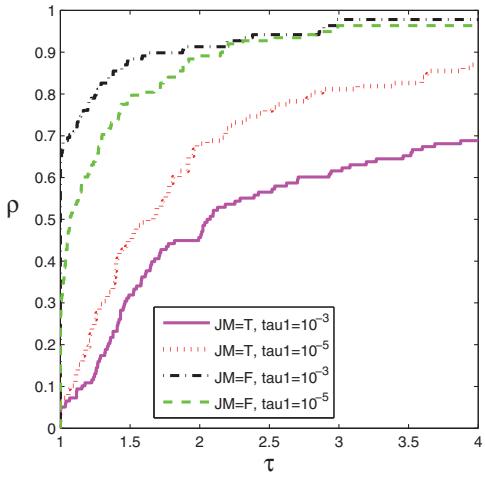

FIG. 4. *Efficiency performance profile with (JM = T) and without (JM = F) the Jennings–Malik strategy based on a drop tolerance. Here* $rsize = 0$.

problems, we see that the Tismenetsky approach ($rsize = -1$) gives the most efficient preconditioner but it is also expensive to compute and we were not able to factorize a number of our largest problems in this case because of insufficient memory, that is, a memory allocation error was returned before the factorization was complete; this is reflected by poor reliability and, as anticipated, makes the original Tismenetsky approach impractical for large problems.

We see that, as $rsize$ is increased from 0 to 10, the efficiency and robustness of the preconditioner steadily increases (along with the time to compute it), but without significantly increasing the total time. Since a larger value of $rsize$ reduces the number of iterations required, if more than one problem is to be solved with the same preconditioner, it may be worthwhile to increase $rsize$ in this case (but the precise choice of $rsize$ is not important).

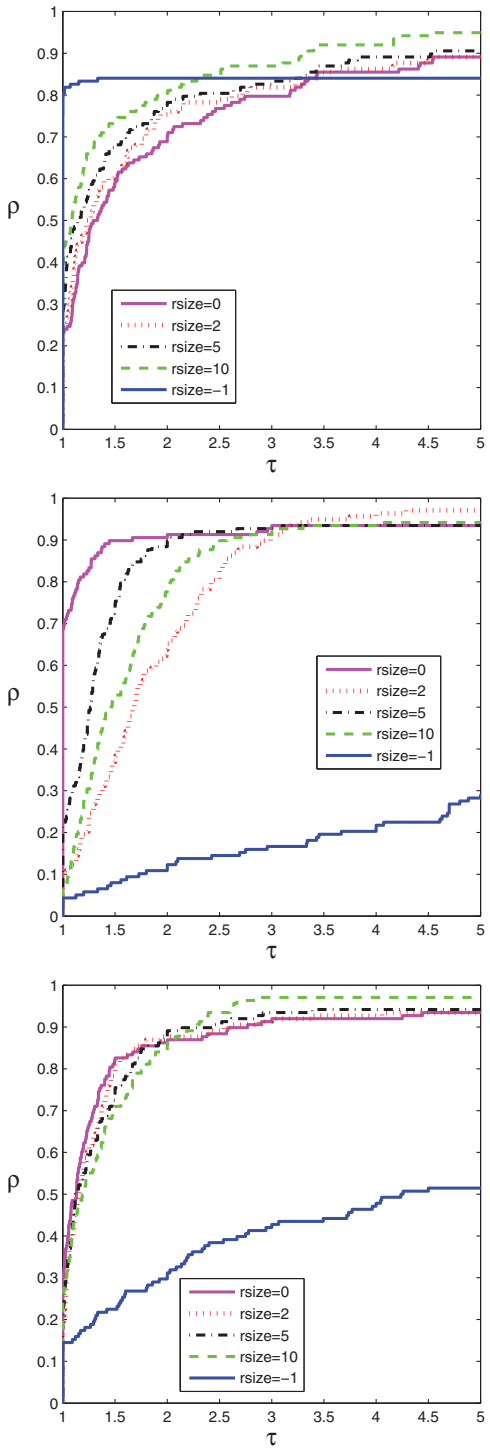

FIG. 5. *Efficiency (top), time to compute the preconditioner (middle), and total time (bottom) performance profiles for rsize varying.*

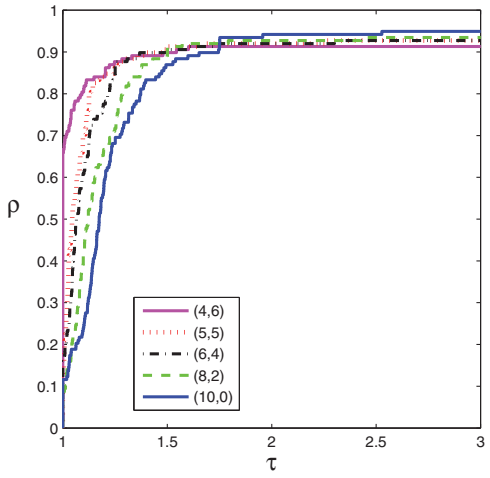

FIG. 6. *Efficiency performance profile for different pairs (lsize, rsize) with* $lsize + rsize = 10$.

5.5. Results for $lsize+rsize$ constant, without Jennings–Malik modifications. We present the efficiency performance profile for $tsize = lsize + rsize = 10$ in Figure 6. We see that increasing $rsize$ at the expense of $lsize$ can improve efficiency. This is because the computed L is sparser for smaller $lsize$, while the use of R helps maintains the quality of the preconditioner.

It is of interest to compare Figure 6 with Figure 1 (in the latter, $lsize$ varies but $rsize = 0$); this clearly highlights the effects of using R. In terms of time, increasing $rsize$ while decreasing $lsize$ keeps the time for computing the incomplete factorization essentially the same, while the cost of each application of the preconditioner reduces, but as the number of iterations increases, we found in our tests that the total time (in our serial implementation) for $tsize$ constant was not very sensitive to the split between $lsize$ and $rsize$.

5.6. Results for $rsize > 0$, with Jennings–Malik modifications. We now consider Jennings–Malik diagonal modifications used with `rsize` $>$ `0`. In our discussion, we refer to line numbers within Algorithm 4.1. We present results for three strategies for dealing with the entries of RR^T, denoted by `jm` $=$ `0`, `1`, and `2`. These strategies can be also considered as a new development motivated by Theorem 3.3. When computing column j, we gather updates to column j of L and column j of R from the previous $j-1$ columns of L and from the previous $j-1$ columns of R according to the formula $LL^T + RL^T + LR^T$. We then consider gathering updates to column j of R from the previous $j-1$ columns of R (RR^T). We will distinguish three different cases. With `jm` $=$ `0`, we allow entries of RR^T that cause no further fill in $LL^T + RL^T + LR^T$ and discard all other entries of RR^T. In this case, we do not use the Jennings–Malik modification on the lines 19–20 (that is, we delete the lines $A_{j,j} = A_{j,j} + |R_{i,k} * R_{j,k}|$ and $A_{i,i} = A_{i,i} + |R_{i,k} * R_{j,k}|$). This decreases the norm of the modification. With `jm` $=$ `1`, we use Jennings–Malik modification for these discarded entries (that is, lines 19–20 are used). Finally, with `jm` $=$ `2`, we discard all entries of RR^T; this is our limited memory variant of the Tismenetsky approach (it deletes lines 14–22). An efficiency performance profile is presented in Figure 7 for `lsize` $=$ `rsize` $=$ `5` and `10`. Note that here we do not apply Jennings–Malik modifications to the entries that are discarded from column j of R before it is stored

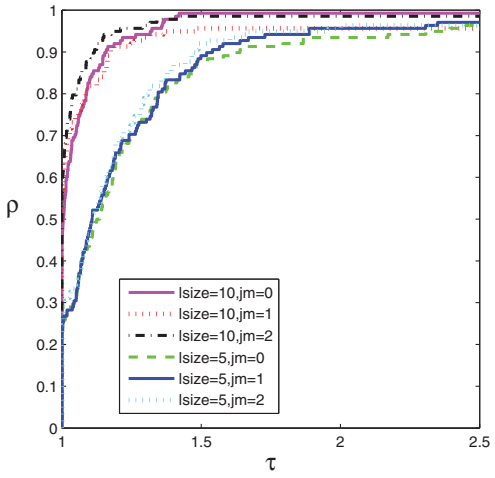

FIG. 7. *Efficiency performance profile for* $lsize = rsize = 5$ *and* 10 *with* `jm` $=$ `0, 1, 2`.

and used in computing the remaining columns of L and R. (Only the `rsize` largest entries are retained in each column of R.) This corresponds to deleting lines 30–35. We see that, considering the whole test set, there is generally little to choose between the three approaches. We also note the very high level of reliability when allowing only a modest number of entries in L. (For `lsize` $=$ `10`, the only problem that was not solved with `jm` $=$ `0` and `jm` $=$ `2` was Oberwolfach/boneS10.)

We also want to determine whether Jennings–Malik modifications for the entries that are discarded from R is beneficial. In Figure 8, we compare running with and without Jennings–Malik modifications (that is, with and without lines 30–35). We can clearly see that in terms of efficiency, time, and reliability, using modifications for the dropped entries is not, in general, beneficial.

In Table 2, we present some more detailed results with and without Jennings–Malik modification for the discarded entries. If Jennings–Malik modification is not used, the problems in the top part of the table require diagonal shifts to prevent breakdown. With Jennings–Malik modification, there is no guarantee that a shift will not be required (examples are Cylshell/s3rmt3m3 and DNVS/shipsec8) but fewer problems require a shift. (In our test set, with and without the use of Jennings–Malik modification the number of problems that require a shift with the chosen parameter settings is 16 and 59, respectively.) From our tests, we see that using diagonal shifts generally results in a higher-quality preconditioner than using Jennings–Malik modification and, even allowing for the extra time needed when the factorization is restarted, the former generally leads to a smaller total time. Note that for problem Janna/Serena, using a diagonal shift leads to a higher-quality preconditioner but, as four shifts are used, the factorization time dominates and leads to the total time being greater than for Jennings–Malik modifications. In this case, the initial nonzero shift $\alpha_1 = 0.001$ leads to a breakdown-free factorization and, as we want to use as small a shift as possible, we decrease the shift (see [43] for full details). However, if we do not try to minimize the shift, the total time is reduced from 69.6 to 38.9 seconds. Clearly, how sensitive the results are to the choice of α is problem dependent.

For the problems that are breakdown free, the preconditioner is of better quality if Jennings–Malik modifications are not used: this is true for all our breakdown-free

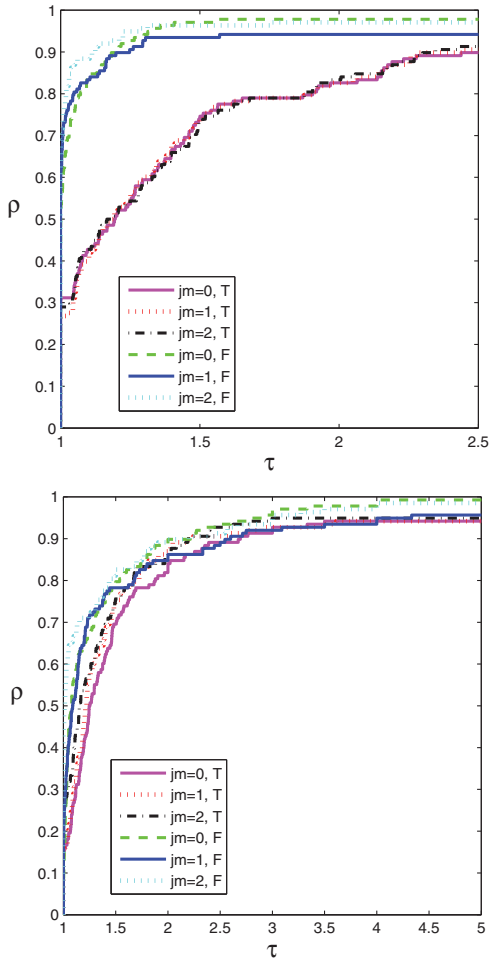

FIG. 8. *Efficiency (top) and total time (bottom) performance profiles with (T) and without (F) Jennings–Malik modifications for the discarded entries ($lsize = rsize = 10$).*

TABLE 2

A comparison of using (T) and not using (F) Jennings–Malik modifications for the discarded entries (`jm` $= 2$, $lsize = rsize = 10$*). The figures in parentheses are the number of diagonal shifts used and the final shift; times are in seconds.*

Problem	Iterations		Factor time		Total time	
	T	F	T	F	T	F
FIDAP/ex15	503 (0, 0.0)	322 (3, 1.60×10^{-2})	0.022	0.033	0.184	0.135
Cylshell/s3rmt3m3	1005 (3, 2.50×10^{-4})	615 (2, 2.00×10^{-3})	0.066	0.034	0.495	0.299
Janna/Fault_639	300 (0, 0.0)	122 (3, 2.50×10^{-4})	6.03	8.74	31.6	18.5
ND/nd24k	407 (0, 0.0)	173 (3, 2.50×10^{-4})	2.56	3.71	14.2	8.59
DNVS/shipsec8	956 (4, 9.76×10^{-7})	648 (3, 2.50×10^{-4})	3.53	1.43	17.2	10.3
GHS_psdef/audikw_1	1447 (0, 0.0)	517 (2, 2.00×10^{-3})	14.1	21.1	335	106
Janna/Serena	165 (0, 0.0)	122 (4, 9.76×10^{-7})	12.7	44.9	44.9	69.6
HB/bcsstk24	133 (4, 9.76×10^{-7})	344 (3, 1.00×10^{-3})	0.093	0.037	0.133	0.142
GHS_psdef/crankseg_2	251 (0, 0.0)	49 (0, 0.0)	2.46	2.17	10.1	3.64
Schenk/AF-shell7	389 (0, 0.0)	325 (0, 0.0)	2.21	1.98	23.5	20.1
Oberwolfach/bone010	1912 (0, 0.0)	1481 (0, 0.0)	12.6	10.1	382	281
Janna/Emilia_923	147 (0, 0.0)	101 (0, 0.0)	7.09	5.45	24.3	16.7

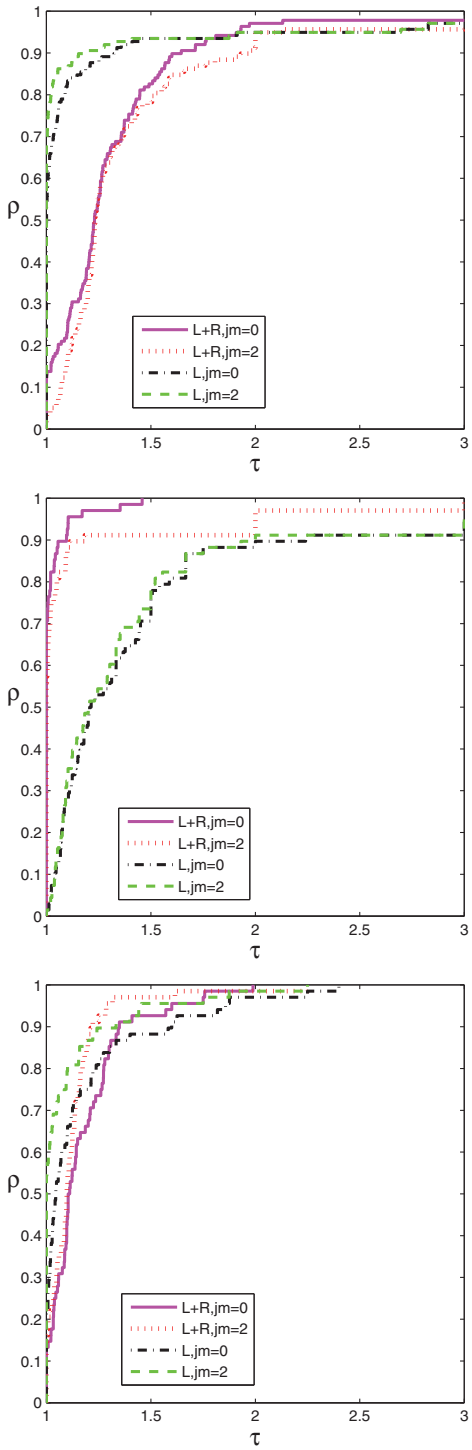

FIG. 9. *Efficiency (top), iteration (middle), and total time (bottom) performance profiles for* $L + R$ *and* L *with* `jm` $=$ `0` *and* `2`.

test problems. Our experiments demonstrate that using modifications can lead to a substantial increase in the iteration count (see, for example, GHS_psdef/crankseg_2). Moreover, when no restarts are required, the use of modifications increases the factorization time.

5.7. The use of $L + R$. So far, we have used R in the computation of L, but once the incomplete factorization has finished, we have discarded R and used L as the preconditioner. We now consider using $L + R$ as the preconditioner. In Figure 9, we present performance profiles for $L + R$ and L with `jm` = `0` and `2`. Here $lsize = rsize = 10$, $droptol1 = 0.001$, $droptol2 = 0.0$. We see that in terms of efficiency, using L is better than using $L + R$. This is because the number of entries in L is much less than in $L + R$. However, $L + R$ is a higher-quality preconditioner, requiring fewer iterations for convergence (with `jm` = `0` giving the best results). In terms of total time, there is little to choose between using $L + R$ and L.

6. Conclusions. In this paper, we have focused on the use of positive semidefinite modification schemes for computing IC factorization preconditioners. We have studied in particular the methods proposed originally by Jennings and Malik and by Tismenetsky and we have presented new theoretical results that aim to achieve a better understanding of the relationship between them. To make the robust but memory-expensive approach of Tismenetsky into a practical algorithm for large-scale problems, we have incorporated the use of limited memory.

A major contribution of this paper is the inclusion of extensive numerical experiments. We are persuaded that using a large test set drawn from a range of practical applications allows us to make general and authoritative conclusions. The experiments emphasize the concept of the efficiency of a preconditioner (defined by (5.2)) as a measure that can be employed to help capture preconditioner usefulness, although we recognize that it can also be necessary to consider other statistics (such as the time and the number of iterations).

Without the use of intermediate memory ($rsize = 0$), our results have shown that increasing the amount of fill allowed within a column of L (that is, increasing $lsize$) does not generally improve the efficiency of the preconditioner. The real improvement comes through following Tismenetsky and introducing intermediate memory. In our version, we prescribe the maximum number of entries in each column of both L and R; we also optionally allow small entries to be dropped to help further sparsify the preconditioner without significant loss of efficiency. Our results show that this leads to a highly robust yet sparse IC preconditioner. An interesting finding is that increasing $rsize$ at the expense of $lsize$ can result in a sparser preconditioner without lose of efficiency.

Our experiments have shown that using Jennings–Malik modifications to make the Tismenetsky approach breakdown free is less effective than employing global diagonal shifts. We obtained the same conclusion for a number of variations of the Jennings–Malik strategy. Provided we can catch zero or negative pivots and then restart the factorization process using a global diagonal shift, we can handle breakdowns. In our tests, this well-established approach was found to be the more efficient overall, that is, it produced a higher-quality preconditioner than using a Jennings–Malik scheme to modify diagonal entries. However, we must emphasize that this conclusion relies on having appropriately prescaled the matrix; if not, a large number of restarts can be required (adding to the computation time) and the diagonal shift needed to guarantee a breakdown-free factorization can be so large as to make the resulting IC preconditioner ineffective. Of course, the Jennings–Malik strategy can suffer from the same

type of drawback, namely, although the factorization is breakdown free, the resulting preconditioner may not be efficient (see [11]). Indeed, if many fill entries are dropped from the factorization, large diagonal modifications may be performed, reducing the accuracy of the preconditioner.

Finally, we note that we have developed a library-quality code `HSL_MI28` based on the findings of this paper. (See [43] for details and for numerical comparisons with other approaches.) This is a general-purpose IC factorization code and is available as part of the HSL mathematical software library [20]. The user specifies the maximum column counts for L and R (and thus the amount of memory to be used for the factorization) but, importantly for nonexperts, it is not necessary to perform a lot of tuning since although the default settings of the control parameters will clearly not always be the best choices for a given class of problems, they have been chosen to give good results for a wide range of problems. Of course, a more experienced user may choose to perform experiments and then to reset the controls. This "black-box" approach is in contrast to the Tismenetsky–Kaporin-related work reported, for example, by Yamazaki et al. [54], where by restricting attention to a specific class of problems, it is possible to determine an interval of useful drop tolerances that limit the size of the computed factor.

Acknowledgments. We are grateful to three anonymous reviewers for their helpful and constructive feedback. We would also like to thank Igor Kaporin for sending us his comments on our manuscript. M. Tůma acknowledges travel support from the Academy of Sciences of the Czech Republic.

REFERENCES

[1] M. A. Ajiz and A. Jennings, *A robust incomplete Choleski-conjugate gradient algorithm*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 949–966.

[2] O. Axelsson, I. Kaporin, I. Konshin, A. Kucherov, M. Neytcheva, B. Polman, and A. Yeremin, *Comparison of Algebraic Solution Methods on a Set of Benchmark Problems in Linear Elasticity*, STW project NNS.4683, Department of Mathematics, University of Nijmegen, the Netherlands, 2000.

[3] L. Beirão da Veiga, V. Gyrya, K. Lipnikov, and G. Manzini, *Mimetic finite difference method for the Stokes problem on polygonal meshes*, J. Comput. Phys., 228 (2009), pp. 7215–7232.

[4] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[5] M. Benzi, *Private Communication*, 2013.

[6] M. Benzi, R. Kouhia, and M. Tůma, *An assessment of some preconditioning techniques in shell problems*, Commun. Numer. Methods Eng., 14 (1998), pp. 897–906.

[7] M. Benzi and M. Tůma, *A robust incomplete factorization preconditioner for positive definite matrices*, Numer. Linear Algebra Appl., 10 (2003), pp. 385–400.

[8] M. Bern, J. R. Gilbert, B. Hendrickson, N. T. Nguyen, and S. Toledo, *Support-graph preconditioners*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 930–951.

[9] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.

[10] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011).

[11] J. K. Dickinson and P. A. Forsyth, *Preconditioned conjugate gradient methods for three-dimensional linear elasticity*, Internat. J. Numer. Methods Engrg., 37 (1994), pp. 2211–2234.

[12] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.

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

[14] V. Eijkhout, *The 'Weighted Modification' Incomplete Factorisation Method*, http://www.netlib.org/lapack/lawns/ (1999).

[15] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *The Yale Sparse Matrix Package (YSMP)–*II*: The non-symmetric codes*, Tech. report 114, Department of Computer Science, Yale University, 1977.

[16] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *The Yale Sparse Matrix Package (YSMP)–*I*: The symmetric codes*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1145–1151.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, 1996.

[18] A. Gupta and T. George, *Adaptive techniques for improving the performance of incomplete factorization preconditioning*, SIAM J. Sci. Comput., 32 (2010), pp. 84–110.

[19] I. Hladík, M. B. Reed, and G. Swoboda, *Robust preconditioners for linear elasticity FEM analyses*, Internat. J. Numer. Methods Engrg., 40 (1997), pp. 2109–2127.

[20] *HSL. A Collection of Fortran Codes for Large-Scale Scientific Computation*, http://www.hsl.rl.ac.uk (2013).

[21] C. Janna, M. Ferronato, and G. Gambolati, *Multilevel incomplete factorizations for the iterative solution of non-linear FE problems*, Internat. J. Numer. Methods Engrg., 80 (2009), pp. 651–670.

[22] A. Jennings and M. A. Ajiz, *Incomplete methods for solving* $A^TAx = b$, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 978–987.

[23] A. Jennings and G. M. Malik, *Partial elimination*, J. Inst. Math. Appl., 20 (1977), pp. 307–316.

[24] A. Jennings and G. M. Malik, *The solution of sparse linear equations by the conjugate gradient method*, Internat. J. Numer. Methods Engrg., 12 (1978), pp. 141–158.

[25] M. T. Jones and P. E. Plassmann, *Algorithm* 740*: Fortran subroutines to compute improved incomplete Cholesky factorizations*, ACM Trans. Math. Softw., 21 (1995), pp. 18–19.

[26] M. T. Jones and P. E. Plassmann, *An improved incomplete Cholesky factorization*, ACM Trans. Math. Softw., 21 (1995), pp. 5–17.

[27] I. E. Kaporin, *High quality preconditioning of a general symmetric positive definite matrix based on its* $U^TU + U^TR + R^TU$ *decomposition*, Numer. Linear Algebra Appl., 5 (1998), pp. 483–509.

[28] I. E. Kaporin, *Using the modified* 2*nd order incomplete Cholesky decomposition as the conjugate gradient preconditioning*, Numer. Linear Algebra Appl., 9 (2002), pp. 401–408.

[29] D. S. Kershaw, *The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations*, J. Comput. Phys., 26 (1978), pp. 43–65.

[30] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Comput., 21 (1999), pp. 24–45.

[31] K. Lipnikov, M. Shashkov, D. Svyatskiy, and Y. Vassilevski, *Monotone finite volume schemes for diffusion equations on unstructured triangular and shape-regular polygonal meshes*, J. Comput. Phys., 227 (2007), pp. 492–512.

[32] K. Lipnikov, D. Svyatskiy, and Y. Vassilevski, *Interpolation-free monotone finite volume method for diffusion equations on polygonal meshes*, J. Comput. Phys., 228 (2009), pp. 703–716.

[33] S. MacLachlan, D. Osei-Kuffuor, and Y. Saad, *Modification and compensation strategies for threshold-based incomplete factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A48–A75.

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

[35] 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. Comput., 31 (1977), pp. 148–162.

[36] N. Munksgaard, *New factorization codes for sparse, symmetric and positive definite matrices*, BIT, 19 (1979), pp. 43–52.

[37] N. Munksgaard, *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Softw., 6 (1980), pp. 206–219.

[38] J. M. Ortega, *Introduction to Parallel and Vector Computing*, Plenum Press, New York, 1988.

[39] M. Papadrakakis and M. C. Dracopoulos, *Improving the efficiency of incomplete Choleski preconditionings*, Commun. Appl. Numer. Methods, 7 (1991), pp. 603–612.

[40] J. K. Reid and J. A. Scott, *Ordering symmetric sparse matrices for small profile and wavefront*, Internat. J. Numer. Methods Engrg., 45 (1999), pp. 1737–1755.

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

[42] J. A. Scott and M. Tůma, *The importance of structure in incomplete factorization preconditioners*, BIT, 51 (2011), pp. 385–404.
[43] J. A. Scott and M. Tůma, `HSL_MI28`*: An Efficient and Robust Limited Memory Incomplete Cholesky Factorization Code*, Tech. report RAL-TR-2013-P-004, Rutherford Appleton Laboratory, 2013.
[44] J. A. Scott and M. Tůma, *On Positive Semidefinite Modification Schemes for Incomplete Cholesky Factorization*, Tech. report RAL-TR-2013-P-005, Rutherford Appleton Laboratory, 2013.
[45] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, Internat. J. Numer. Methods Engrg., 23 (1986), pp. 239–251.
[46] S. W. Sloan, *A Fortran program for profile and wavefront reduction*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 2651–2679.
[47] M. Suarjana and K. H. Law, *A robust incomplete factorization based on value and space constraints*, Internat. J. Numer. Methods Engrg., 38 (1995), pp. 1703–1719.
[48] M. Tismenetsky, *A new preconditioning technique for solving large sparse linear systems*, Linear Algebra Appl., 154–156 (1991), pp. 331–353.
[49] R. S. Varga, E. B. Saff, and V. Mehrmann, *Incomplete factorizations of matrices and connections with h-matrices*, SIAM J. Numer. Anal., 17 (1980), pp. 787–793.
[50] X. Wang, *Incomplete Factorization Preconditioning for Linear Least Squares Problems*, Ph.D. thesis, Department of Computer Science, University of Illinois Urbana-Champaign, 1993.
[51] X. Wang, R. Bramley, and K. A. Gallivan, *A Necessary and Sufficient Symbolic Condition for the Existence of Incomplete Cholesky Factorization*, Tech. report TR440, Department of Computer Science, Indiana University, Bloomington, 1995.
[52] X. Wang, K. A. Gallivan, and R. Bramley, *Incomplete Cholesky Factorization with Sparsity Pattern Modification*, Tech. report, Department of Computer Science, Indiana University, Bloomington, 1993.
[53] X. Wang, K. A. Gallivan, and R. Bramley, *CIMGS: An incomplete orthogonal factorization preconditioner*, SIAM J. Sci. Comput., 18 (1997), pp. 516–536.
[54] I. Yamazaki, Z. Bai, W. Chen, and R. Scalettar, *A high-quality preconditioning technique for multi-length-scale symmetric positive definite linear systems*, Numer. Math. Theory Methods Appl., 2 (2009), pp. 469–484.