SIAM J. SCI. COMPUT.
Vol. 36, No. 6, pp. A2984–A3010

ON SIGNED INCOMPLETE CHOLESKY FACTORIZATION PRECONDITIONERS FOR SADDLE-POINT SYSTEMS*

JENNIFER SCOTT† AND MIROSLAV TŮMA‡

Abstract. Limited-memory incomplete Cholesky factorizations can provide robust preconditioners for sparse symmetric positive-definite linear systems. In this paper, the focus is on extending the approach to sparse symmetric indefinite systems in saddle-point form. A limited-memory signed incomplete Cholesky factorization of the form LDL^T is proposed, where the diagonal matrix D has entries ± 1. The main advantage of this approach is its simplicity as it avoids the use of numerical pivoting. Instead, a global shift strategy involving two shifts (one for the $(1,1)$ block and one for the $(2,2)$ block of the saddle-point matrix) is used to prevent breakdown and to improve performance. The matrix is optionally prescaled and preordered using a standard sparse matrix ordering scheme that is then postprocessed to give a constrained ordering that reduces the likelihood of breakdown and need for shifts. The use of intermediate memory (memory used in the construction of the incomplete factorization but subsequently discarded) is shown to significantly improve the performance of the resulting preconditioner. Some new theoretical results are presented, and for problems arising from a range of practical applications, numerical results are given to illustrate the effectiveness of the signed incomplete Cholesky factorization as a preconditioner. Comparisons are made with a recent incomplete LDL^T code that employs pivoting.

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

AMS subject classifications. 65F05, 65F50

DOI. 10.1137/140956671

1. Introduction.

We are interested in solving linear systems $Kx = b$ where the sparse symmetric matrix K is of the form

$$K = \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix}. \tag{1.1}$$

Here A is $n \times n$ symmetric positive definite, B is rectangular $m \times n$ and of full rank $(m \le n)$, and C is $m \times m$ symmetric positive semidefinite. Matrices of the form (1.1) are often called *saddle-point* matrices or, in the special case $C = 0$, KKT matrices, in reference to the Karush–Kuhn–Tucker first-order necessary optimality conditions for the solution of general nonlinear programming problems. Saddle-point problems arise frequently in many applications in science and engineering, including in equality and inequality constrained nonlinear programming, interior point algorithms in both linear and nonlinear optimization, sparse optimal control, mixed finite-element discretizations of partial differential equations in fields such as fluid and solid mechanics, and circuit analysis.

*Submitted to the journal's Methods and Algorithms for Scientific Computing section February 11, 2014; accepted for publication (in revised form) September 8, 2014; published electronically December 23, 2014.

http://www.siam.org/journals/sisc/36-6/95667.html

†Scientific Computing, Rutherford Appleton Laboratory, Harwell Oxford, Oxfordshire, OX11 0QX, United Kingdom (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, 182 07 Praha 8, Czech Republic (tuma@cs.cas.cz). This author's work was partially supported by the Grant Agency of the Czech Republic, project 13-06684 S.

Typically, K is large. In some cases it may be possible as well as desirable to use a direct solver; indeed, a number of sparse direct solvers (including the packages `MA57` [17] and `HSL_MA86` [32] from the HSL mathematical software library [35]) have been designed with such systems in mind. However, the memory demands of direct solvers can mean that, for very large K, it is necessary to use an iterative method, usually a Krylov subspace-based method (see, for example, [64]). Moreover, in some applications it may not be necessary to solve the system with high accuracy or the linear solve may be part of a nonlinear iteration and here an iterative method may then be the method of choice. Unfortunately, Krylov methods tend to converge very slowly when applied to saddle-point systems and a good preconditioner is needed to accelerate convergence. Over the last 20 years or so, a vast amount of work has been devoted to the development of effective preconditioners for saddle-point problems. A quick search of the literature reveals numerous publications incorporating a wide range of application areas; an excellent survey of the work done up to 2005, together with a comprehensive reference list, is given in section 10 of [4] (see also [5] for a later and more concise overview).

As discussed in [4, 5], much effort has focused on the development of block diagonal and block triangular preconditioners as well as on constraint preconditioners. Comparatively little work appears to have concentrated on the development of reliable incomplete factorization techniques for saddle-point systems. Following work done on generating good orderings and scalings for direct solvers for indefinite systems [20, 55], Hagemann and Schenk [30] proposed using a maximum weighted matching algorithm to preprocess the matrix. The objective is to permute large entries on to the subdiagonal that can then be used to form stable 2×2 pivots, allowing the incomplete factorization to avoid the use of dynamic pivoting (which can add a computational overhead as well as significantly complicating the software development). If a pivot is found to be too small, breakdown is prevented by adding a small perturbation. In [30], some good results are reported (sparse factors and lower iteration counts) for tests on saddle-point problems.

Li and Saad [38] developed pivoting strategies for sparse symmetric matrices to improve the robustness of their Crout variant of the ILU preconditioner [39]. They found that incorporating Bunch–Kaufman pivoting [10] can be efficiently and effectively integrated within an incomplete factorization for sparse symmetric indefinite matrices and reported some encouraging results for general indefinite problems as well as for some KKT problems. Using the work of Li and Saad, Greif, He, and Liu [29] have recently made available an incomplete factorization package called SYM-ILDL that is designed for general symmetric indefinite matrices. This package optionally starts by scaling the matrix to be equilibrated in the maximum norm, and then preorders using the reverse Cuthill–McKee algorithm or approximate minimum degree. As in Li and Saad, there is an option to use Bunch–Kaufman pivoting during the factorization to maintain stability and avoid breakdown. The user controls the maximum allowed fill within each column and also the dropping of small entries. Currently, no published results are available for this new code but initial findings appear consistent with those reported by Li and Saad [28] (see also section 6 below).

For saddle-point systems, an alternative approach is based on the observation that K can be factored into the form

$$K = \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix}, \tag{1.2}$$

where

$$A = L_{11}L_{11}^T$$

is the Cholesky factorization of A,

$$L_{21} = BL_{11}^{-T},$$

and

$$S = C + L_{21}L_{21}^T = L_{22}L_{22}^T$$

is the Cholesky factorization of the (negative) Schur complement [49, 69]. Note that this factorization always exists, without pivoting (although numerical stability is not guaranteed). Bridson [9] refers to (1.2) as a *signed Cholesky factorization*. If we set

$$\mathcal{L} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix},$$

then $P = \mathcal{L}\mathcal{L}^T$ can be used as a (split) preconditioner. Since the preconditioned matrix $\mathcal{L}^{-1}K\mathcal{L}^{-T}$ has two distinct eigenvalues ± 1, a symmetric Krylov subspace method such as MINRES or SYMMLQ [47] would converge in at most two iterations. In practice, the exact factor $\mathcal{L}$ is replaced by an incomplete one. This can be achieved by first computing an incomplete Cholesky (IC) factorization

$$A \approx \tilde{L}_{11}\tilde{L}_{11}^T,$$

which can be done using one of the many available approaches (see, for example, [59] for a historical overview and a list of references to work done since the 1950s). The second step is to compute a sparse approximation $\tilde{L}_{21}$ to L_{21} by solving the lower triangular system

$$\tilde{L}_{11}\tilde{L}_{21}^T = B^T \tag{1.3}$$

and then possibly applying some dropping criteria to preserve sparsity in $\tilde{L}_{21}$. Finally, an IC factorization of the approximate Schur complement

$$C + \tilde{L}_{21}\tilde{L}_{21}^T \approx \tilde{L}_{22}\tilde{L}_{22}^T \tag{1.4}$$

is computed. The resulting incomplete factor

$$\tilde{\mathcal{L}} = \begin{pmatrix} \tilde{L}_{11} & 0 \\ \tilde{L}_{21} & \tilde{L}_{22} \end{pmatrix}$$

and its transpose can then be used to define a positive-definite factorized preconditioner $\tilde{P} = \tilde{\mathcal{L}}\tilde{\mathcal{L}}^T$. Some very limited numerical results using the SYMMLQ and Uzawa [23] algorithms are given in [49]; a more comprehensive study of how this approach performs in practice (and, in particular, how to obtain a good sparse approximation $\tilde{L}_{21}$ to L_{21}) appears to be currently lacking. This is something we plan to report on in the future, using our IC factorization code `HSL_MI28` [58] to compute each of the two IC factorizations.

In this paper, we take a different approach to obtain an incomplete factorization. Our idea is to compute a signed IC factorization. The signed complete Cholesky factorization approach has been used for saddle-point systems by, for example, Bridson [9]

(see also [56]). A signed IC approach is attractive for us as it allows us to use a modified version of our existing IC factorization code and, at the same time, permits the exploitation of more general orderings that are not restricted to ordering the $(1,1)$ block of the matrix separately from the $(2,2)$ block. Allowing the use of more general permutations appears to be a preferable strategy when factorizing KKT systems, as discussed and demonstrated, for example, by Fourer and Mehrotra in their paper [24] on solving the indefinite linear systems that arise in an interior-point method. Note that the signed factorization avoids the need to compute an explicit sparse approximation of $\tilde{L}_{21}$ as this block is part of the global Cholesky-like factor. Importantly, we also avoid the need for pivoting and the use of 2×2 pivots (but see the comments on the stability of this approach below). Performing threshold partial pivoting at each step of the factorization process not only complicates the implementation but can add a significant time overhead. See also the related motivation for static pivoting introduced for LU factorizations in [40].

Before describing our approach in more detail, we briefly mention similar concepts for obtaining an unreduced Cholesky-like factorization, that is, the factorization algorithm is applied to the whole matrix K, not preceded by the null-space or range-space transformations of the matrix blocks used in segregated approaches [4]. If the full rank condition for the B block in (1.1) is replaced by positive definiteness of the C block we get a class of symmetric quasi-definite (SQD) matrices. Vanderbei [65] shows that SQD matrices are strongly factorizable, i.e., a signed Cholesky factorization exists for such matrices under any symmetric permutation; this is a stronger result than we have for our saddle-point matrix K. Further, a stability analysis related to the factorization of SQD matrices is given by Gill, Saunders, and Shinnerl [25] (see also [26]), which shows the importance of the effective condition number of K for the stability of the factorization. In particular, the effective condition number is small if the maximum of the norms $||B^T A^{-1} B||$ and $||B^T C^{-1} B||$ is small with respect to the norm of K. Also note that the analysis of a symmetric indefinite factorization based on the related generalized QR factorization in [50] points out that conditioning of the principal leading submatrices may be determined by other factors such as the number of sign changes in the diagonal of the signed factorization. In practice, neither the SQD condition nor the full rank condition for the block B needs to be satisfied. Instead, the existence of a signed Cholesky factorization can be forced by "regularization," that is, by modifying the diagonal entries. Saunders and Tomlin [54] used fixed regularization parameters (diagonal shifts) for the blocks A and $-C$ to ensure stability a priori. After regularization of both blocks, the signed Cholesky factorization always exists and they used this approach to successfully solve test problems from the Netlib collection (http://www.netlib.org/lp/data/) via the barrier method using a sparse direct Cholesky solver. Dynamic regularization by perturbing diagonal entries in the signed Cholesky factorization that results in a SQD matrix was proposed by Altman and Gondzio [2]. This strategy seems to introduce less perturbation to the logarithmic barrier method used to solve optimization problems: see also numerical comparisons and notes on signed Cholesky and indefinite factorizations in [1, 6, 8]. While Altman and Gondzio use a complete signed Cholesky factorization within the HOPDM optimization code [27], the diagonal entry modification in case of a wrong sign (the modification should enforce positive definiteness of A and negative definiteness of $-C$) or an entry of small magnitude is based on that originally proposed by Kershaw [37]. Another regularization strategy combined with reorderings restricted to blocks (priority minimum degree) of the saddle-point problem was proposed by Vanderbei and Shanno [67] and implemented in the optimization code LOQO [66].

They shift the diagonal of the block corresponding to the primal variables by adding a multiple of a unit matrix. Moreover, in order to get a strong (complete) factorization of the regularized matrix, they allow both increases and decreases to the shifts in a manner that is similar to the strategy proposed by Manteuffel [42] and as implemented, for example, by Lin and Moré [41]; see also [58]. Thus the signed Cholesky factorization has attracted attention because its simplicity and because the systems to be solved, often obtained in sequence, may need regularization. More sophisticated and expensive dynamic reorderings that combine symbolical and numerical phases may then be considered as overkill. Note that the complete signed Cholesky implementation was also popular in solving indefinite systems within shift-and-invert based eigensolvers, especially in engineering communities. An example is the implementation of SKYPACK [43] for the block Lanczos eigenvalue software BLZPACK (http://crd-legacy.lbl.gov/~osni/#Software), where part of the motivation to use a signed Cholesky factorization comes from the choice of skyline data structure for the sparse matrices.

Let us summarize the reasons leading us to develop a state-of-the-art approach to solve a large class of indefinite systems. First, as mentioned, we are interested in preconditioned iterative methods in order to be able to solve very large problems. Second, we want to exploit orderings that are as general as possible since general orderings are often claimed to lead to the most efficient solvers. Furthermore, an important number of real-world problems need to be regularized and algebraic modifications seem to be very natural. Our previous project [58, 59] considered a large spectrum of different application areas leading to linear systems with symmetric positive-definite system matrices and found that regularization by flexible strategies based on the Manteuffel shifts [42] is preferable to ad hoc modifications. We are keen to use such a strategy in a signed IC code. Last, but not least, we want preconditioners based on incomplete factorizations that are not only time and memory efficient but also robust.

The rest of the paper is organized as follows. In section 2, we briefly recall the signed Cholesky factorization approach using the description given by Bridson. Then in section 3, we summarize our limited memory IC factorization for positive-definite problems. This is extended to a signed IC factorization in section 4. In section 5, some theoretical results for the signed IC approach are given. Numerical results, including comparisons with SYM-ILDL, are presented in section 6, and concluding remarks are made in section 7.

2. Constrained ordering and signed IC factorizations. In this section, we consider the case of a complete factorization of the matrix (1.1). The aim of a constrained ordering is to find a permutation Q such that QKQ^T can be factorized stably without the need for numerical pivoting and without modifying the entries in K, while still limiting the number of entries in the factor. This problem has been examined for special classes of matrices by a number of authors. Of practical interest is the class of $\mathcal{F}$ matrices, where each column of B has exactly two entries which sum to zero and $C = 0$. These arise in, for example, Stokes flow problems. Tůma [62] and De Niet and Wubs [16] present methods for these problems, while Bridson [9] proposed a constrained ordering for more general saddle-point problems.

We use the terminology of Bridson and divide the nodes of the adjacency graph of the matrix K into two disjoint sets: those that correspond to the diagonal entries of A are known as A-nodes and the remaining nodes as C-nodes. The ordering constraint proposed in [9] is extremely simple: a C-node can be ordered only after all its A-node

neighbors in the graph of K have been ordered. Bridson has the following result, which is included here for completeness.

THEOREM 2.1. *Let A be positive definite, C be positive semidefinite, and B be of full row rank. Then if a C-node is ordered only after all its A-node neighbors, the signed Cholesky factorization*

$$PKP^T = LDL^T$$

exists, where P is a permutation matrix corresponding to the ordering of the nodes, L is a lower triangular matrix with positive diagonal entries, and D is a diagonal matrix with entries ± 1.

Proof. First observe that the inverse of a saddle-point matrix satisfying the above conditions can be expressed as

$$\begin{pmatrix} A & B^T \\ B & -C \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} - A^{-1}B^TS^{-1}BA^{-1} & A^{-1}B^TS^{-1} \\ S^{-1}BA^{-1} & -S^{-1} \end{pmatrix}, \tag{2.1}$$

where the negative Schur complement S given by

$$S = C + BA^{-1}B^T$$

is positive definite.

We now proceed by induction. The first node to be eliminated must be an A-node with pivot $d_1 = a_{11} > 0$. Assume $i-1$ steps of the factorization have been performed. Partitioning the first $i-1$ nodes into A-nodes and C-nodes, the ith principal submatrix of (a permutation of) K is

$$\begin{pmatrix} K_{i-1} & k_{i,:}^T \\ k_{i,:} & k_{ii} \end{pmatrix} = \begin{pmatrix} A_{i-1} & B_{i-1}^T & u^T \\ B_{i-1} & -C_{i-1} & v^T \\ u & v & k_{ii} \end{pmatrix}, \tag{2.2}$$

where $k_{i,:}$ denotes the first $i-1$ entries in row i. Since A_{i-1} is a principal submatrix of the positive-definite matrix A, it must also be positive definite. The ordering constraint implies that all the nonzeros in the rows of B corresponding to already ordered C-nodes must appear in B_{i-1} (otherwise there would be a C-node ordered before i with an A-node neighbor ordered at i or later). Since B is assumed to have full row rank, B_{i-1} also has full row rank: B_{i-1} is a subset of the rows of B with possibly some fully zero columns deleted. Thus K_{i-1} satisfies the same conditions as K.

Case 1: Node i is an A-node. It follows from the ordering constraint that i can have no previously ordered C-node neighbors and so $v = 0$ in (2.2). Using the form of the inverse in (2.1), the pivot is

$$\begin{aligned} d_{ii} &= k_{ii} - \begin{pmatrix} u & 0 \end{pmatrix} \begin{pmatrix} A_{i-1} & B_{i-1}^T \\ B_{i-1} & -C_{i-1} \end{pmatrix}^{-1} \begin{pmatrix} u^T \\ 0 \end{pmatrix} \\ &= k_{ii} - u(A_{i-1}^{-1} - A_{i-1}^{-1}B_{i-1}^TS_{i-1}^{-1}B_{i-1}A_{i-1}^{-1})u^T \\ &= (k_{ii} - uA_{i-1}^{-1}u^T) + uA_{i-1}^{-1}B_{i-1}^TS_{i-1}^{-1}B_{i-1}A_{i-1}^{-1}u^T. \end{aligned} \tag{2.3}$$

Note that

$$\begin{pmatrix} A_{i-1} & u^T \\ u & k_{ii} \end{pmatrix}$$

is a principal submatrix of the positive-definite matrix A and hence is positive definite. It follows that its final pivot $k_{ii} - uA_{i-1}^{-1}u^T$ is positive. Furthermore, since the negative Schur complement S_{i-1} is positive definite, the second term in (2.3) is also positive. Therefore, $d_{ii} > 0$.

Case 2: Node *i* is a *C*-node. In this case, we join the ith row and column to the other C-nodes in the partition (2.2). This $i \times i$ matrix also satisfies the rank condition and thus its inverse is of the form (2.1). The ith pivot is the reciprocal of the (i, i) entry of the $i \times i$ principal submatrix, which in this case comes from the diagonal of the negative definite matrix $-S_{i-1}$. Thus $d_{ii} < 0$.

As both cases give nonzero pivots, by induction the ordering constraint ensures the LDL^T factorization exists. Moreover, the pivots associated with the A-nodes are guaranteed to be positive and those associated with C-nodes are guaranteed to be negative. By rescaling, $L \leftarrow L|D|^{1/2}$ and $D \leftarrow sign(D) = \text{diag}(\pm 1)$, the diagonal matrix is fully determined in advance by the structure of the problem, independent of the numerical values. This gives a *signed Cholesky* factorization of K. □

The signed Cholesky factorization allows Bridson to modify a Cholesky factorization code to perform the factorization of the indefinite matrix K without numerical pivoting. A stability analysis is missing but Bridson reports that numerical experiments indicate, for well-scaled problems, the constrained ordering is generally sufficient to avoid numerical pivoting; this was supported by additional experiments given by Scott [56]. The hope is that if an initial ordering is chosen to reduce fill in L, the additional fill that results from modifying the ordering to a constrained ordering will be modest. If the constrained ordering is close to the initial ordering, the potential benefits include a fast factorization and, importantly, the analyse phase of the direct solver can accurately predict the size of the factors and other data structures required during the numerical factorization.

For a direct solver, choosing a fill-reducing ordering is essential (both for the memory needed and for the flop count). Bridson proposed two approaches to computing a constrained ordering. The first modifies the minimum degree algorithm (or one of its variants) to incorporate the constraint within it. An alternative approach is to postprocess a given fill-reducing ordering to satisfy the constraint. If a C-node is the next node in the supplied ordering, it is included in the modified ordering only once all its A-node neighbors have been ordered (that is, a C-node is postponed until after all its A-node neighbors). For many large problems, orderings based on nested dissection are frequently recommended in preference to those based on minimum degree. The advantages of the postprocessing approach are that it can be applied to any fill-reducing ordering and it is very cheap and straightforward to implement. Hogg and Scott [34, 56] considered this approach and compared using it to compute a signed Cholesky factorization (with no pivoting for stability) with using a fill-reducing ordering with a solver designed to factorize indefinite matrices with threshold pivoting incorporated for stability. Their reported results showed that the constrained ordering leads to significantly denser factors and higher flop counts so that it is unlikely to be competitive in practice for a direct solver.

The situation in the incomplete case is potentially somewhat different since, while it is known that the ordering used can be important (see, for example, the numerical results reported in [58] and the references therein), the choice of ordering is generally less crucial than in the complete factorization. This is because, in the incomplete case, the number of entries in the factor is determined not by the ordering but by user-defined parameters that may include the amount of fill allowed in each column

and drop tolerances. Thus our aim is to combine using a constrained ordering with a (modified) IC factorization code to compute a signed IC factorization that we can use with an iterative solver.

We remark that Rehman, Vuik, and Segal [63] propose an a priori ordering of the nodes in their development of incomplete factorization preconditioners for the (unsymmetric) saddle-point systems that arise from the finite-element discretization of incompressible Navier–Stokes equations. They employ a level-based band- or profile-reduction ordering, which they postprocess level by level so that, at each level, the A-nodes precede the C-nodes.

3. A limited-memory IC factorization. We now summarize the limited-memory IC factorization algorithm that is implemented within the HSL package `HSL_MI28`. We assume here that A is a symmetric and positive-definite matrix for which an IC factorization is required. For such A, `HSL_MI28` computes an IC factorization $(QL)(QL)^T$, where Q is a permutation matrix, chosen to preserve sparsity. The matrix A is optionally scaled and, if necessary, following the approach of Manteuffel [42], shifted to avoid breakdown of the factorization. Thus the incomplete factorization of $\bar{A} = \bar{S}Q^T AQ\bar{S} + \alpha I$ is computed, where $\bar{S} = \{\bar{s}_i\}$ is a diagonal scaling matrix and α is a positive shift. The user supplies the lower triangular part of A in compressed sparse column format and the computed L is returned to the user in the same format; a separate entry performs the preconditioning operation $y = Pz$, where $P = (\bar{L}\bar{L}^T)^{-1}$, $\bar{L} = Q\bar{S}^{-1}L$, is the incomplete factorization preconditioner.

The algorithm implemented by `HSL_MI28` is a limited memory approach by Tismenetsky and Kaporin. The basic scheme is based on a matrix factorization of the form

$$\bar{A} = (L + R)(L + R)^T - E, \tag{3.1}$$

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 but is subsequently discarded, and E has the structure

$$E = RR^T. \tag{3.2}$$

The Tismenetsky incomplete factorization does not compute the full update and thus a positive semidefinite modification is implicitly added to A. The matrix R represents intermediate memory, that is, memory that is used in the construction of the preconditioner but is not part of the preconditioner.

Following the ideas of Kaporin [36], `HSL_MA28` uses drop tolerances to limit the memory required in the computation of the incomplete factorization. The user controls the dropping of small entries from L and R and the maximum number of entries within each column of L and R (and thus the amount of memory for L and the intermediate work and memory used in computing the incomplete factorization). The parameters $lsize$ and $rsize$ must be set by the user to the maximum number of fill entries in each column of L and the number of entries in R, respectively. Further details are given in [58, 59].

We present a summary outline of our left-looking incomplete factorization as Algorithm 3.1. It shows the basic steps but, for simplicity, omits details of our sparse implementation. Here $A_{:,j}$, $L_{:,j}$ and $R_{:,j}$ denote the jth columns of the lower triangular parts of A, L, and R, respectively, and n_j is the number of entries in column j of A.

The scalar *small* is used to determine whether a diagonal entry is sufficiently large; if at any stage a diagonal entry is less than *small*, the factorization is considered to have broken down, and in this case, the shift α is increased and the factorization restarted. $droptol1 > droptol2 \geq 0$ are chosen drop tolerances. The user may choose to supply a positive initial shift α_{in}.

ALGORITHM 3.1. OUTLINE OF THE HSL_MI28 INCOMPLETE CHOLESKY ALGORITHM FOR POSITIVE-DEFINITE A.

Input: *Symmetric positive definite* $A \in R^{n\times n}$*;*
*lsize, rsize, droptol*1*, droptol*2*, initial shift* α_{in}
Output: *Incomplete Cholesky factor L, final shift* α_{out}

Compute a sparsity-preserving ordering Q *for* A *and permute:* $A \leftarrow Q^T AQ$

Compute a diagonal scaling $\bar{S}$ *and scale:* $A \leftarrow \bar{S}A\bar{S}$

Set breakdown = false and $\alpha = \alpha_{in}$, $\alpha_0 = 0$

Loop over shifts
do
  Set $A \leftarrow A + (\alpha - \alpha_0)I$ *and* $d(1:n) = (a_{11}, a_{22}, ..., a_{nn})$
  for $j = 1 : n$ **do**
    Apply LL^T*,* RL^T *and* LR^T *updates from columns* $1 : j-1$ *to* $A_{:,j}$ *and* $d(j+1:n)$
    if $\min(d(j+1:n) < small$ **then**
      Set breakdown = true, $\alpha_0 = \alpha$ *and increase* α
      exit *(Restart factorization with larger shift)*
    end if
    Sort entries of $A_{:,j}$ *by magnitude*
    Keep $n_j + lsize$ *entries of largest magnitude in* $L_{:,j}$*,*
      *provided each is at least droptol*1
    Keep the $rsize$ *entries that are next largest in magnitude in* $R_{:,j}$*,*
      *provided each is at least droptol*2
  end do
  if *breakdown = false* **then**
    Set $\alpha_{out} = \alpha - \alpha_0$ *and* **return** *(Breakdown-free factorization complete)*
  end if
end do

The factorization proceeds column by column. At each stage $j \geq 2$, updates from the previously computed columns 1 to $j-1$ of L and R are applied to column j of A. The update operations are as follows:

```
for k < j and L_{j,k} ≠ 0 do
        A_{j:n,j} ⇐ A_{j:n,j} − L_{j:n,k} ∗ L_{j,k}      ! LL^T updates
        A_{j:n,j} ⇐ A_{j:n,j} − R_{j:n,k} ∗ L_{j,k}      ! RL^T updates
end
for k < j and R_{j,k} ≠ 0 do
        A_{j:n,j} ⇐ A_{j:n,j} − L_{j:n,k} ∗ R_{j,k}      ! LR^T updates
end
```

The diagonal entries of columns $j+1$ to n are also updated. This allows us to detect a potential breakdown as soon as possible. Once a breakdown is detected, the shift α is increased (the strategy for doing this is explained in [58]) and then the factorization of the shifted matrix $A + (\alpha - \alpha_0)I$ is restarted, where α_0 is the previous shift.

Note that the widely used ICFS code of Lin and Moré [41] implements a special case in which ordering is not incorporated, $droptol1 = droptol2 = 0$ and $rsize = 0$ (so that there is no dropping of entries by size, no intermediate memory is used, and only LL^T updates are applied).

4. Signed incomplete Cholesky factorization. We now discuss how to modify Algorithm 3.1 for saddle-point systems while avoiding the need to incorporate pivoting. Our signed IC factorization is listed as Algorithm 4.1 (again, the sparse implementation details are omitted). Here N_j is the number of entries in column j of K. This algorithm is implemented within a new package `HSL_MI30`.

ALGORITHM 4.1. OUTLINE OF THE `HSL_MI30` SIGNED INCOMPLETE CHOLESKY ALGORITHM FOR SADDLE-POINT K.

Input: *Symmetric saddle-point matrix $K \in R^{N\times N}$ with $N = n + m$;*
lsize, rsize, droptol1, droptol2, initial shifts $\alpha_{in}(1:2)$
Output: *Incomplete signed Cholesky factor L, diagonal D with n positive and m negative entries,*
final shifts $\alpha_{out}(1:2)$

Compute a sparsity-preserving ordering Q for K
Post-process the ordering $\hat{Q} \leftarrow Q$ (see section 2)
Permute the matrix: $K \leftarrow \hat{Q}^T K \hat{Q}$

Compute a diagonal scaling $\bar{S}$ and scale: $K \leftarrow \bar{S}K\bar{S}$

Set breakdown = false and $\alpha(1:2) = \alpha_{in}(1:2)$, $\alpha_0(1:2) = 0$

Loop over shifts
do
  Set $K \leftarrow K + G$ where G is diagonal with
    $G_{ii} = \begin{cases} \alpha(1) - \alpha_0(1) & \text{if } i \text{ is an } A \text{ node} \\ \alpha_0(2) - \alpha(2) & \text{if } i \text{ is a } C \text{ node} \end{cases}$
  Set $d(1:N) = (k_{11}, k_{22}, ..., k_{NN})$
  for $j = 1 : N$ **do**
    Apply LL^T, RL^T and LR^T updates from columns $1 : j-1$ to $K_{:,j}$ and $d(j+1:N)$
    if $\min(d(i) : i \text{ is an } A \text{ node}) < small$ **then**
      Set breakdown = true, $\alpha_0(1:2) = \alpha(1:2)$ and increase $\alpha(1)$
      exit *(Restart factorization with larger $\alpha(1)$)*
    else if $\max(d(i) : i \text{ is a } C \text{ node}) > -small$ **then**
      Set breakdown = true, $\alpha_0(1:2) = \alpha(1:2)$ and increase $\alpha(2)$
      exit *(Restart factorization with larger $\alpha(2)$)*
    end if
    Sort entries of $K_{:,j}$ by magnitude
    Keep $N_j + lsize$ entries of largest magnitude in $L_{:,j}$,
      provided each is at least droptol1

Keep the rsize entries that are next largest in magnitude in $R_{:,j}$,
*provided each is at least droptol*2
end do
if *breakdown* = *false* **then**
Set $\alpha_{out}(1:2) = \alpha(1:2) - \alpha_0(1:2)$ *and* **return**
(Breakdown-free factorization complete)
end if
end do

In addition to the step that postprocesses the computed ordering so that a C-node is never ordered before its A-node neighbors, the key difference between the two algorithms is that, for the saddle-point case, we employ two shifts, $\alpha(1:2)$. Both are nonnegative. The first, $\alpha(1)$, is increased each time the incomplete factorization breaks down on an A-node, while the second, $\alpha(2)$, is increased each time the incomplete factorization breaks down on a C-node. For simplicity of notation, assuming the natural ordering is used (so that $Q = I$), the incomplete factorization of the scaled and shifted matrix

$$\bar{K} = \bar{S} \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} \bar{S} + \begin{pmatrix} \alpha(1)I & 0 \\ & -\alpha(2)I \end{pmatrix}$$

is computed. We note that recently, Chen, Phoon, and Toh [12] use a perturbation of the $(2,2)$ block in a zero-level fill incomplete factorization; they refer to this as a stabilized incomplete factorization. In their study, they use a fixed shift $\alpha(2) = -4$ (with $\alpha(1) = 0$). Furthermore, in their paper on incomplete factorization preconditioners for Navier–Stokes solvers, Rehman, Vuik, and Segal [63] comment that convergence can sometimes be improved by perturbing the $(2,2)$ block.

It is clear that the changes that must be made to the Cholesky code are minimal; it is essentially only necessary to distinguish between A and C-nodes. We observe that we did experiment with using a single shift (so that $\alpha(1) = \alpha(2)$) but found that this gave us significantly poorer results.

Note that this algorithm differs from that described in section 1 since, even with the natural ordering, we do not compute an incomplete factorization of A on its own and then solve (1.3) before computing an incomplete factorization of the approximate Schur complement (1.4). Rather, we work with the columns of length $N = n + m$ of the matrix

$$\begin{pmatrix} A \\ B \end{pmatrix}.$$

This has the advantage that the sparsity of the (2,1) block of the incomplete factor is dealt with as the factorization of the (1,1) block proceeds. Furthermore, it is not necessary to limit the factorization exclusively to the A-nodes before proceeding to the C-nodes; we require only that a C-node is ordered after all its A-node neighbors. This gives greater flexibility in the ordering of the factorization.

Finally, we remark that Orban [46] has recently proposed a limited-memory incomplete LDL^T factorization for the important class of SQD matrices. His algorithm is a generalization of that of Lin and Moré [41]. It can be regarded as a special case of Algorithm 4.1 with *droptol*1 = *droptol*2 = 0, *rsize* = 0, and $\alpha(1) = \alpha(2)$ and, as it is for SQD matrices, there is no requirement to postprocess the sparsity-preserving ordering.

5. Theoretical results. This section presents some results that may lead to a better theoretical understanding of limited memory incomplete factorizations. Our results closely follow those in [45, 46]. However, unlike the SQD case that was recently considered by Orban [46], for our saddle-point systems, we cannot assume that the matrices have nonzero diagonal entries and so we are not able to exploit the results summarized, for example, in [7].

Early analysis of incomplete factorizations was built on theory for special matrices, such as M- and H-matrices, that correspond naturally to the early stages of solving linear systems by preconditioned iterative methods. For example, the proof of the breakdown-free property of incomplete factorizations for M-matrices is an important component of the seminal paper of Meijerink and van der Vorst [44] (see also [42, 68]).

DEFINITION 5.1. *A nonsingular real square matrix A with nonpositive off-diagonal entries is called an M-matrix if all the entries of its inverse are nonnegative. A nonsingular real square matrix A is called an H-matrix if its comparison matrix $M(A) = \{M(A)_{ij}\}$ defined as*

$$M(A)_{ij} = \begin{cases} |a_{ij}| & (i = j), \\ -|a_{ij}| & (i \neq j) \end{cases}$$

is an M-matrix.

Note that H-matrices are very close to strictly diagonally dominant matrices [7] (they are so-called generalized diagonally dominant matrices). The following basic result can be found in [7].

LEMMA 5.2. *The Schur complement of an M-matrix (respectively, an H-matrix) is an M-matrix (respectively, an H-matrix). Furthermore, for real square matrices, if A is an H-matrix and elementwise B satisfies $M(B) \geq M(A)$, then B is also an H-matrix.*

We start by considering a single step of three possible factorization approaches: the complete factorization (which we denote as CO), the limited memory approach of Lin and Moré and Orban (LM) [41, 46], and our proposed incomplete symmetric Tismenetsky approach (TI). Assuming $d_1 = a_{11}$ is nonzero, we can write one step of the factorization of a symmetric matrix A in the form

$$A = \begin{pmatrix} d_1 & w^T \\ w & F \end{pmatrix} \equiv \begin{pmatrix} d_1 & l^T & r^T \\ l & F_{LL} & F_{LR} \\ r & F_{RL} & F_{RR} \end{pmatrix}. \tag{5.1}$$

For a CO factorization, we have the Schur complement

$$S_{CO} = F - d_1^{-1} \begin{pmatrix} l \\ r \end{pmatrix} \begin{pmatrix} l^T & r^T \end{pmatrix}. \tag{5.2}$$

The LM factorization keeps only part of the pivot column (part l in (5.1), assuming the rows that correspond to the retained entries are ordered first). Further, Lin and Moré update only F_{LL} and, importantly, the diagonal entries of F_{RR}. The TI approach updates F_{LL}, F_{RL}, and F_{LR} and possibly also the diagonal entries of F_{RR}. The following result is closely related to results in [41, 45, 46].

LEMMA 5.3. *Let A be a symmetric H-matrix with $d_1 = a_{11}$ nonzero. Denote the Schur complements for the CO, LM, and TI factorizations after one step with pivot d_1 by S_{CO}, S_{LM}, and S_{TI}, respectively. Further, let the Schur complement of $M(A)$ be $S_{M(A)}$. Then, elementwise, $S_{M(A)}$ is not larger than any of $M(S_{CO})$, $M(S_{TI})$, and $M(S_{LM})$.*

Proof. From (5.1), the comparison matrix of A is

$$M(A) = \begin{pmatrix} |d_1| & -|l|^T & -|l|^T \\ -|l| & M(F_{LL}) & M(F_{LR}) \\ -|r| & M(F_{RL}) & M(F_{RR}) \end{pmatrix}.$$

After one step of the CO factorization, the Schur complement of $M(A)$ is equal to

$$S_{M(A)} = \begin{pmatrix} M(F_{LL}) - |d_1^{-1}||l||l|^T & M(F_{LR}) - |d_1^{-1}||l||r|^T \\ M(F_{RL}) - |d_1^{-1}||r||l|^T & M(F_{RR}) - |d_1^{-1}||r||r|^T \end{pmatrix}, \tag{5.3}$$

and from (5.2), the comparison matrix of the Schur complement of A is

$$M(S_{CO}) = M\left(F - d_1^{-1}\begin{pmatrix} l \\ r \end{pmatrix}\begin{pmatrix} l^T & r^T \end{pmatrix}\right).$$

Setting $s = \{s_i\} = \begin{pmatrix} l \\ r \end{pmatrix}\begin{pmatrix} l^T & r^T \end{pmatrix}$, the entries of $M(S_{CO})$ can be expressed as

$$M(S_{CO})_{ij} = \begin{cases} |f_{ii} - d_1^{-1}s_i^2|, & i = j, \\ -|f_{ij} - d_1^{-1}s_i s_j|, & i \neq j. \end{cases}$$

Using the triangle inequality elementwise for the diagonal and the off-diagonal entries separately, we obtain

$$|f_{ii} - d_1^{-1}s_i^2| \geq |f_{ii}| - |d_1|^{-1}s_i^2 \tag{5.4}$$

and

$$-|f_{ij} - d_1^{-1}s_i s_j| \geq -|f_{ij}| - |d_1|^{-1}|s_i||s_j| \tag{5.5}$$

for $i = j$ and $i \neq j$, respectively. From (5.3) it follows that $S_{M(A)} \leq M(S_{CO})$ elementwise. Consider now the comparison matrices $M(S_{TI})$ and $M(S_{LM})$. The Schur complement update in both replaces some of the off-diagonal updates $s_i s_j$ by zeros because of dropping. But even in this case the inequalities (5.4) and (5.5) remain valid and the required result may be deduced. Note that the proof does not provide a partial ordering among off-diagonal entries of $M(S_{CO})$, $M(S_{TI})$, and $M(S_{LM})$. ∎

Results of this kind for M- or H-matrices, as well as stronger results that typically assume nonzero diagonal entries, are useful as well as popular. However, the fact that we can get such results for a wide class of incomplete factorization methods potentially indicates that the resulting accuracy may be poor [13]. In the other words, *lower bounds for the sequence of the Schur complements may be weak* and we may obtain a factorization for a problem that is far from the original. We also know that the size of the modification may determine the efficiency of the preconditioned iterative method [19, 22, 58], but see the note in [41] stating that it is not clear whether this conclusion transfers to limited memory preconditioners. We conclude that it is still necessary to have a good choice of incomplete factorization and its implementation. This idea will hopefully remain a driving force for the development of new approaches.

We now consider the TI factorization of the saddle-point matrix (1.1) (without drop tolerances).

THEOREM 5.4. *Consider the incomplete TI factorization of the symmetric saddle-point matrix* (1.1) *with a pivot sequence such that a C-node is ordered only after all*

its A-node neighbors. Assume that for an A-pivot, all its C-node neighbors are in L, and for a C-pivot, all its C-node neighbors are in R. Then all the entries of the diagonal factor D are ± 1 *and the factorization is breakdown-free.*

Proof. Assume $i-1$ steps have succeeded and consider the ith pivot k_{ii}. As in Theorem 2.1, we consider the partitioned ith principal submatrix

$$\begin{pmatrix} A_{i-1} & B_{i-1}^T & u^T \\ B_{i-1} & -C_{i-1} & v^T \\ u & v & k_{ii} \end{pmatrix}.$$

Recall that, at each step, the TI factorization is based on adding a positive semidefinite matrix to the matrix that is being factorized (see [59]). Under the stated assumptions, at the ith step a positive semidefinite matrix has only been added to A_{i-1}; let Δ denote this positive semidefinite matrix.

Case 1: Node *i* is a *A*-node. In this case, the proof is very similar to that of the complete factorization. As in Theorem 2.1, the ordering implies that v is equal to zero. Further, the update of k_{ii} includes just two terms as in (2.3). The first of these terms equal to $u(A_{i-1}+\Delta)^{-1}u^T$. Again, as in (2.3), the second term updating k_{ii} is based on the negative Schur complement $S_{i-1} = C_{i-1} + B_{i-1}(A_{i-1}+\Delta)^{-1}B_{i-1}^T$ and it is equal to $u(A_{i-1}+\Delta)^{-1}B_{i-1}^T S_{i-1}^{-1} B_{i-1}(A_{i-1}+\Delta)^{-1}u^T$ and thus it is also positive.

Case 2: Node *i* is a *C*-node. The update of k_{ii} can be split into two steps. First, we subtract from k_{ii} the positive value $u(A_{i-1}+\Delta)^{-1}u^T$ and so k_{ii} remains negative. Second, k_{ii} should be updated using the negative Schur complement $S_{i-1} = C_{i-1} + B_{i-1}(A_{i-1}+\Delta)^{-1}B_{i-1}^T$. But S_{i-1} corresponds to C-nodes and, because of the assumption that all C-node neighbors of a C-pivot are in R and the TI factorization does not perform RR^T updates, k_{ii} is not updated further and is thus positive. ∎

Remark 5.5. The result in Theorem 5.4 proposes a modification of the signed factorization that safeguards the existence of the IC factorization, even for saddle-point matrices with zero diagonal entries. Note that, in practice, when developing a fill-in efficient implementation, we generally need to relax the theoretical conditions. For example, the diagonal entries of the $(2,2)$ block could be modified by adding a negative semidefinite (diagonal) matrix.

Remark 5.6. Theorem 5.4 does not hold for the LM factorization, even when K is a symmetric saddle-point H-matrix. In this case, even when we have the monotonicity property in the sequence of the Schur complements as described in Lemma 5.3, there is no guarantee that the diagonal C-node entries will be nonzero.

For the LM factorization, we require more stringent assumptions.

THEOREM 5.7. *Assume that the symmetric saddle-point matrix* (1.1) *is an H-matrix and that the constraint ordering is as in Theorem* 5.4. *Assume further that all the A-nodes are fully updated, that is, they correspond to the CO factorization. Then the LM factorization that updates all the diagonal entries at each stage is breakdown-free.*

Proof. The complete factorization of the A-nodes with the diagonal updates of all the nodes is necessary to ensure the diagonal C-node entries become nonzero. The H-matrix property explained in Lemma 5.3 can then be applied for all the subsequent Schur complements. ∎

While the theoretical results presented here appear promising, as already observed, practical implementations almost invariably employ further relaxations and extend solvability from (relatively weak, as we mentioned above) special matrices to more general ones. The real challenge is thus (1) to minimize the fill-in in the

computed factors, (2) to use as much information on the problem as possible for the computation, and (3) to restrict the memory use. The breakdown-free property can then be forced by scaling and moving the scaled matrix closer to a related H-matrix through the use of diagonal shifts [41].

6. Numerical experiments.

6.1. Test environment. Unless stated otherwise, the 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. The ifort Fortran compiler (version 12.0.0) with option -O3 is used. The implementations of the SYMMBK algorithm [11] and the GMRES(100) algorithm (with right preconditioning) [53] offered by the HSL routines `HSL_MI02` and `MI24`, respectively, are 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

$$\|K\hat{x} - b\|_2 \leq 10^{-8}\|b\|_2, \tag{6.1}$$

where $\hat{x}$ is the computed solution. In addition, for each test we impose a limit of 1000 iterations.

Our test problems are real indefinite matrices taken from the University of Florida Sparse Matrix Collection [15]. The first set (Test Set 1) has zero $(2,2)$ block ($C = 0$); they are listed in Table 1. The second set (Test Set 2) is interior-point optimization matrices with $C = 10^{-8}I$; they are listed in Table 2. Here we give the order n of the $(1,1)$ block A, the order m of the $(2,2)$ block C, and the number $nz(K)$ of entries in the lower triangular part of K. In addition, we use the direct solver `HSL_MA97` [33] to compute the number of entries in the complete factor of K. For the `HSL_MA97` runs, we use the scaling from a symmetrized version of the package `MC64` [18, 20]. We remark that this scaling has been found to work well for direct solvers when used to solve "tough" indefinite systems [31, 34]. We also use the default ordering (which is either approximate minimum degree or nested dissection, and this is selected automatically following [21]), except for the problems marked *. For these, we found we obtain a sparser complete factorization by using a matching-based ordering. We report $fill_L$ to be the ratio of the number of entries in the factor to $nz(K)$. This statistic is reported for later comparison with the fill for the incomplete factorizations.

Following [57], in our experiments we define the *efficiency* of the preconditioner P to be

$$efficiency = iter \times nz(L), \tag{6.2}$$

where $iter$ is the iteration count for $P = (LDL^T)^{-1}$. The lower the value of (6.2), the better the preconditioner. We also define the fill in the incomplete factor to be the ratio

$$fill_{IC} = (\text{number of entries in the incomplete factor})/nz(K), \tag{6.3}$$

6.2. Results for $C = 0$. In Table 3, we present results for Test Set 1. The values of the shifts $\alpha_{out}(1:2)$ are reported, together with $fill_{IC}$, the efficiency, and the number of iterations. We use the natural ordering (so that all A-nodes are ordered ahead of the C-nodes) and symmetrized `MC64` scaling. The drop tolerances are $droptol1 = 10^{-3}$ and $droptol2 = 10^{-4}$ (which are the default values for `HSL_MI30`). We ran GMRES(100) for each problem with $lsize = rsize =$1, 5, 10, and 20 and selected

TABLE 1

*Test problems with $C = 0$ (Test Set 1). n and m denote the order of A and C (see (1.1)), $nz(K)$ is the number of entries in the lower triangular part of K, $fill_L$ is the ratio of the number of entries in the complete factor of K to $nz(K)$. QP = quadratic programming problem, FE = finite-element, PDE = partial differential equation. * denotes matching-based ordering is used.*

Identifier	n	m	$nz(K)$	$fill_L$	Description/Application
GHS_indef/aug3dcqp	27543	8000	77829	18.3	3D PDE
GHS_indef/boyd1	93261	18	652246	1.00	Convex QP
GHS_indef/brainpc2*	13807	13800	96601	3.61	Biological model
GHS_indef/cont-201*	40397	40198	239596	20.1	Convex QP
GHS_indef/cont-300*	90597	90298	539396	22.1	Convex QP
GHS_indef/d_pretok	129160	53570	885416	17.3	Mixed FE model
GHS_indef/darcy003	234128	155746	1167685	7.86	Mixed FE model
GHS_indef/mario001	23130	15304	114643	6.51	FE model Stokes problem
GHS_indef/ncvxqp9*	9004	7500	31547	11.4	Nonconvex QP
GHS_indef/olenski0	61030	27233	402623	12.1	Mixed FE model
GHS_indef/qpband	15000	5000	30000	1.67	Convex QP
GHS_indef/sit100	7142	3120	34094	13.7	Mixed FE model
GHS_indef/stokes64	8450	4096	74242	8.98	FE model Stokes problem
GHS_indef/stokes128	33282	16384	295938	10.7	FE model Stokes problem
GHS_indef/tuma1	13360	9607	50560	10.9	Mixed FE model
GHS_indef/tuma2	7515	5477	28440	10.5	Mixed FE model
GHS_indef/turon_m	133814	56110	912345	15.7	Mixed FE model

TABLE 2

Interior-point test problems (Test Set 2). n and m denote the order of A and C (see (1.1)), $nz(K)$ is the number of entries in the lower triangular part of K, $fill_L$ is the ratio of is the number of entries in the complete factor of K to $nz(K)$.

Identifier	n	m	$nz(K)$	$fill_L$
GHS_indef/c-55	19121	13659	218115	21.5
GHS_indef/c-59	23813	17469	260909	17.6
GHS_indef/c-63	25505	18729	239469	13.7
GHS_indef/c-68	36546	28264	315408	29.2
GHS_indef/c-69	38432	29026	345714	10.9
GHS_indef/c-70	39302	29622	363955	13.7
GHS_indef/c-71	44814	31824	468096	37.1
GHS_indef/c-72	47950	36114	395811	11.7
Schenk_IBMNA/c-big	201877	143364	1343126	39.0

the best result (in terms of the lowest value of the efficiency (6.2)). For problems GHS_indef/cont-201 and GHS_indef/cont-300, we did not achieve convergence with these settings but did get convergence by choosing $droptol1 = droptol2 = 0.0$, $lsize = rsize = 20$ and running SYMMBK.

In the positive-definite case, we found that using intermediate memory ($rsize > 0$) was beneficial [58, 59]. To assess whether it is also helpful for the signed IC factorization, we run with the same settings (except $rsize = 0$) and report the results in Table 4. We observe that with $rsize = 0$, more problems require a nonzero shift $\alpha_{out}(2)$ and that the value of the nonzero $\alpha_{out}(2)$ is greater than for $rsize > 0$. Furthermore, with the same $lsize$, the fill ratio $fill_{IC}$ is generally greater and the efficiency poorer. For a number of problems, including GHS_indef/brainpc2 and GHS_indef/stokes64, the iteration count is significantly higher with $rsize = 0$, and for problems GHS_indef/cont-300 and GHS_indef/stokes128, we fail to achieve the requested accuracy within the limit of 1000 iterations. We conclude that, overall, using intermediate memory improves the quality of the incomplete factorization.

TABLE 3
GMRES(100) *convergence results for Test Set* 1. $*$ *indicates SYMMBK was used.*

Identifier	$lsize$	$rsize$	$\alpha_{out}(1)$	$\alpha_{out}(2)$	$fill_{IC}$	efficiency	iter
GHS_indef/aug3dcqp	1	1	0.0	0.0	1.35	1.0×10^{5}	1
GHS_indef/boyd1	1	1	0.0	0.0	0.82	3.8×10^{6}	7
GHS_indef/brainpc2	1	1	0.0	2.56×10^{-1}	1.28	1.6×10^{7}	132
GHS_indef/cont-201*	20	20	0.0	1.00×10^{-3}	7.89	3.9×10^{8}	207
GHS_indef/cont-300*	20	20	0.0	1.00×10^{-3}	7.87	1.6×10^{9}	390
GHS_indef/d_pretok	5	5	0.0	2.56×10^{-1}	1.96	4.8×10^{7}	28
GHS_indef/darcy003	20	20	0.0	0.0	5.44	2.8×10^{8}	44
GHS_indef/mario001	20	20	0.0	0.0	5.37	9.9×10^{6}	16
GHS_indef/ncvxqp9	10	10	0.0	4.19×10^{3}	4.01	2.5×10^{5}	2
GHS_indef/olesnik0	20	20	0.0	0.0	4.43	4.8×10^{7}	27
GHS_indef/qpband	1	1	0.0	0.0	1.17	3.5×10^{4}	1
GHS_indef/sit100	20	20	0.0	0.0	3.61	1.7×10^{6}	14
GHS_indef/stokes64	10	10	0.0	0.0	2.73	3.2×10^{7}	157
GHS_indef/stokes128	10	10	0.0	0.0	2.73	4.3×10^{8}	539
GHS_indef/tuma1	20	20	0.0	0.0	5.79	5.9×10^{6}	20
GHS_indef/tuma2	20	20	0.0	0.0	6.55	3.4×10^{6}	18
GHS_indef/turon_m	20	20	0.0	1.60×10^{-2}	4.36	2.2×10^{8}	56

TABLE 4
GMRES(100) *convergence results for Test Set* 1 *with* $rsize = 0$. $*$ *indicates SYMMBK was used.* – *denotes failure to converge within* 1000 *iterations.*

Identifier	$lsize$	$rsize$	$\alpha_{out}(1)$	$\alpha_{out}(2)$	$fill_{IC}$	efficiency	iter
GHS_indef/aug3dcqp	1	0	0.0	0.0	1.56	1.2×10^{5}	1
GHS_indef/boyd1	1	0	0.0	0.0	0.82	3.8×10^{6}	7
GHS_indef/brainpc2	1	0	0.0	4.10	1.43	7.2×10^{7}	519
GHS_indef/cont-201*	20	0	0.0	1.60×10^{-2}	7.89	1.3×10^{9}	710
GHS_indef/cont-300*	20	0	0.0	1.60×10^{-2}	7.87	–	–
GHS_indef/d_pretok	5	0	0.0	0.0	2.09	5.9×10^{7}	32
GHS_indef/darcy003	20	0	0.0	0.0	6.13	3.1×10^{8}	43
GHS_indef/mario001	20	0	0.0	0.0	6.00	1.0×10^{7}	15
GHS_indef/ncvxqp9	10	0	0.0	4.19×10^{3}	6.44	4.1×10^{5}	2
GHS_indef/olesnik0	20	0	0.0	0.0	5.45	6.8×10^{7}	31
GHS_indef/qpband	1	0	0.0	0.0	1.17	3.5×10^{4}	1
GHS_indef/sit100	20	0	0.0	0.0	4.36	2.4×10^{6}	16
GHS_indef/stokes64	10	0	0.0	1.60×10^{-2}	2.74	5.9×10^{7}	289
GHS_indef/stokes128	10	0	0.0	1.60×10^{-2}	2.73	–	–
GHS_indef/tuma1	20	0	0.0	1.60×10^{-2}	8.22	6.6×10^{6}	16
GHS_indef/tuma2	20	0	0.0	1.60×10^{-2}	7.93	3.2×10^{6}	14
GHS_indef/turon_m	20	0	0.0	2.56×10^{-1}	5.22	3.5×10^{8}	73

It may be important to limit the amount of fill, so in Table 5 we present results with $lsize = 5$, $rsize = 10$. We omit the first six problems and GHS_indef/qpband because either they fail to converge with these settings (problems GHS_indef/cont-201 and GHS_indef/cont-300) or the results already presented in Table 3 use less memory. We see that the fill is now less than 3.0 but, for some problems (including GHS_indef/darcy003 and GHS_indef/mario001), the quality of the preconditioner, in terms of the efficiency as well as the iteration count, is significantly poorer.

We have also considered varying $lsize$ and $rsize$ while keeping $lsize + rsize$ constant. Results for a subset of our test problems for a range of pairs of values $(lsize, rsize)$ are given in Table 6. We see that, as $lsize$ increases, the level of fill increases and the iteration count reduces. However, as $rsize$ increases, breakdown is avoided and the efficiency can improve. Thus we conclude that reducing the memory

TABLE 5
GMRES(100) *convergence results for Test Set* 1 *with* $lsize = 5$, $rsize = 10$.

Identifier	$\alpha_{out}(1)$	$\alpha_{out}(2)$	$fill_{IC}$	efficiency	iter
GHS_indef/darcy003	0.0	0.0	2.69	1.1×10^{9}	349
GHS_indef/mario001	0.0	0.0	2.68	2.7×10^{7}	89
GHS_indef/ncvxqp9	0.0	1.05×10^{3}	3.34	3.2×10^{5}	3
GHS_indef/olesnik0	0.0	0.0	2.13	6.9×10^{7}	81
GHS_indef/sit100	0.0	0.0	2.12	4.0×10^{6}	55
GHS_indef/stokes64	0.0	0.0	1.90	2.9×10^{7}	204
GHS_indef/stokes128	0.0	0.0	1.89	4.1×10^{8}	730
GHS_indef/tuma1	0.0	0.0	2.78	8.3×10^{6}	59
GHS_indef/tuma2	0.0	0.0	2.91	4.4×10^{6}	50
GHS_indef/turon_m	0.0	0.0	2.08	2.3×10^{8}	119

TABLE 6
GMRES(100) *convergence results for problems from Test Set* 1 *with* $lsize + rsize = 20$.

Identifier	$(lsize, rsize)$	$\alpha_{out}(1)$	$\alpha_{out}(2)$	$fill_{IC}$	efficiency	iter
GHS_indef/mario001	$(20, 0)$	0.0	0.0	6.00	1.0×10^{7}	15
	$(15, 5)$	0.0	0.0	4.71	1.3×10^{7}	24
	$(10, 10)$	0.0	0.0	3.84	1.5×10^{7}	35
	$(5, 15)$	0.0	0.0	2.68	2.7×10^{7}	87
GHS_indef/stokes64	$(20, 0)$	0.0	0.0	4.42	3.6×10^{7}	110
	$(15, 5)$	0.0	0.0	3.57	4.1×10^{7}	154
	$(10, 10)$	0.0	0.0	2.73	3.2×10^{7}	157
	$(5, 15)$	0.0	0.0	1.90	2.6×10^{7}	182
GHS_indef/tuma1	$(20, 0)$	0.0	1.60×10^{-2}	8.22	6.7×10^{6}	16
	$(15, 5)$	0.0	0.0	4.90	6.7×10^{6}	27
	$(10, 10)$	0.0	0.0	3.90	7.2×10^{6}	37
	$(5, 15)$	0.0	0.0	2.77	8.3×10^{6}	59
GHS_indef/turon_m	$(20, 0)$	0.0	2.56×10^{-1}	5.22	3.5×10^{8}	73
	$(15, 5)$	0.0	2.56×10^{-1}	3.78	2.3×10^{8}	67
	$(10, 10)$	0.0	0.0	3.01	2.3×10^{8}	83
	$(5, 15)$	0.0	0.0	2.09	2.1×10^{8}	113

used for the factor while increasing the intermediate memory by a corresponding amount can be beneficial.

In Table 7 we present results for different orderings. Here the $lsize$ and $rsize$ values are the same as in Table 3. The orderings tested are the profile reduction ordering of Sloan [48, 60, 61] (as implemented by the HSL package `MC61`), reverse Cuthill–McKee (RCM) [14] (again, implemented within `MC61`), and approximate minimum degree (AMD) [3] (`HSL_MC68`). With the exception of the relaxed Sloan ordering, each ordering algorithm is applied to the sparsity pattern of K and then postprocessed to give a constrained ordering, as described in section 2. Since waiting to order a C-node until all its A-node neighbors have been ordered can lead to significantly denser factors and hence the need to drop more entries from an incomplete factorization, an obvious approach is to try relaxing the constraint. In particular, we could require that only one of the A-node neighbors of the C-node must have already been ordered. During the postprocessing of the initial ordering, this will clearly have the effect of postponing fewer C-nodes and those that are postponed will be brought back into the ordering sooner. This approach with the Sloan ordering applied to K we refer to as a relaxed Sloan ordering. We see that, in most cases, the relaxed Sloan ordering leads to a much poorer quality preconditioner compared to the constrained Sloan ordering. Our observation is that the relaxed ordering breaks down more frequently

TABLE 7

GMRES(100) *iteration results for Test Set* 1 *run with different orderings (*MC64 *scaling). The lowest iteration count for each problem is in bold.* ∗ *indicates SYMMBK was used. – denotes failure to converge within* 1000 *iterations.*

Identifier	*lsize*	*rsize*	Natural	Constrained Sloan	Constrained AMD	Constrained RCM	Relaxed Sloan
GHS_indef/aug3dcqp	1	1	**1**	**1**	**1**	**1**	10
GHS_indef/boyd1	1	1	7	**1**	10	10	10
GHS_indef/brainpc2	1	1	**132**	–	–	251	84
GHS_indef/cont-201*	20	20	**207**	–	741	–	–
GHS_indef/cont-300*	20	20	**390**	–	–	–	–
GHS_indef/d_pretok	5	5	**28**	93	122	96	–
GHS_indef/darcy003	20	20	44	**37**	46	46	308
GHS_indef/mario001	20	20	16	**14**	17	16	46
GHS_indef/ncvxqp9	10	10	2	**1**	2	2	13
GHS_indef/olesnik0	20	20	27	**23**	27	27	–
GHS_indef/qpband	1	1	**1**	**1**	**1**	**1**	**1**
GHS_indef/sit100	20	20	14	**11**	12	12	165
GHS_indef/stokes64	10	10	157	**70**	97	97	139
GHS_indef/stokes128	10	10	539	**197**	444	444	577
GHS_indef/tuma1	20	20	20	12	13	12	**11**
GHS_indef/tuma2	20	20	18	12	**11**	**11**	12
GHS_indef/turon_m	20	20	56	**35**	48	38	–

TABLE 8

GMRES(100) *iteration results for Test Set* 1 *run with different scalings (natural order). The lowest iteration count for each problem is in bold.* ∗ *indicates SYMMBK was used. – denotes failure to converge within* 1000 *iterations.*

Identifier	*lsize*	*rsize*	MC64	MC77	l_2	None
GHS_indef/aug3dcqp	1	1	**1**	**1**	–	79
GHS_indef/boyd1	1	1	7	57	3	**1**
GHS_indef/brainpc2	1	1	132	238	186	**100**
GHS_indef/cont-201*	20	20	**207**	246	209	–
GHS_indef/cont-300*	20	20	**390**	475	–	–
GHS_indef/d_pretok	5	5	**28**	48	44	41
GHS_indef/darcy003	20	20	44	44	42	**41**
GHS_indef/mario001	20	20	**16**	**16**	**16**	**16**
GHS_indef/ncvxqp9	10	10	2	1	1	1
GHS_indef/olesnik0	20	20	27	28	27	**23**
GHS_indef/qpband	1	1	**1**	**1**	**1**	**1**
GHS_indef/sit100	20	20	14	17	15	**12**
GHS_indef/stokes64	10	10	157	171	**78**	149
GHS_indef/stokes128	10	10	**539**	980	–	568
GHS_indef/tuma1	20	20	20	19	**18**	**18**
GHS_indef/tuma2	20	20	18	17	17	**16**
GHS_indef/turon_m	20	20	56	52	**50**	**50**

and can lead to the use of larger shifts. In particular, breakdown occurs on A-nodes, resulting in large values of $\alpha_{out}(1)$. For example, for problem GHS_indef/d_pretok, for the constrained Sloan ordering, $\alpha_{out}(1) = \alpha_{out}(2) = 0.0$, but for the relaxed Sloan ordering, $\alpha_{out}(1) = 524$, $\alpha_{out}(2) = 0.001$, and 12 restarts were required. In general, the constrained Sloan ordering appears to be the best constrained ordering, although it did not lead to the required convergence for GHS_indef/brainpc2 and the tough GHS_indef/cont problems (for these, the natural order, which forces all the C-nodes after all the A-nodes, gives the best results).

In Table 8 we present results for different scalings. The scalings tested are symmetrized MC64, equilibration scaling using MC77 [51, 52], and l_2 scaling, in which the

TABLE 9

GMRES(100) *convergence results for Test Set* 2 *with* $lsize = rsize = 10$ *and* $\alpha_{in}(1:2) = 0.01$. $nshift$ *denotes the number of nonzero shifts used and* T_f *is the incomplete factorization time. The figures in parentheses are the iteration counts for* $lsize = 10$, $rsize = 0$.

Identifier	$\alpha_{out}(1)$	$\alpha_{out}(2)$	$nshift$	T_f	$fill_{IC}$	efficiency	iter	
GHS_indef/c-55	0.01	0.64	6	0.96	2.08	5.3×10^7	117	(147)
GHS_indef/c-59	0.01	0.64	5	1.22	2.10	6.0×10^7	110	(129)
GHS_indef/c-63	0.01	0.64	5	1.23	2.30	4.8×10^7	87	(166)
GHS_indef/c-68	0.01	1.28	5	0.97	2.32	2.7×10^7	37	(43)
GHS_indef/c-69	0.01	0.32	4	1.25	2.35	5.4×10^7	67	(73)
GHS_indef/c-70	0.01	0.32	4	1.66	2.33	6.0×10^7	71	(72)
GHS_indef/c-71	0.01	0.02	2	1.10	2.17	7.9×10^7	78	(88)
GHS_indef/c-72	0.01	0.32	4	0.88	2.39	6.3×10^7	67	(78)
Schenk_IBMNA/c-big	0.01	0.64	5	5.01	2.63	3.8×10^8	109	(268)

TABLE 10

GMRES(100) *convergence results for Test Set* 2 *with* $lsize = rsize = 10$ *and* $\alpha_{in}(1:2) = 0.0$. – *denotes failure to converge within* 1000 *iterations.* $nshift$ *denotes the number of nonzero shifts used and* T_f *is the incomplete factorization time.*

Identifier	$\alpha_{out}(1)$	$\alpha_{out}(2)$	$nshift$	T_f	$fill_{IC}$	efficiency	iter
GHS_indef/c-55	0.0	16.4	9	0.71	2.06	1.3×10^8	281
GHS_indef/c-59	0.0	32.8	9	0.51	2.17	2.2×10^8	382
GHS_indef/c-63	0.0	32.8	9	0.70	2.30	2.1×10^8	383
GHS_indef/c-68	0.0	131	10	1.10	2.34	2.9×10^8	398
GHS_indef/c-69	0.0	65.5	10	1.08	2.39	2.4×10^8	288
GHS_indef/c-70	0.0	32.8	9	1.04	2.31	3.2×10^8	379
GHS_indef/c-71	0.0	32.8	9	1.12	2.16	3.7×10^8	362
GHS_indef/c-72	0.0	65.5	9	0.83	2.39	4.5×10^8	471
Schenk_IBMNA/c-big	0.0	262	11	6.18	–	–	–

entries in column j of K are normalized by the 2-norm of column j. We also report results for no scaling. We see that some problems are well-scaled and each of the three scalings has little effect on the iteration count (for example, GHS_indef/darcey003 and GHS_indef/mario001). However, whereas we achieved convergence in all cases using `MC64` or `MC77` scaling, we had a number of failures if no scaling was used and also we had failures with the l_2 scaling. While no strategy is consistently the best, in these tests, `MC64` generally gives lower iteration counts than `MC77` and the difference is sometimes large (for example, GHS_indef/brainpc2 and GHS_indef/stokes128).

6.3. Results for $C = 10^{-8}I$. In Table 9, we present results for Test Set 2 (interior-point matrices). We use $lsize = rsize =$10 with the (constrained) Sloan ordering, `MC77` scaling, and drop tolerances $droptol1 = 10^{-3}$ and $droptol2 = 10^{-4}$. In these tests, we use initial shifts $\alpha_{in}(1:2) = 0.01$ to regularize the problem [54]. The iteration counts for the same settings but $lsize = 10$, $rsize = 0$ (no intermediate memory) are also reported. We see that using intermediate memory generally leads to a significant improvement in the preconditioner quality. In each case, our choice $\alpha_{in}(2) = 0.01$ was not large enough and we had to increase the C-shift and restart; the number of shifts used is reported.

For these interior-point problems, we found it was beneficial to regularize by using nonzero initial shifts: if we set $\alpha_{in}(1:2) = 0.0$, in our tests the factorization did not break down on an A-node (so that $\alpha_{out}(1) = 0.0$) but it was necessary to significantly increase the C-shift, leading to a much poorer quality preconditioner. This is illustrated in Table 10. Note that the need to use more shifts (and hence

TABLE 11

GMRES(100) *convergence results for Test Set* 2 *with* $lsize = rsize = 30$ *and* $\alpha_{in}(1:2) = 0.01$. *Results are also given for the code SYM-ILDL run with* $fill = 12.0$ *and* $tol = 0.003$. – *denotes failure to converge within* 1000 *iterations.* $nshift$ *denotes the number of nonzero shifts used.*

Identifier	Signed IC						SYM-ILDL		
	$\alpha_{out}(1)$	$\alpha_{out}(2)$	$nshift$	$fill_{IC}$	efficiency	iter	$fill_{IC}$	efficiency	iter
GHS_indef/c-55	0.01	0.08	2	3.37	3.0×10^7	41	4.35	7.4×10^7	78
GHS_indef/c-59	0.0025	0.01	2	3.67	3.9×10^7	41	4.33	1.1×10^8	97
GHS_indef/c-63	0.01	0.02	2	3.79	4.3×10^7	47	4.46	6.5×10^7	61
GHS_indef/c-68	0.01	0.01	1	4.06	1.8×10^7	14	4.31	8.3×10^7	61
GHS_indef/c-69	0.0025	0.01	2	4.00	4.1×10^7	30	3.81	5.5×10^7	42
GHS_indef/c-70	0.01	0.01	1	3.88	6.3×10^7	45	3.61	2.9×10^7	22
GHS_indef/c-71	0.01	0.01	1	3.51	8.7×10^7	53	4.03	7.0×10^7	37
GHS_indef/c-72	0.01	0.01	1	3.91	5.7×10^7	37	4.02	9.7×10^7	61
Schenk_IBMNA/c-big	0.01	0.01	1	4.44	3.3×10^8	56	4.47	–	–

TABLE 12

Times (in seconds) for performing the incomplete factorization and running GMRES(100). T_f *and* T_g *are the times to compute the factorization and run GMRES, respectively. The total time* T *is the sum of* T_f *and* T_g. – *denotes failure to converge within* 1000 *iterations.*

Identifier	Signed IC (HSL_MI30)						SYM-ILDL		
	$lsize = rsize = 10$			$lsize = rsize = 30$					
	T_f	T_g	T	T_f	T_g	T	T_f	T_g	T
GHS_indef/c-55	0.96	0.90	1.86	3.23	0.32	3.55	1.95	0.85	2.80
GHS_indef/c-59	1.22	1.10	2.32	2.22	0.42	2.64	2.49	1.39	3.88
GHS_indef/c-63	1.23	0.86	2.09	2.30	0.49	2.79	1.59	0.78	2.67
GHS_indef/c-68	0.97	0.43	1.41	3.19	0.20	3.39	3.26	1.09	4.35
GHS_indef/c-69	1.25	0.97	2.23	3.88	0.46	4.34	1.97	0.70	2.67
GHS_indef/c-70	1.66	1.10	2.76	3.14	0.77	3.91	2.10	0.34	2.44
GHS_indef/c-71	1.10	1.53	2.63	4.68	1.34	6.02	4.12	0.96	5.08
GHS_indef/c-72	0.88	1.30	2.18	2.45	0.99	3.44	2.76	1.43	4.19
Schenk_IBMNA/c-big	5.01	16.33	21.34	14.52	8.48	23.00	82.26	-	-

to restart the factorization a greater number of times) does not necessarily lead to a slower factorization time. For fixed $lsize$ and $rsize$, the important point for the factor time is not how many breakdowns there are but how early in the factorization the breakdowns occur: an early breakdown will lead to little increase in the time compared to no breakdown but a breakdown when the factorization is close to completion will effectively double the time.

To illustrate the effects of using larger $lsize$ and $rsize$ values, in Table 11 we present results for $lsize = rsize = 30$; timings are given in Table 12. If we compare these with Table 9 ($lsize = rsize = 10$), we see that the fill increases (but is still much less than for a complete factorization), while the value of the C-shift is reduced. This leads to a reduction in the iteration count (by around 50% for many of the problems) and an improvement in the efficiency for most cases. However, the timings T_f reported in Table 12 (which include times for restarting after a shift change) illustrate that when using the preconditioner to solve a single problem (or a small number of problems), faster total times are achieved with $lsize = rsize = 10$. If the preconditioner is to be used for many problems, the additional time needed to compute the incomplete factorization with larger $lsize$ and $rsize$ can be offset by savings in the GMRES times. For example, for problem GHS_indef/c-55, the time for solving 10 systems using the factorization computed with $lsize = rsize = 10$ is 9.96 seconds, whereas with $lsize = rsize = 30$ the time drops by more than a half to 3.45 seconds.

6.4. Comparisons with SYM-ILDL. It is of interest to consider how the performance of our signed incomplete factorization preconditioner compares with that of an incomplete indefinite factorization that incorporates pivoting. The package we use for comparison is SYM-ILDL by Greif and Liu. As discussed in section 1, this is based on the earlier work by Li and Saad [38] and performs an incomplete factorization of sparse symmetric indefinite matrices with Bunch–Kaufman pivoting [10] used for numerical stability (and to prevent breakdown). Thus, for any symmetric indefinite K (which need not be a saddle-point matrix) it computes an incomplete factorization of the form LDL^T, where D is block diagonal, with blocks of order 1 and 2, corresponding to 1×1 and 2×2 pivots. The matrix is preordered using AMD or RCM and prescaled to be equilibrated in the maximum norm. The input parameters that can be set by the user to control the number of entries within L are $fill$ and tol. Each column of the computed incomplete factor L is guaranteed to have fewer than $fill \cdot ne(K)/n$ entries, where $ne(K)$ is the number of entries in K (upper and lower triangular parts). It has default value 1.0. The parameter tol controls the aggressiveness of the dropping of small entries. In each column k of L, entries that are less than $tol \cdot ||L_{k+1:n,k}||_1$ in magnitude are discarded. The default setting for tol is 0.001. SYM-ILDL is a C++ code and in our tests we use the g++ compiler with option -03.

Table 11 includes results for SYM-ILDL for the Test Set 2 problems run with the settings $fill = 12.0$ and $tol = 0.003$ and the AMD ordering is used. With these choices, the level of fill is similar to that for the signed IC factorization. We see that in some cases (in particular, c-70), SYM-ILDL produces a higher-quality preconditioner, but for other problems (including c-63 and c-68) the preconditioner computed by our signed IC factorization is better.

We next report timings for our signed IC code `HSL_MI30` and for SYM-ILDL. These runs are performed on an Intel Core 2 Quad Q8400 2.66-GHz processor; the gfortran and g++ compilers are used for the two codes, respectively (both with option -O3). We employ the "standalone" mode for SYM-ILDL (the alternative is to call SYM-ILDL from within MATLAB). This requires the user to execute the compiled program through the command line. The input matrix K is held in a file that is named on the command line; the optional parameters that control the action must also be entered on the command line. In addition, a flag may be set to indicate whether the output matrices should be saved. If so, they are written in coordinate format to files in an external folder. The matrices that are saved (using our notation of S for a scaling matrix and Q for a permutation matrix) are K, L, D, Q, S, and $B = Q^T SKSQ$. To run an iterative solver, it is necessary for the user to write a program to both read the matrices L, D, Q, S and to use them as a preconditioner with an iterative solver. We have written a Fortran code to do this. In Table 12 we do *not* include the time taken to write out the matrices and then read them back in; instead, for both codes, we report only the time T_f to compute the factorization, the time T_g for GMRES, and the total time T, which is just the sum of these two. For GMRES, a simple sparse matrix-vector product is used; it is beyond the scope of this study to implement efficient parallel routines for these products but doing so could potentially substantially reduce T_g, altering the relative costs of the factorization and iteration phases of the solution process. Note that the factorization time includes the time taken for computing the ordering and scaling and, for `HSL_MI30`, it includes the time to restart the factorization after a shift change. We observe that the command line interface for SYM-ILDL makes it unsuitable currently for incorporation into another code, such as an interior-point solver.

TABLE 13
GMRES(100) *convergence results for Test Set* 1. *The settings for* $lsize$ *and* $rsize$ *for* HSL_MI30 *are as for Table* 3. $nshift$ *denotes the number of nonzero shifts used.*

Identifier	HSL_MI30				SYM-ILDL			
	$nshift$	$fill_{IC}$	Efficiency	iter	$fill$	$fill_{IC}$	efficiency	iter
GHS_indef/aug3dcqp	0	1.25	9.7×10^{4}	1	1.0	1.61	1.2×10^{5}	1
GHS_indef/boyd1	2	0.40	1.3×10^{7}	52	1.0	0.86	1.1×10^{7}	19
GHS_indef/brainpc2	3	1.28	2.9×10^{7}	238	1.0	0.72	1.5×10^{6}	22
GHS_indef/d_pretok	0	1.88	8.0×10^{7}	48	1.0	1.96	2.7×10^{8}	156
GHS_indef/darcy003	0	5.02	2.0×10^{8}	35	10.0	4.84	2.3×10^{8}	40
GHS_indef/mario001	0	4.91	7.9×10^{6}	11	4.0	4.25	1.2×10^{7}	24
GHS_indef/ncvxqp9	3	5.28	1.7×10^{5}	1	3.0	3.64	7.0×10^{4}	61
GHS_indef/olesnik0	0	3.77	3.9×10^{7}	26	4.0	4.81	6.6×10^{7}	34
GHS_indef/qpband	0	1.17	3.5×10^{4}	1	0.5	1.67	3.5×10^{4}	1
GHS_indef/sit100	0	3.17	2.2×10^{6}	20	5.0	3.24	6.2×10^{6}	56
GHS_indef/stokes64	0	2.73	1.2×10^{7}	57	2.0	3.73	1.1×10^{8}	379
GHS_indef/stokes128	0	2.73	1.2×10^{8}	149	4.0	5.86	1.3×10^{9}	740
GHS_indef/tuma1	4	5.39	6.3×10^{6}	23	15.0	4.91	3.2×10^{6}	13
GHS_indef/tuma2	4	5.25	3.3×10^{6}	22	15.0	4.66	1.4×10^{6}	11
GHS_indef/turon_m	0	3.84	1.5×10^{8}	42	6.0	5.58	3.5×10^{8}	67

TABLE 14
Times (in seconds) for performing the incomplete factroization and running GMRES(100). T_f *and* T_g *are the times to compute the factorization and run GMRES, respectively. The total time* T *is the sum of* T_f *and* T_g.

Identifier	HSL_MI30			SYM-ILDL		
	T_f	T_g	T	T_f	T_g	T
GHS_indef/aug3dcqp	0.03	0.01	0.04	0.06	0.05	0.11
GHS_indef/boyd1	0.23	0.67	0.91	0.30	0.23	0.53
GHS_indef/brainpc2	4.05	1.03	5.07	1.24	0.05	1.29
GHS_indef/d_pretok	0.56	2.28	2.85	1.18	10.43	11.60
GHS_indef/darcy003	3.39	4.09	7.49	3.87	4.95	8.82
GHS_indef/mario001	0.30	0.09	0.39	0.32	0.16	0.48
GHS_indef/ncvxqp9	0.13	0.00	0.13	0.09	0.16	0.25
GHS_indef/olesnik0	0.88	0.52	1.40	1.43	0.81	2.24
GHS_indef/qpband	0.17	0.00	0.18	0.02	0.00	0.02
GHS_indef/sit100	0.06	0.03	0.09	0.09	0.10	0.19
GHS_indef/stokes64	0.08	0.14	0.22	0.25	1.31	1.56
GHS_indef/stokes128	0.33	1.83	2.16	2.28	14.75	17.03
GHS_indef/tuma1	0.38	0.08	0.46	0.19	0.04	0.23
GHS_indef/tuma2	0.20	0.04	0.23	0.09	0.02	0.11
GHS_indef/turon_m	2.28	2.63	4.91	4.05	5.68	9.73

In Table 12, HSL_MI30 times are given for $lsize = rsize = 10$ and $lsize = rsize = 30$. We see that the former produces both faster factorization times and faster GMRES times compared to $lsize = rsize = 30$ and SYM-ILDL. However, with the notable exception of Schenk_IBMNA/c-big, for most problems, T_f is less for SYM-ILDL than for HSL_MI30 with $lsize = rsize = 30$, although in some cases where the latter requires fewer iterations, HSL_MI30 has the smaller total time.

Finally, we consider Test Set 1 ($C = 0$). Results are given in Tables 13 and 14. Again, the HSL_MI30 times include the time to restart the factorization after a change in shift (the number of shifts is reported in column 2 of Table 13). The input parameter $fill$ for SYM-ILDL is chosen for each problem to give good performance with a level of fill in the factor that is generally similar to that for the signed IC factorization; the default drop tolerance $tol = 0.001$ is used. Problems GHS_indef/cont-201 and

GHS_indef/cont-300 are omitted as SYM-ILDL did not converge. The signed IC code is run with the MC77 equilibration scaling and the same choices for $lsize$ and $rsize$ as in Table 3. The natural order is used for the first four problems and the (constrained) Sloan ordering for the remainder. Again, we see that the signed IC approach can outperform SYM-ILDL (for example, the Stokes problems), but for some problems (such as GHS_indef/brainpc2), the latter is faster and produces a higher-quality preconditioner. We are not able to predict for which problems which approach will give the better results.

7. Concluding remarks. In this paper, we have looked at extending the robust limited-memory incomplete Cholesky factorization algorithm of [58, 59] to sparse symmetric indefinite systems in saddle-point form. By using two diagonal shifts to prevent breakdown, we are able to compute a signed incomplete Cholesky factorization of the form LDL^T, where the diagonal matrix D has entries ± 1. Some new theoretical results have been given and numerical results presented to illustrate the effectiveness of the approach. The effects of different orderings and scalings on the preconditioner have also been investigated. As in the positive-definite case, we have shown that the use of intermediate memory can improve the quality of the preconditioner. Furthermore, the use of regularization parameters (that is, nonzero initial diagonal shifts) for the blocks A and $-C$ has been found to substantially improve performance for interior-point optimization matrices. We have developed a new software package HSL_MI30 that implements our signed IC factorization algorithm; this Fortran package (which also offers a MATLAB interface) is part of the HSL mathematical software library [35].

We have presented some numerical results for the recent SYM-ILDL package, which uses Bunch–Kaufman pivoting to avoid breakdown. In the case of a direct solver, it has been reported [56] that a signed Cholesky factorization code in general performs less well than a carefully engineered indefinite code that incorporates threshold partial pivoting. However, our results show that our limited-memory signed IC approach (implemented as HSL_MI30) can be competitive with SYM-ILDL. It is now our intention to develop an incomplete indefinite factorization code that incorporates pivoting and also uses intermediate memory.

Acknowledgments. We are grateful to Chen Greif and Paul Liu for discussions around their code SYM-ILDL and for advice on computing the numerical results for SYM-ILDL that are reported in sections 6.3 and 6.4. Thanks also to Chen for carefully reading and commenting on a draft of this paper. The version of SYM-ILDL used was downloaded from https://github.com/inutard/matrix-factor on December 10, 2013. All results were performed using the HSL implementations of SYMMBK and GMRES (available from http://www.hsl.rl.ac.uk/catalogue/). We would also like to thank two anonymous referees for their careful reading of our manuscript and for their helpful and constructive comments. Travel support from the Academy of Sciences of the Czech Republic is also acknowledged.

REFERENCES

[1] G. Al-Jeiroudi, J. Gondzio, and J. Hall, *Preconditioning indefinite systems in interior point methods for large scale linear optimisation*, Optim. Methods Softw., 23 (2008), pp. 345–363.

[2] A. Altman and J. Gondzio, *Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization*, Optim. Methods Softw., 11 (1999), pp. 275–302.

[3] P. R. Amestoy, T. A. Davis, and I. S. Duff, *Algorithm* 837*: AMD, an approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 381–388.
[4] M. Benzi, G. H. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta Numer., 14 (2005), pp. 1–137.
[5] M. Benzi and A. Wathen, *Some preconditioning techniques for saddle point problems*, in Model Order Reduction: Theory, Research Aspects and Applications, Math. Ind. 13, Springer, New York, 2008, pp. 195–211.
[6] L. Bergamaschi, J. Gondzio, and G. Zilli, *Preconditioning indefinite systems in interior point methods for optimization*, Comput. Optim. Appl., 28 (2004), pp. 149–171.
[7] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979.
[8] S. Bonettini and V. Ruggiero, *Some iterative methods for the solution of a symmetric indefinite KKT system*, Comput. Optim. Appl., 38 (2007), pp. 3–25.
[9] R. Bridson, *An Ordering Method for the Direct Solution of Saddle-Point Matrices*, preprint, http://www.cs.ubc.ca/~rbridson/kktdirect (2007).
[10] J. R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp., 31 (1977), pp. 162–179.
[11] R. Chandra, *Conjugate Gradient Methods for Partial Differential Equations*, Ph.D. thesis, Yale University, 1978.
[12] X. Chen, K.-K. Phoon, and K.-C. Toh, *Performance of zero-level fill-in preconditioning techniques for iterative solutions with geotechnical applications*, Internat. J. Geomechanics, 12 (2012), pp. 596–605.
[13] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.
[14] E. H. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 24th National Conference of the ACM, ACM Press, 1969, pp. 157–172.
[15] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011).
[16] A. C. de Niet and F. W. Wubs, *Numerically stable LDL^T-factorization of F-type saddle point matrices*, IMA J. Numer. Anal., 29 (2009), pp. 208–234.
[17] I. S. Duff, *MA*57*: A new code for the solution of sparse symmetric definite and indefinite systems*, ACM Trans. Math. Software, 30 (2004), pp. 118–154.
[18] I. S. Duff and J. Koster, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 973–996.
[19] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.
[20] I. S. Duff and S. Pralet, *Strategies for scaling and pivoting for sparse symmetric indefinite problems*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 313–340.
[21] I. S. Duff and J. A. Scott, *Towards an Automatic Ordering for a Symmetric Sparse Direct Solver*, Technical report RAL-TR-2006-001, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2005.
[22] V. Eijkhout, *Analysis of parallel incomplete point factorizations*, Linear Algebra Appl., 154/156 (1991), pp. 723–740.
[23] H. C. Elman and G. H. Golub, *Inexact and preconditioned Uzawa algorithms for saddle point problems*, SIAM J. Numer. Anal., 31 (1994), pp. 1645–1661.
[24] R. Fourer and S. Mehrotra, *Solving symmetric indefinite systems in an interior-point method for linear programming*, Math. Program., 62 (1993), pp. 15–39.
[25] P. E. Gill, M. A. Saunders, and J. R. Shinnerl, *On the stability of Cholesky factorization for symmetric quasidefinite systems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 35–46.
[26] G. H. Golub and C. F. Van Loan, *Unsymmetric positive definite linear systems*, Linear Algebra Appl., 28 (1979), pp. 85–97.
[27] J. Gondzio, *HOPDM (version* 2.12*): A fast LP solver based on a primal-dual interior point method*, European J. Oper. Res., 85 (1995), pp. 221–225.
[28] C. Greif, *Preconditioners for linear systems arising from interior-point methods*, International Conference on Preconditioning Techniques for Scientific and Industrial Applications, University of Oxford, 2013.
[29] C. Greif, S. He, and P. Liu, *SYM-ILDL: C++ Package for Incomplete Factorizations of Symmetric Indefinite Matrices*, https://github.com/inutard/matrix-factor (2013).
[30] M. Hagemann and O. Schenk, *Weighted matchings for preconditioning symmetric indefinite linear systems*, SIAM J. Sci. Comput., 28 (2006), pp. 403–420.
[31] J. D. Hogg and J. A. Scott, *The Effects of Scalings on the Performance of a Sparse Symmetric Indefinite Solver*, Technical report RAL-TR-2008-007, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2008.

[32] J. D. Hogg and J. A. Scott, *An Indefinite Sparse Direct Solver for Large Problems on Multicore Machines*, Technical report RAL-TR-2010-011, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2010.

[33] J. D. Hogg and J. A. Scott, *HSL_MA97: A Bit-Compatible Multifrontal Code for Sparse Symmetric Systems*, Technical report RAL-TR-2011-024, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2011.

[34] J. D. Hogg and J. A. Scott, *Pivoting strategies for tough sparse indefinite systems*, ACM Trans. Math. Software, 40 (2013).

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

[36] 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.

[37] 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.

[38] N. Li and Y. Saad, *Crout versions of ILU factorization with pivoting for sparse symmetric matrices*, Electron. Trans. Numer. Anal., 20 (2005), pp. 75–85.

[39] N. Li, Y. Saad, and E. Chow, *Crout versions of ILU for general sparse matrices*, SIAM J. Sci. Comput., 25 (2003), pp. 716–728.

[40] S. X. Li and J. W. Demmel, *Making parallel Gaussian elimination scalable by static pivoting*, in Proceedings of SuperComputing'98, Orlando, FL, 1998, pp. 519–523.

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

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

[43] O. Marques, *Skypack User's Guide*, Technical report, National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory, 2009.

[44] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[45] A. Messaoudi, *On the stability of the incomplete LU-factorizations and characterizations of H-matrices*, Numer. Math., 69 (1995), pp. 321–331.

[46] D. Orban, *Limited-Memory LDLT Factorization of Symmetric Quasi-Definite Matrices*, Technical report G-2013-87, GERAD, Montreat, QC, Canada, 2013.

[47] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[48] 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.

[49] W. Ren and J. Zhao, *Iterative methods with preconditioners for indefinite systems*, J. Comput. Math., 17 (1999), pp. 89–96.

[50] M. Rozložník, A. Smoktunowicz, and F. Okulicka-Dłużewska, *Indefinite orthogonalization with rounding errors*, submitted.

[51] D. Ruiz, *A Scaling Algorithm to Equilibrate Both Rows and Columns Norms in Matrices*, Technical report RAL-TR-2001-034, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001.

[52] D. Ruiz and B. Uçar, *A Symmetry Preserving Algorithm for Matrix Scaling*, Technical report Inria RR-7552, Inria, Grenoble, France, 2011.

[53] Y. Saad and M. H. Schulz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[54] M. A. Saunders and J. A. Tomlin, *Solving Regularized Linear Programs Using Barrier Methods and KKT Systems*, Technical report SOL-96-4, SOL, Department of Operations Research, Stanford University, 1996.

[55] O. Schenk and K. Gärtner, *On fast factorization pivoting methods for symmetric indefinite systems*, Electron. Trans. Numer. Anal., 23 (2006), pp. 158–179.

[56] J. A. Scott, *A Note on a Simple Constrained Ordering for Saddle-Point Systems*, Technical report RAL-TR-2009-007, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2009.

[57] J. A. Scott and M. Tůma, *The importance of structure in incomplete factorization preconditioners*, BIT Numer. Math., 51 (2011), pp. 385–404.

[58] J. A. Scott and M. Tůma, `HSL_MI28`*: An efficient and robust limited memory incomplete Cholesky factorization code*, ACM Trans. Math. Software, 40 (2014).

[59] J. A. Scott and M. Tůma, *On positive semidefinite modification schemes for incomplete Cholesky factorization*, SIAM J. Sci. Comput., 36 (2014), pp. A609–A633.
[60] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, Internat. J. Numer. Methods Engrg., 23 (1986), pp. 239–251.
[61] S. W. Sloan, *A Fortran program for profile and wavefront reduction*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 2651–2679.
[62] M. Tůma, *A note on the LDLT decomposition of matrices from saddle-point problems*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 903–915.
[63] M. ur Rehman, C. Vuik, and G. Segal, *A comparison of preconditioners for incompressible Navier-Stokes solvers*, Internat. J. Numer. Methods Fluids, 57 (2008), pp. 1731–1751.
[64] H. A. van der Vorst, *Iterative Krylov Methods for Large Linear Systems*, Cambridge Monogr. Appl. Comput. Math., Cambridge University Press, Cambridge, UK, 2003.
[65] R. J. Vanderbei, *Symmetric quasidefinite matrices*, SIAM J. Optim., 5 (1995), pp. 100–113.
[66] R. J. Vanderbei, *LOQO user's manual—version* 3.10, Optim. Methods Softw., 11/12 (1999), pp. 485–514.
[67] R. J. Vanderbei and D. F. Shanno, *An interior-point algorithm for nonconvex nonlinear programming*, Comput. Optim. Appl., 13 (1999), pp. 231–252.
[68] 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.
[69] J. Zhao, *The generalized Cholesky factorization method for saddle-point problems*, Appl. Math. Comput., 92 (1998), pp. 49–58.