Preprint
RAL-P-2015-002

Solving symmetric indefinite systems using memory efficient incomplete factorization preconditioners

J Scott, M Tuma,

February 2015

Submitted for publication in Siam Journal of Scientific Computing

ISSN 1361- 4762

SOLVING SYMMETRIC INDEFINITE SYSTEMS USING MEMORY EFFICIENT INCOMPLETE FACTORIZATION PRECONDITIONERS

JENNIFER SCOTT* AND MIROSLAV TŮMA†

Abstract. Sparse symmetric indefinite linear systems of equations arise in numerous practical applications. In many situations, an iterative method is the method of choice but a preconditioner is normally required for this to be effective. In this paper, the focus is on the development of incomplete factorization algorithms that can be used to compute high quality preconditioners for general indefinite systems, including saddle-point problems. A limited memory approach is used that generalises recent work on incomplete Cholesky factorization preconditioners. A number of new ideas are proposed with the goal of improving the stability, robustness and efficiency of the resulting preconditioner. These include the monitoring of stability as the factorization proceeds and the use of pivot modifications when potential instability is observed. Numerical experiments involving test problems arising from a range of real-world applications are used to demonstrate the effectiveness of our approach and comparisons are made with a state-of-the-art sparse direct solver.

Key words. sparse matrices, sparse linear systems, indefinite symmetric systems, iterative solvers, preconditioning, incomplete factorizations, pivoting.

AMS subject classifications. Primary, 65F05, 65F50; Secondary, 15A06, 15A23

1. Introduction.

Large sparse symmetric indefinite linear systems of equations arise in a wide variety of practical applications. In many cases, the systems are of saddle-point type; for a comprehensive overview of the numerical solution of saddle-point problems, see the paper by Benzi et al. [5]. However, in other cases (including problems coming from statistics, acoustics, optimization, eigenvalue problems, and sequences of shifted systems), the indefinite systems possess no nice block structure. The development of incomplete factorization preconditioners that are applicable to saddle-point systems as well as to more general indefinite systems is the main focus of this paper.

In recent years, considerable effort has gone into the development of robust and efficient sparse direct solvers for symmetric indefinite systems. Notable examples include `MA57` [19], `HSL_MA97`, [36, 37], MUMPS [2], PARDISO [60], WSMP [32] and, most recently, the package `SSIDS` for GPU architectures [34]. A key feature of these solvers is the employment of sparsity preserving orderings combined with 1×1 and 2×2 pivots for numerical stability (see, for example, [3, 39, 56]). A numerical overview can be found in the survey [28].

A significant attraction of direct methods is that they can often be used (possibly in combination with a refinement technique) as black box solvers. Moreover, the increase in the amount of main memory available on modern computers as well as the use of out-of-core techniques (such as described, for example, in [55]) has extended the size of systems that they can tackle to well beyond that which could have been envisaged 20 years ago. Nevertheless, to solve still larger systems (typically those from three dimensional problems) it is necessary to use an iterative method. Iterative methods can also be the most efficient option if only an approximation to the solution is needed (for example, if the problem data is not known to high accuracy). However, in general, to be effective iterative methods need to be used in combination with a preconditioner. Unfortunately, the construction of a suitable preconditioner is highly problem dependent. A number of possible approaches have been proposed for indefinite systems. For those of saddle-point type, an enormous amount of effort has gone into exploiting the underlying block structure and retaining it throughout the solution process. An overview of work on these so-called segregated approaches up until 2005 can be found in [5]. Other techniques that make use of the block structure include, for example, constraint preconditioners [42, 48], and symmetric-triangular (ST) preconditioners [72]. Alternatively, the saddle-point structure may be partially exploited. For example, the structure may be used as a

*Scientific Computing Department, Rutherford Appleton Laboratory, Harwell Oxford, Oxfordshire, OX11 0QX, UK. `jennifer.scott@stfc.ac.uk`. Supported by EPSRC grant EP/I013067/1.

†Institute of Computer Science, Academy of Sciences of the Czech Republic. `tuma@cs.cas.cz`. Partially supported by the Grant Agency of the Czech Republic Project No. 13-06684 S. Travel support from the Academy of Sciences of the Czech Republic is also acknowledged.

starting point before the blocks are "mixed" through the use of more general symmetric permutations. The motivation here is that general permutations can lead to incomplete factorization preconditioners that are sparser (and cheaper to apply) than those resulting from a segregated approach. A theoretical background that supports such approaches used, for example, in optimization, is available for symmetric quasi-definite (SQD) systems. Vanderbei [70] shows that such matrices are strongly factorizable while a stability analysis related to the factorization of SQD matrices is given by Gill et al. [26] (see also [27]). Recent work emphasizing the connection of optimization and numerical linear algebra includes [30, 31, 51] while an overview of various approaches for saddle-point problems in optimization that use indefinite factorizations rather than the segregated approach is given in [65].

In some approaches, the incomplete factorization of an indefinite matrix is forced to be positive definite. This may be achieved by taking absolute values of 1×1 pivots and transforming 2×2 pivots into positive definite matrices; see the influential paper by Gill et al. [25] and the recent symmetric positive definite (absolute value) preconditioner of Vecharynski and Knyazev [71]. The motivation for doing this is that a symmetric Krylov space method with short recurrences such as MINRES [52], the conjugate residual method [47, 68], or SYMMLQ [52] can then be employed. For saddle-point systems, Gould, Orban and Rees [29] recently showed that MINRES and SYMMLQ are well defined in the presence of an indefinite preconditioner. For more general indefinite systems, if the preconditioner is indefinite, a general non symmetric iterative method such as GMRES [57] needs to be used since, in general, the preconditioned matrix is not symmetric in any inner product [53].

As already observed, in many practical applications the indefinite system does not have a saddle-point structure. Again, it is possible to construct either a positive definite or a general indefinite preconditioner, with implications for the choice of iterative method. An important contribution by Chow and Saad [14] considers the more general class of incomplete LU preconditioners for solving indefinite problems. While there were some early attempts to compute such preconditioners, efficient implementations were lacking. The 2005 work of Li and Saad [44] represented a key step in the development of well-implemented general indefinite preconditioners based on a left-looking approach. Li and Saad integrated pivoting procedures [3, 12, 13] with scaling [10] and reordering [16]. Building on this, Greif, He, and Liu [31] recently developed a new incomplete factorization package SYM-ILDL for general symmetric indefinite matrices. This code includes optional equilibration scaling of the matrix and ordering using the reverse Cuthill-McKee algorithm [16] or approximate minimum degree [1], while Bunch-Kaufman pivoting [12] is employed to try and maintain stability and avoid breakdown.

Sophisticated ordering techniques based on bipartite graph matchings motivated further ideas for the development of incomplete factorizations of indefinite systems. In the mid 1990s, Olschowka and Neumaier [50] introduced the use of weighted matchings for the scaling of non symmetric matrices; their ideas were developed and implemented for sparse non symmetric problems in [22]; see also their use for ILU factorizations coupled with non symmetric iterative solvers in [6, 61]. The symmetric adaption was originally proposed by Duff and Gilbert [20] and was subsequently built upon by Duff and Pralet [24] (see also [38]), while Hagemann and Schenk [33] employed matching-based orderings when computing incomplete factorizations of symmetric indefinite systems.

In this paper, we consider incomplete factorizations of general indefinite systems (including saddle-point systems) and propose a number of new ideas with the goal of improving the stability, robustness and efficiency of the resulting preconditioner. Our incomplete factorization is based on the limited memory version of the approach of Tismenetsky [69] that we proposed and implemented in the software package `HSL_MI28` [63, 64] for symmetric positive definite systems (see also [65] for saddle-point problems). Incomplete factors L and R are computed; R is used in the computation of L but is normally subsequently discarded. Using a nonzero R can significantly improve the quality of the preconditioner and, in the positive definite case, it was demonstrated in [64] that its memory can be used to enable L to be sparser and thus more efficient to apply. Our proposed approach for general symmetric indefinite systems incorporates a number of ordering and pivoting strategies and uses preprocessing to try to minimize the modifications in the pivot ordering needed for stability. Further, our factorization algorithm includes the use of diagonal modifications to improve stability and generalizes such modifications to 2×2 blocks. Even with well bounded entries in the factors, the triangular solves that are needed when applying an incomplete factorization preconditioner can be highly unstable. One of our key ideas is that of monitoring stability as the factorization proceeds and, in the case of possible instability, we propose using the intermediate R memory to formulate an optimization problem that aims to improve the stability of the factorization by moving either all of R or selected entries of R to L.

The rest of the paper is organised as follows. In Section 2, we describe our incomplete factorization algorithm and the different pivoting strategies that it offers. Then in Section 3, we look at using a shift and/or a diagonal multiplier to prevent the factorization from becoming unstable. We introduce the concept of local growth and show how 1×1 or 2×2 pivots can be modified to reduce the local growth. In Section 4, we propose monitoring possible instability as the factorization proceeds. Numerical results for a range of problems from real-world applications are presented in Section 5; these demonstrate the efficiency and effectiveness of our proposed approach. Finally, in Section 6, our findings are summarised and some concluding comments are made.

2. Factorization and pivoting.

In this section, we describe our incomplete factorization algorithm and the pivoting options that it offers.

2.1. Limited-memory incomplete factorization.

We first summarize the limited-memory incomplete Cholesky (IC) factorization algorithm that is implemented within the package `HSL_MI28` from the HSL mathematical software library [40, 63]. 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, shifted to avoid breakdown of the factorization (see Section 3). Thus the incomplete factorization of $\bar{A} = \bar{S}Q^TAQ\bar{S} + \alpha I$ 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 based on a limited memory version of the left-looking approach by Tismenetsky [69] and Kaporin[41]. The basic scheme is based on a matrix factorization of the form

$$\bar{A} = (L + R)(L + R)^T - E, \tag{2.1}$$

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

$$E = RR^T.$$

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 [41], `HSL_MA28` optionally 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 (so that column j of L has at most $lszie + nz_j$ entries, where nz_j is the number of entries in column j of A) and the maximum number of entries in each column of R, respectively. Further details are given in [63, 64, 65].

We now consider the case where A is symmetric indefinite. In this case, (2.1) will be replaced by

$$\bar{A} = (L + R)D(L + R)^T - E, \tag{2.2}$$

where L and R are as before (with L now having unit diagonal entries), D is block diagonal with 1×1 and 2×2 blocks and E is of the form

$$E = RDR^T.$$

2.2. Pivoting strategies. In the positive definite case, pivoting is not used. Furthermore, for saddle-point systems, we are able to avoid pivoting by using an appropriately chosen constrained ordering (see [65] for details). However, in the indefinite case, pivoting is needed to try and maintain stability. We incorporate a number of pivoting strategies; namely, that of Bunch and Kaufman [12], Bunch tridiagonal pivoting [11], and the threshold variant of the Bunch and Kaufman approach introduced by Liu [46]. We also allow the option of pivoting down the diagonal with breakdown considered to have occurred if the pivot candidate is too small, in which case a shift is applied. This option may be suitable if the matrix is (close to) positive definite or if preprocessing of the matrix has ensured large diagonal entries.

The partial pivoting strategy by Bunch and Kaufman has been widely used for factorizing (dense) symmetric indefinite matrices using 1×1 and 2×2 pivots. The algorithm for choosing the i-th pivot may be outlined as follows.

Bunch-Kaufman partial pivot strategy (1977)

```
$\alpha_p := (1 + \sqrt{17})/8 \approx 0.64$
Find $j \neq i$ such that $|a_{ji}| = \max\{|a_{ki}|, k \neq i\} =: \lambda$
if $|a_{ii}| \geq \alpha_p \, |\lambda|$ then
    use $a_{ii}$ as a $1 \times 1$ pivot
else
    $\sigma := \max\{|a_{kj}|, k \neq j\}$
    if $|a_{ii}| \, \sigma \geq \alpha_p \, \lambda^2$ then
        use $a_{ii}$ as a $1 \times 1$ pivot
    else if $|a_{jj}| \geq \alpha_p \, \sigma$ then
        use $a_{jj}$ as a $1 \times 1$ pivot
    else
        use $\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix}$ as a $2 \times 2$ pivot
    end
end
```

The parameter $\alpha_p := (1 + \sqrt{17})/8$ is chosen to minimize the bound on element growth. This pivoting strategy was incorporated into the incomplete factorizations of Li and Saad [44] and Greif et al. [31].

Although Bunch-Kaufman pivoting is popular for dense systems, it is generally viewed as not suitable for sparse systems. In the sparse case, if $j - i$ is large then the choice of j as a 1×1 pivot (or (i, j) as a 2×2 pivot) at the i-th step can seriously adversely effect the sparsity of the computed factors. Consequently, in the sparse case, it is common to use a threshold-based strategy that limits the search so that $j - i$ cannot be large (see, for example, [21, 56]). Here we use the localized threshold pivoting as proposed by Liu [46]. This is a simple modification of the Bunch-Kaufman strategy. Recall that, at each stage, λ is the largest off-diagonal entry in the first column of the reduced matrix. In the Bunch-Kaufman approach, λ is always chosen to form the 2×2 pivot. Liu proposed imposing a threshold condition on the selection of 2×2 pivots. A sparsity threshold τ in the range $(0, 1]$ must be a chosen and, in place of the largest entry in the first column, the entry of smallest row index in the first column that satisfies the threshold condition is chosen. The strategy for picking the i-th pivot is as follows.

The optimal value of α_p now depends on the value of τ. Liu [46] shows that, given τ, the optimal α_p satisfies a cubic equation and is monotonically increasing with respect to τ. The threshold strategy has the advantage of being more localized than the Bunch-Kaufman approach, possibly at the expense of stability.

```
Liu threshold partial pivoting (1987)
Choose a sparsity threshold value $\tau$ such that $0 < \tau \leq 1$
Find $j \neq i$ such that $|a_{ji}| = \max\{|a_{ki}|, k \neq i\} =: \lambda$
Find $s \neq i$ such that $s = \min\{k, k \neq i, |a_{ki}| \geq \tau\, |a_{\lambda i}|\}$.
if $|a_{ii}| \geq \alpha_p\, |\lambda|$ then
    use $a_{ii}$ as a $1 \times 1$ pivot
else
    $\sigma := \max\{|a_{ks}|, k \neq s\}$
    if $|a_{ii}|\, \sigma \geq \alpha_p\, \lambda^2$ then
        use $a_{ii}$ as a $1 \times 1$ pivot
    else if $|a_{ss}| \geq \alpha_p\, \sigma$ then
        use $a_{ss}$ as a $1 \times 1$ pivot
    else
        use $\begin{pmatrix} a_{ii} & a_{si} \\ a_{si} & a_{ss} \end{pmatrix}$ as a $2 \times 2$ pivot
    end
end
```

A tridiagonal matrix is, of course, a special case of a sparse matrix and for such matrices it is possible to use a yet more localized pivoting strategy. In 1974, Bunch [11] introduced the following scheme for the i-th pivot of a symmetric tridiagonal matrix (for simplicity, we assume here that the subdiagonal entries are nonzero).

```
Bunch tridiagonal pivoting strategy (1974)
$\alpha_p = (\sqrt{5} - 1)/2 \approx 0.62$
$\sigma$:= the entry of largest absolute in the initial matrix
if $|a_{ii}|\, \sigma \geq \alpha_p\, |a_{i+1,i1}|^2$ then
    use $a_{ii}$ as a $1 \times 1$ pivot
else
    use $\begin{pmatrix} a_{ii} & a_{i,i+1} \\ a_{i+1,i} & a_{i+1,i+1} \end{pmatrix}$ as a $2 \times 2$ pivot
end
```

This strategy was used by Hagemann and Schenk [33] in combination with a symmetric version of a maximum weighted matching ordering for the incomplete factorization of sparse symmetric indefinite problems. The matching-based ordering is used to a priori permute large entries to the subdiagonal positions; the hope is that these can be used to provide stable 2×2 pivots (for the sparse direct case, see [24, 39]). Hagemann and Schenk employ a local perturbation if the candidate pivot is too small to be inverted. Advantages of this approach are that no entries beyond the diagonal and subdiagonal are searched and no swapping of rows/columns is needed during the factorization, which offers potential time savings and significantly simplifies the software development. Hagemann and Schenk report encouraging results for some saddle-point systems arising from interior-point problems; results for general indefinite systems are less positive.

3. The use of shifts and multipliers. We start by recalling the use of shifts in the case where A is symmetric and positive definite. The problem of breakdown during an incomplete Cholesky factorization because of the occurrence of zero or negative pivots is well known. Arbitrarily small pivots can also lead to unstable and therefore inaccurate factorizations. In the late 1970s, Kershaw [43] proposed locally replacing non-positive diagonal entries by a small positive number; the hope being that if only a few entries need to be replaced, the resulting factorization will still yield an acceptable preconditioner. This idea helped popularise incomplete factorizations, but ad hoc local perturbations with no relation to the overall matrix can lead to large growth in the entries and to unstable preconditioners. Thus a more commonly used approach is the one originally suggested by Manteuffel [49] that involves factorizing the diagonally shifted matrix $A + \alpha I$ for some positive α. Note that provided α is large enough, the incomplete factorization of the shifted positive definite matrix always exists, although currently the only way to find a suitable global shift is essentially by trial-and-error (see [45] and [64] for a discussion of the strategies used for increasing and decreasing shifts in the IC factorization codes ICFS and `HSL_MI28`, respectively). The power of the technique was perhaps somewhat underestimated, probably because of the need to restart the factorization with each new shift and the fact that the shifted matrix may be far from the original one if a suitable small value of α cannot be found. However, the recent results presented in [63, 64] illustrate how effective it can be in increasing the stability of the factorization for positive definite problems coming from a wide range of practical applications. Moreover, by monitoring the diagonal entries as the factorization progresses, the extra work that results from restarting the factorization (possibly multiple times) is generally not found to be prohibitive.

A relatively simple generalization of the global shift strategy was used by Scott and Tůma [65] for symmetric indefinite systems in saddle-point form. They employ two shifts: a positive shift for the $(1, 1)$ block and a negative shift for the $(2, 2)$ block. The shifts can always be chosen such that a signed incomplete Cholesky factorization exists. Such a shifting strategy is closely connected to the regularization techniques used by the numerical optimization community (see, for example, Saunders and Tomlin [59]). The results given in [65] illustrate the use of two shifts can lead to effective preconditioners.

Shifts have also been used in the construction of incomplete factorizations of general sparse matrices. In particular, the package IFPACK offers level-based ILUT(k) preconditioners and suggests the use of a global shift if the computed factors are found to be unstable. IFPACK factorizes the scaled and shifted matrix B, whose entries are given by

$$b_{ij} = \begin{cases} a_{ij} & \text{if } i \neq j \\ \rho\, a_{ii} + sgn(a_{ii})\, \alpha & \text{if } i = j, \end{cases} \tag{3.1}$$

where α and ρ are positive parameters that must be determined by the user. The documentation for the code suggests a trial-and-error method for selecting suitable values.

In this paper, our interest is in the general symmetric indefinite case. Our incomplete factorization approach uses both 1×1 and 2×2 pivots combined with a shift strategy to compute a factorization of the form LDL^T, where D is block diagonal, with blocks of order 1 and 2. If at some stage of the factorization a suitable pivot cannot found, the matrix A is shifted and the factorization is restarted. There are two possibilities. We could retain the sequence of pivots that was chosen before breakdown occurred; this would potentially have the advantage of saving work in the pivot searches. Alternatively, because of pivoting, the elimination order generated after the shift strategy is applied may be different. For implementation reasons, we adopt the latter approach and recompute from scratch the pivot sequence for each new choice of shift(s). Of course, there is no guarantee that the new choice will not lead to breakdown at an earlier step than previously. However, it is always possible to choose shifts such that the diagonal blocks of the shifted matrix are sufficiently diagonally dominant for the factorization to be breakdown free. Of course, such a choice may lead to an inaccurate factorization of the unshifted matrix and hence to a poor quality preconditioner.

We assume that the chosen 2×2 pivots satisfy the following condition.

Assumption 3.1. *2×2 pivots are chosen such that the (positive) product of their off-diagonal entries is larger than the product of the magnitudes of their diagonal entries.* Note that the Bunch Kaufman pivoting satisfies this assumption. Namely, the 2×2 pivot (i, j) is chosen only if both $|a_{jj}| < \alpha_p \sigma$ and $|a_{ii}|\sigma < \alpha_p \lambda^2$. Putting these together, we have $|a_{ii}||a_{jj}| < \alpha_p^2 \lambda^2 < \lambda^2$, from which it follows that the assumption is satisfied.

In general, the stability of matrix factorizations is reflected in a quantity called the *growth factor*, which measures possible growth of the entries in the factors. Standard references on symmetric indefinite factorizations such as [9, 13] derive formulae for the growth factor based on the global behavior of the pivoted factorization, see also the recent discussion of the growth factor in [18]. The derivation of the growth factor employs such quantities as the maximum magnitudes of the diagonal and off-diagonal entries of A. In this paper, we use shifts and so we are interested in the growth caused by shifting pivots and, in particular, the case of 2×2 pivots.

DEFINITION 3.1. *Consider a 1×1 or 2×2 (nonsingular) pivot P used in an indefinite factorization. The value of the entry of largest absolute value in P^{-1} is called the* local growth factor. If $p \in R \rightarrow p \in R \setminus \{0\}$, the local growth factor is just $\theta = 1/|p|$. Consider now a 2×2 pivot P given by

$$P = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, \tag{3.2}$$

with inverse

$$P^{-1} = \frac{1}{ac - b^2} \begin{pmatrix} c & -b \\ -b & a \end{pmatrix}. \tag{3.3}$$

Then the corresponding local growth factor is

$$\theta = \frac{max(|a|, |b|, |c|)}{|ac - b^2|}.$$

In the following, we describe how the local growth factor is influenced by a shift and, in particular, how θ can be decreased by the use of a shift and also possibly a positive multiplier $\rho \geq 1.0$. The case of a 1×1 pivot $P = p$, as in (3.1), is simple. We use a positive shift α and a multiplier $\rho \geq 1.0$

$$p = \rho\, p + sgn(p)\, \alpha$$

and the local growth factor is reduced. The case of 2×2 pivots needs to be considered more carefully. First, Assumption 3.1 implies that to increase the diagonal dominance of (3.2) it is important to modify its off-diagonal entries. One possible approach to shifting a 2×2 pivot is to derive the shift from a comparison with the situation of two consecutive 1×1 pivots. While the analysis of diagonal pivoting strategies starting with the seminal contribution of Bunch and Parlett [13] based on upper bounds of the magnitudes of entries, often uses this approach, local determination of the shift with possible restarts does not allow this. For example, if we consider a 2×2 pivot with zeros on the diagonal, the first column does not update the second. For this reason, in the following we discuss shifts for such pivots independently and separately from the motivation for 1×1 pivots. For simplicity of explanation, we follow [21] and distinguish three basic types of 2×2 pivots. Throughout our discussion, the shift α is positive and the multiplier ρ is at least 1.0.

3.1. Oxo pivots. An *oxo* pivot is a 2×2 pivot of the form

$$P_{oxo} = \begin{pmatrix} 0 & b \\ b & 0 \end{pmatrix}.$$

Taking into account (3.3), this pivot clearly implies the local growth factor

$$\theta_{oxo} = \frac{|b|}{b^2} = \frac{1}{|b|}.$$

Thus stability of P_{oxo} is increased by increasing the absolute value of its off diagonal entry b. The modified oxo pivot is

$$P_{oxo}^{+} = \begin{pmatrix} 0 & \rho\, b + sgn(b)\, \alpha \\ \rho\, b + sgn(b)\, \alpha & 0 \end{pmatrix},$$

which has a smaller local growth factor equal to

$$\theta_{oxo}^{+} = \frac{|\rho\, b + sgn(b)\, \alpha|}{(\rho\, b + sgn(b)\, \alpha)^2} = \frac{1}{|\rho\, b + sgn(b)\, \alpha|}.$$

3.2. Tile pivots. Tile pivots have one non zero diagonal entry and one diagonal entry equal to zero, that is,

$$P_{tile} = \begin{pmatrix} a & b \\ b & 0 \end{pmatrix}.$$

To improve pivot stability by shifting the entries of P_{tile}, first consider the effect of modifying the off diagonal entry b to $\rho\, b + sgn(b)\,\alpha$. Clearly, the local growth factor (see (3.3)) after the modification is

$$\theta^+_{tile} = \frac{max(|\rho\, b + \mathrm{sgn}(b)\,\alpha|, |a|)}{(\rho\, b + sgn(b)\,\alpha)^2} = max\left(\frac{1}{|\rho\, b + sgn(b)\,\alpha|}, \frac{|a|}{(\rho\, b + sgn(b)\,\alpha)^2}\right). \tag{3.4}$$

It is easy to see that the following holds.

OBSERVATION 3.1. *The local growth factor (3.4) is a non increasing function of $\alpha > 0$ and $\rho \geq 1.0$.*

This means that if the shift $\alpha > 0$ is applied in this way, the local growth factor is either unchanged or is reduced, which is our goal. However, if $|a| > |\rho\, b + \mathrm{sgn}(b)\,\alpha|$, θ^+_{tile} can be reduced further by decreasing $|a|$. In addition, the Euclidean condition number decreases by decreasing $|a|$, as we can see from the following result.

LEMMA 3.2. *The Euclidean condition number of P_{tile} is an increasing function of $|a| > 0$.*

Proof: The characteristic equation of the eigenvalue problem connected to P_{tile} is $-\lambda(a-\lambda) - b^2 = 0$. Therefore, its condition number is given by

$$\kappa_{tile} = \frac{|a + \sqrt{a^2 + 4b^2}|}{|a - \sqrt{a^2 + 4b^2}|}.$$

Since $a \neq 0$ this can be rewritten as

$$\kappa_{tile} = \frac{1 + \sqrt{1 + 4(b/a)^2}}{|1 - \sqrt{1 + 4(b/a)^2}|}.$$

Hence

$$\kappa_{tile} = \frac{2 + 4(b/a)^2 + 2\sqrt{1 + 4(b/a)^2}}{4(b/a)^2} = 1 + 1/(2(b/a)^2) + (\sqrt{1/(b/a)^4 + 4/(b/a)^2})/2.$$

The last expression clearly reveals that κ_{tile} can be decreased if $|a|$ is decreased. Note that the form of the expression for κ_{tile} is such as to allow its easy application later. □

Lemma 3.2 and (3.4) imply the practical modification procedure. By modifying the off diagonal entries described above the local growth factor is made smaller. Then, if we have in addition

$$|a| > |\rho\, b + \mathrm{sgn}(b)\,\alpha|$$

we can further decrease the local growth factor and, simultaneously, the condition number of P_{tile}. It does not have a sense to decrease the magnitude of the nonzero diagonal entry further under $|\rho\, b + \mathrm{sgn}(b)\,\alpha|$ because the monitored quantity, the local growth factor, cannot be decreased any more. Summarizing this, we replace a by $a - sgn(a)\,\delta$ with

$$\delta = min\,(\alpha, |a| - |\rho\, b + sgn(b)\,\alpha|) > 0. \tag{3.5}$$

The modified tile pivot can be then written as

$$P^+_{tile} = \begin{pmatrix} a - sgn(a)\delta & \rho\, b + sgn(b)\,\alpha \\ \rho\, b + sgn(b)\,\alpha & 0 \end{pmatrix}$$

with the local growth factor

$$\theta^+_{tile} = \frac{max\,(|\rho\, b + sgn(b)\,\alpha|\,, |a - sgn(a)\,\delta|)}{(\rho\, b + sgn(b)\,\alpha)^2}.$$

3.3. Full 2×2 pivots. Finally, consider a full 2×2 pivot

$$P_{full} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

with $a, b, c \neq 0$. Recall that we assume that $b^2 > |ac|$ (Assumption 3.1). Without loss of generality, we also assume $|a| \geq |c| > 0$ since otherwise we can exchange these diagonal entries.

As in the previous case, the stability of P_{full} is increased by modifying the off diagonal entry b to become $\rho\, b + sgn(b)\alpha$. The local growth factor of the pivot with modified off diagonal entries is then

$$\theta^{+}_{full} = \frac{max(|\rho\, b + sgn(b)\,\alpha|, |a|)}{(|\rho\, b + sgn(b)\,\alpha)^2}. \tag{3.6}$$

Similarly to above, we have the following observation.

Observation 3.2. *The local growth factor (3.6) is a decreasing function of $\alpha > 0$ and $\rho \geq 1.0$.*

Since Assumption 3.1 implies that $b^2 > |ac|$ we have $|c| < |\rho\, b + sgn(b)\,\alpha|$ for $|a| > |\rho\, b + sgn(b)\,\alpha|$. As in the case of the tile pivots, the local growth factor can be still decreased further by decreasing $|a|$ as shown in the following result.

Lemma 3.3. *Assume $b^2 - |ac| > 0$, $|a| \geq |c| > 0$. Then*

$$q = \frac{b^2 - ac}{(a+c)^2} \tag{3.7}$$

is a decreasing function of $|a|$.

Proof: Consider first the case $a > 0$. The first derivative of q with respect to a is then

$$-\frac{(c^2 + 2b^2 - ac)}{(a+c)^3}. \tag{3.8}$$

For all a and c with $b^2 - |ac| > 0$, $c^2 + 2b^2 - ac$ is positive and the derivative is negative. Thus if a and c are both positive or if $c < 0$ and $a > 0$ then since $a + c > 0$, the derivative is negative as in the first case. Summarizing this, we conclude q is a decreasing function of $a > 0$. If a is negative we will get the same conclusion considering the growth in the pivot $-P_{full}$ and decreasing the entry $-a > 0$. We can conclude that q is a decreasing function of $|a|$. $\square$

Lemma 3.3 can be then used to prove similar property for the condition number of the full pivot as we have for tile pivots.

Theorem 3.4. *The Euclidean condition number of P_{full} under the assumptions from Lemma 3.3 is an increasing function of $|a| > 0$.*

Proof: The characteristic equation of the eigenvalue problem connected to P_{full} is $(c-\lambda)(a-\lambda) - b^2 = 0$. Therefore, its condition number is

$$\kappa_{full} = \frac{1 + \sqrt{1 + 4\frac{b^2-ac}{(a+c)^2}}}{\left|1 - \sqrt{1 + 4\frac{b^2-ac}{(a+c)^2}}\right|}.$$

As in Lemma 3.2, this can be rewritten as

$$1 + 1/\left(2\frac{b^2 - ac}{(a+c)^2}\right) + \left(\sqrt{1/\left(\frac{b^2 - ac}{(a+c)^2}\right)^2 + 4/\left(\frac{b^2 - ac}{(a+c)^2}\right)}\right)/2$$

The result from Lemma 3.3 completes the proof. $\square$

As in the case of tile pivots, Lemma 3.4 implies that for $|a| > |\rho\, b + sgn(b)\,\alpha|$ we can simultaneously decrease both the local growth factor and the condition number of P_{full}. In practice, we should again limit size of the modification of a by the use of $a - sgn(a)\,\delta$ with δ given by (3.5). The modified full 2×2 pivot can then be written as

$$P^{+}_{full} = \begin{pmatrix} a - sgn(a)\,\delta & \rho\, b + sgn(b)\,\alpha \\ \rho\, b + sgn(b)\,\alpha & c \end{pmatrix}$$

with the local growth factor

$$\theta^{+}_{full} = \frac{max\,(|\rho\, b + sgn(b)\,\alpha|\,, |a - sgn(a)\,\delta|\,, |c|)}{(\rho\, b + sgn(b)\,\alpha)^2}.$$

4. Instability growth.

A well-studied problem in sparse symmetric indefinite factorizations as well as in sparse nonsymmetric factorizations is the growth in the size of the entries of the factors. As already discussed, the usual approach in *complete* factorizations is to employ a pivoting scheme so that the entries in the factors are bounded. In Sections 2 and 3, we considered trying to limit growth in the factors through the use of both pivoting and global shifts and multipliers. Nevertheless, even with well bounded entries in L and D, the triangular solves can be highly unstable. A sign of unstable triangular solves is when $||L^{-1}||$ is very large and, unfortunately, this can occur without the presence of small pivots. The problem was discussed by Chow and Saad [14], who proposed checking three quantities: the size of the inverse of the smallest pivot, the size of the largest entry in the computed factors and a statistic they call *condest*. This is defined to be

$$condest = ||(LDL^T)^{-1}e||_\infty,$$

where $e = (1, ..., 1)^T$ is the vector of all ones. It measures the stability of the triangular solves and is also a lower bound for $||(LDL^T)^{-1}||_\infty$ and indicates a relation between unstable triangular solves and poorly conditioned L factor. IFPACK [59] also uses *condest* and, as already discussed (equation (3.1)) proposes *a priori* diagonal perturbations if the condition estimate is larger than machine precision.

In this section, we propose monitoring possible instability as the factorization proceeds. In contrast to the nonsymmetric factored approximate inverse factorization of Bollhöfer and Saad [7, 8], we do not use dropping or pivoting information on the approximate inverse. Instead, we are concerned with the stability of the triangular solves when the factors are used in the subsequent iterative method. Our proposed approach enables us to monitor stability at each stage of the factorization, allowing us to restart the factorization with a new shift or multiplier or with different parameter settings as soon as possible instability is detected. Furthermore, as in [14], our monitoring of stability can serve as a measure of the usefulness of the computed factors. As a consequence, we are able to obtain more robust incomplete symmetric indefinite factorizations that provide higher quality preconditioners. This is illustrated in Section 5.

Recall two facts related to our indefinite factorization algorithm. First, the (block) diagonal entries may be modified (when used as pivots or block pivots). As observed previously, there is extensive experimental evidence in the positive definite case and for saddle point systems [63, 65] that this can be efficient and more robust than using ad hoc local modifications. Second, in the positive definite case, we observed in [64] that the memory used for the preconditioner L, which is parametrized by *lsize* can, to some extent, be replaced by the intermediate memory R, which is parametrized by *rsize* (that is, *lsize* can be reduced and *rsize* increased without significantly effecting the quality of L as a preconditioner). As R is used for the computation of L but is then discarded, this can lead to a sparser preconditioner that is less expensive to apply. Our experimental results (see Section 5) illustrate that monitoring stability as the factorization proceeds can provide an indication as to whether R should be discarded or whether retaining it (or part of it) could lead to a higher quality preconditioner. The approach we propose is based on these observations and motivated also by Chow and Saad's use of *condest*.

The factors L and R are computed in ν steps, where ν is equal to n minus the number of 2×2 pivots. Let L_k, R_k and D_k of dimension n_k $(k = 1, \dots, \nu)$ (with $n_\nu \equiv n$) denote the leading principal submatrices of L, R and D, respectively. Further, let p_k be the dimension of the k-th pivot ($p_k = 1$ or 2). Also let δ_k denote the k-th diagonal pivot. We then introduce the following definition of instability growth.

DEFINITION 4.1. *Let the instability growth factor g_k at the k-th factorization step be the entry of largest absolute value in the vector $D_k^{-1}L_k^{-1}e_k$, where e_k is the n_k dimensional vector of all ones. The instability growth factor at the final (ν-th) step is denoted by g.*

To compute g_k, for $k = 2, \dots, \nu$, consider the factor L_k in the bordered form

$$L_k = \begin{pmatrix} L_{k-1} & \\ l_k & I_k \end{pmatrix},$$

where I_k is the identity matrix of dimension p_k, l_k is the $p_k \times n_{k-1}$ block of the off diagonal entries in the k-th (block) row of L, and L_1 is the identity matrix I_1. g_k can be computed as follows.

```
$v_1 = I_1$
$g_1 = ||D_1^{-1}||_\infty$
for $k = 2, \dots, \nu$
    $v_k = L_k^{-1}e_k = \begin{pmatrix} L_{k-1} & \\ l_k & I_k \end{pmatrix}^{-1} e_k \equiv \begin{pmatrix} L_{k-1}^{-1} & \\ -l_kL_{k-1}^{-1} & I_k \end{pmatrix} e_k \equiv \begin{pmatrix} v_{k-1} \\ e_{p_k} - l_kv_{k-1} \end{pmatrix}$
    $g_k = ||D_k^{-1}v_k||_\infty = max(g_{k-1}, ||\delta_k^{-1}(v_k)_{n_{k-1}+1:n_k}||_\infty)$
end
```

This computation requires us to store the current instability factor and a vector v that is computed incrementally. (Note that its k-th update has dimension n_k.) That is, for a particular $2 \le k \le \nu$, we update positions $n_{k-1}+1 : n_k$ of v_k using

$$(v_k)_{n_{k-1}+1:n_k} = e_{p_k} - l_kv_{k-1}. \tag{4.1}$$

The instability growth factor g_k can be used to monitor stability at each step of the factorization and to control when a shift and restart are needed. However, we have observed experimentally that, for hard indefinite systems, adding the computed R to L can reduce g for the resulting L. Thus if g is large, this suggests it can be advantageous to add R to L (that this, it can lead to a more efficient preconditioner, albeit one that requires more memory than L alone). Alternatively, we could selectively add to L entries of R that could decrease g. Our experiments in Section 5 illustrate that if this partial addition of R with L leads to a significant improvement in the stability of the factors measured by *condest*, then in terms of the efficiency of the preconditioner (see Section 5, equation (5.1) for a definition of the efficiency of a preconditioner), the improvement for $L + R$ is typically even better.

Thus we can regard R as a source of additional entries that can potentially improve the final incomplete factor. We propose a possible simple strategy based on this idea. Observe that since R is used in the updates in the same way as L, it is sufficient at each step of the factorization to flag the entries of R that may be moved to L and then the actual merging of these entries into L can be done once the factorization is complete.

For $k = 2, \dots, \nu$, let the factor R_k be written in the bordered form

$$R_k = \begin{pmatrix} R_{k-1} & \\ r_k & 0_k \end{pmatrix},$$

where 0_k is the null matrix of dimension p_k, r_k is the $p_k \times n_{k-1}$ block of the off diagonal entries in the k-th (block) row of R, and R_1 is the null matrix 0_1. Let λ_k denote a set of column indices from $j \in \{1, \dots, n_{k-1}\}$ for which $(l_k)_{*j}$ is nonzero. Then equation (4.1) can be written as

$$(v_k)_{n_{k-1}+1:n_k} = e_{p_k} - \sum_{j\in\lambda_k} (l_k)_{*j}(v_{k-1})_j.$$

Our goal is to find a subset ρ_k of indices $j \in \{1,\ldots,n_{k-1}\}$ for which $(r_k)_{*j}$ is nonzero and minimizes the sum

$$e_{p_k} - \sum_{j\in\lambda_k} (l_k)_{*j}(v_{k-1})_j - \sum_{j\in\rho_k} (r_k)_{*j}(v_{k-1})_j.$$

This sum is used to evaluate g_k after scaling by the corresponding diagonal entry/entries. This approximation problem is a special instance of the sparse approximation problem, in particular, selecting a few columns from a source matrix S such that their sum well approximates a target matrix T is a matrix generalization of the *subset sum selection problem* [15] in which the matrix has at most two rows. Let us denote by $\bar{l}_k$ the $p_k \times n_{k-1}$ block with the scaled entries $(\bar{l}_k)_{ij} = (l_k)_{ij}(v_{k-1})_j$, $i = 1,\ldots,p_k$, $j = 1,\ldots,n_{k-1}$. Further denote by $\bar{r}_k$ the $p_k \times n_{k-1}$ block with entries $(\bar{r}_k)_{ij} = (r_k)_{ij}(v_{k-1})_j$, $i = 1,\ldots,p_k$, $j = 1,\ldots,n_{k-1}$. The problem is then defined as follows.

PROBLEM 4.1. *For $k = 2,\ldots,n_\nu$, find a subset $\bar{\rho}_k$ of indices from the index set ρ_k that minimizes some norm of*

$$e_{p_k} - \sum_{j\in\lambda_k} \sum_{i=n_{k-1}+1}^{n_k} (\bar{l}_k)_{ij} - \sum_{j\in\bar{\rho}_k} \sum_{i=n_{k-1}+1}^{n_k} (\bar{r}_k)_{ij}$$

As we only seek to reduce the instability growth approximately, we propose a simple greedy strategy for the related subset sum selection problem. Instead of columns of r_k and l_k, we consider in the objective function their 1-norms. The algorithm is below.

```
do
    $sum = e_{p_k} - \sum_{j\in\lambda_k} (\bar{l}_k)_{*j}$
    $j_\rho = \mathrm{argmin}_{j\in\rho_k} ||sum + (\bar{r}_k)_{*j}||_1$
    $sum^+ = sum + (\bar{r}_k)_{*j_\rho}$
    if $||sum^+||_1 < ||sum||_1$ then
        $sum = sum^+$
        $\lambda_k = \lambda_k \cup \{j_\rho\}$
        $\rho_k = \rho_k \setminus \{j_\rho\}$
    else
        exit
    end
end do
```

Entries of R that correspond to the chosen set of indices form a factor Rs. Our experiments illustrate that, if *condest* for L is large, for $L + Rs$ it is typically smaller, leading to the iterative method converging in a smaller number of iterations.

5. Numerical experiments.

5.1. Test environment.

All the software used to obtained the results presented in this paper is written in Fortran and the gfortran Fortran compiler (version 4.8.2) with option -O3 is used. The implementations of the GMRES(1000) algorithm (with right preconditioning) [58] offered by the HSL routine `MI24` 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

$$\|A\hat{x} - b\|_2 \le 10^{-8}\|b\|_2,$$

where $\hat{x}$ is the computed solution. In addition, for each test we impose a limit of 2000 iterations. Following [62], in our experiments we define the *efficiency* of the preconditioner P to be

$$\textit{efficiency} = \textit{iter} \times nz(L), \tag{5.1}$$

where *iter* is the iteration count for $P = (LDL^T)^{-1}$. The lower the value of (5.1), the better the preconditioner. We also define the fill in the incomplete factor to be the ratio

$$fill_{IL} = (\text{number of entries in the incomplete factor})/nz(K), \tag{5.2}$$

Our test problems are real indefinite matrices taken from the University of Florida (UFL) Sparse Matrix Collection [17].

5.2. Results for interior-point optimization matrices. Our first set (Test Set 1) are interior-point optimization matrices that are of saddle point form, namely,

$$A = \begin{pmatrix} H & B^T \\ B & -C \end{pmatrix}. \tag{5.3}$$

For the problems in Test Set 1 (which are listed in Table 5.1), H is $n \times n$ symmetric positive definite, B is rectangular $m \times n$ ($m \leq n$), and $C = 10^{-8}I$ (where I is the identity matrix) is $m \times m$. These problems represent a subset of the largest c-xx problems in the GHS_indef and Schenk_IBMNA test sets (and include those from these sets that were used in [65]). Table 5.1 gives the order n of the $(1,1)$ block H, the order m of the $(2,2)$ block C and the number $nz(A)$ of entries in the lower triangular part of A. Here and elsewhere, we use the direct solver `HSL_MA97` [36] to compute the number of entries in the complete factor of A. For the `HSL_MA97` runs, we use the scaling from a symmetrized version of the package `MC64` [22, 23]. We remark that this scaling has been found to work well for direct solvers when used to solve "tough" indefinite systems [35, 39]. We use the nested dissection ordering and also a matching-based ordering computed using the HSL package `MC80`. We report $fill_L(ND)$ (respectively, $fill_L(match)$) to be the ratio of the number of entries in the factor for the default ordering (respectively, matching-based ordering) to $nz(A)$ (we also tried approximate minimum degree ordering but found that, for most of these examples, the resulting fill was greater than for nested dissection). These ratios are reported for later comparison with the fill for the incomplete factorizations. Note that the matching-based ordering leads to significantly denser factors.

Table 5.1

Interior-point test problems (Test Set 1). n and m denote the order of H and C (see (5.3)), $nz(A)$ is the number of entries in the lower triangular part of A, $fill_L(ND)$ (respectively, $fill_L(match)$) is the ratio of the number of entries in the complete factor of A for the nested dissection ordering (respectively, for the matching-based ordering) to $nz(A)$.

Identifier	n	m	$nz(A)$	$fill_L(ND)$	$fill_L(match)$
Schenk_IBMNA/c-54	17664	14129	208890	2.8	4.7
GHS_indef/c-55	19121	13659	218115	15.6	27.1
Schenk_IBMNA/c-56	19923	15987	208075	3.0	4.4
GHS_indef/c-59	23813	17469	260909	14.3	24.0
Schenk_IBMNA/c-61	25043	18575	176817	5.4	8.0
Schenk_IBMNA/c-62	25158	16573	300536	22.8	28.7
GHS_indef/c-63	25505	18729	239469	9.5	14.3
GHS_indef/c-68	36546	28264	315403	19.1	31.6
GHS_indef/c-69	38432	29026	345686	7.8	12.6
GHS_indef/c-70	39302	29622	363955	9.5	15.9
GHS_indef/c-71	44814	31824	468079	29.4	46.9
GHS_indef/c-72	47950	36114	395805	9.2	12.4
Schenk_IBMNA/c-73	86417	83005	724348	3.5	5.8
Schenk_IBMNA/c-big	201877	143364	1343050	30.4	45.3

For any sparse matrix factorization algorithm, the ordering of the matrix is important and the effects of sparse matrix orderings on the convergence of preconditioned Krylov subspace methods have been widely reported on in the literature (the survey by Benzi [4], for example, contains a large number of references and a brief summary discussion). In the positive definite case, we found that preordering using the profile reduction algorithm of Sloan [54, 66, 67] generally led to a high quality incomplete Cholesky factorization preconditioner [63]. To investigate whether the same conclusion is valid for the the c-xx problems, in Table 5.2, we compare the number of GMRES iterations required when the incomplete factorization is combined with different orderings.

Table 5.2
A comparisons of the number of iterations of GMRES(1000) required using different ordering algorithms for Test Set 1. AMD denote approximate minimum degree, ND denotes nested dissection, RCM denotes reverse Cuthill McKee and SL denotes the Sloan algorithm. For each problem, the lowest number of iterations is in bold. – denotes failure to converge within 2000 iterations.

Identifier						Matching-based			
	None	AMD	ND	RCM	SL	AMD	ND	RCM	SL
Schenk_IBMNA/c-54	1956	–	–	–	–	306	**266**	294	–
GHS_indef/c-55	–	78	135	160	560	30	**26**	31	42
Schenk_IBMNA/c-56	174	156	1440	447	828	78	**66**	87	312
GHS_indef/c-58	727	35	47	41	54	28	**26**	41	52
GHS_indef/c-59	73	41	98	135	174	28	**27**	35	88
Schenk_IBMNA/c-61	–	15	39	19	47	**9**	11	13	16
Schenk_IBMNA/c-62	–	426	100	435	473	**26**	31	47	119
GHS_indef/c-63	–	66	72	129	294	20	21	**18**	95
GHS_indef/c-68	711	102	161	81	69	**6**	**6**	31	22
GHS_indef/c-69	198	22	74	38	87	**13**	15	22	40
GHS_indef/c-70	1532	46	115	73	319	**16**	17	21	27
GHS_indef/c-71	655	152	98	82	246	**21**	**21**	25	42
GHS_indef/c-72	164	29	125	38	126	**14**	**14**	19	30
Schenk_IBMNA/c-73	–	–	–	200	–	**11**	22	142	–
Schenk_IBMNA/c-big	–	13	68	740	851	**9**	11	11	16

The orderings considered are approximate minimum degree (AMD), nested dissection (ND), reverse Cuthill McKee (RCM) and the Sloan algorithm (SL). These are applied to A and are also each combined with a matching-based ordering (that is, each is applied to the compressed graph that is obtained using a matching algorithm; see [39] for details). These results are for $lsize = rsize = 10$, Bunch Kaufman pivoting and `MC64` scaling. Here and throughout the remainder of this subsection, no shifts are used and entries of R are not used to improve L. Note that, as $lsize$ and $rsize$ are held constant, $fill_{IL}$ (5.2) is similar for each ordering. The results clearly indicate that for this class of problems, using a matching-based ordering leads to higher quality preconditioners. Moreover, employing either AMD or ND combined with matching generally gives the best ordering in terms of the GMRES iteration count. In the remainder of this section, we use AMD in combination with matching.

We now compare the different pivoting strategies discussed in Section 2.2; results are given in Tables 5.3 and 5.4. $lsize$ and $rsize$ are again set to 10. We see that, for most of these problems, having performed the matching ordering, pivoting is not needed (that is, pivoting does not improve the preconditioner). Moreover, the fill is significantly less than for the direct solver (recall Table 5.1). The timings in Table 5.4 are for solving for a single right-hand side and the time T_f to compute the incomplete factorization includes the time for preordering and scaling the matrix. The results illustrate the overheads that pivoting incurs. In particular, we see that Bunch Kaufman can be much more expensive than using the threshold pivoting of Liu, which is only slightly more expensive than the simple tridiagonal strategy,

Tables 5.5 and 5.6 report results for different choices of $lsize$ and $rsize$ (with Liu threshold pivoting, `MC64` scaling and AMD combined with the matching ordering). We see that using a non zero R can substantially improve the quality of the preconditioner albeit at additional cost in the time taken to compute the incomplete factorization. Of course, if more than one system is to be solved using the preconditioner, the cost of the incomplete factorization can be amortized over the number of right-hand sides and it is then clearly advantageous to use $rsize > 0$ and, if there is sufficient memory available, to increase both $lsize$ and $rsize$. For comparison, in Table 5.6, we also report times for the direct solver `HSL_MA97`. Since `HSL_MA97` is an OpenMP code, the reported timings are for running the code in parallel

Table 5.3

A comparison of the pivoting strategies for Test Set 1. BK denotes Bunch Kaufman, Btri denotes Bunch tridiagonal pivoting and Liu denotes the threshold partial pivoting strategy of Liu (with threshold $\tau = 0.1$). Iterations is the number of iterations of GMRES(1000) performed. – denotes failure to converge within 2000 iterations.

Identifier	$fill_{IL}$				Iterations			
	None	BK	Btri	Liu	None	BK	Btri	Liu
Schenk_IBMNA/c-54	2.10	1.79	2.28	1.91	–	306	–	195
GHS_indef/c-55	2.19	2.20	2.22	2.19	31	30	33	30
Schenk_IBMNA/c-56	2.30	2.33	2.40	2.30	70	78	42	28
GHS_indef/c-58	1.84	2.08	2.10	1.85	57	28	30	58
GHS_indef/c-59	2.19	2.33	2.34	2.19	23	28	27	23
Schenk_IBMNA/c-61	2.58	2.84	2.87	2.58	8	9	9	8
Schenk_IBMNA/c-62	2.11	2.12	2.14	2.11	30	26	26	29
GHS_indef/c-63	2.41	2.51	2.52	2.41	19	20	19	19
GHS_indef/c-68	2.15	2.26	2.26	2.15	6	6	6	6
GHS_indef/c-69	2.35	2.54	2.56	2.35	13	13	13	13
GHS_indef/c-70	2.35	2.48	2.49	2.35	16	16	14	16
GHS_indef/c-71	2.24	2.36	2.38	2.24	22	21	20	22
GHS_indef/c-72	2.35	2.59	2.65	2.37	16	14	18	17
Schenk_IBMNA/c-73	1.25	1.42	1.45	1.25	16	11	10	10
Schenk_IBMNA/c-big	2.65	2.65	2.65	2.65	9	9	9	9

Table 5.4

Timings (in seconds) for Test Set 1 using the pivoting strategies. BK denotes Bunch Kaufman, Btri denotes Bunch tridiagonal pivoting and Liu denotes the threshold partial pivoting strategy of Liu (with threshold $\tau = 0.1$). T_f and T_g are the times to compute the factorization and run GMRES(1000), respectively. The total time T is the sum of T_f and T_g. The fastest total time is in bold. – denotes failure to converge within 2000 iterations.

Identifier	None			BK			Btri			Liu		
	T_f	T_g	T	T_f	T_g	T	T_f	T_g	T	T_f	T_g	T
Schenk_IBMNA/c-54	–	–	–	0.76	2.10	2.86	–	–	–	0.46	0.99	**1.45**
GHS_indef/c-55	0.33	0.08	**0.41**	0.47	0.08	0.54	0.32	0.09	**0.41**	0.34	0.08	0.42
Schenk_IBMNA/c-56	0.23	0.23	0.46	0.43	0.26	0.69	0.23	0.13	0.36	0.25	0.08	**0.32**
GHS_indef/c-58	0.38	0.21	0.59	0.50	0.09	0.59	0.33	0.10	**0.44**	0.41	0.21	0.62
GHS_indef/c-59	0.35	0.07	**0.43**	0.44	0.09	0.54	0.34	0.09	**0.43**	0.35	0.08	**0.43**
Schenk_IBMNA/c-61	0.20	0.02	**0.22**	0.30	0.02	0.32	0.21	0.02	0.24	0.20	0.02	**0.22**
Schenk_IBMNA/c-62	0.42	0.11	**0.53**	0.55	0.09	0.64	0.45	0.10	0.56	0.46	0.12	0.58
GHS_indef/c-63	0.38	0.06	0.44	0.49	0.07	0.56	0.34	0.06	**0.40**	0.38	0.06	0.44
GHS_indef/c-68	0.77	0.03	**0.79**	1.41	0.03	1.44	0.78	0.03	0.81	0.78	0.03	0.80
GHS_indef/c-69	0.53	0.07	0.60	0.70	0.06	0.76	0.49	0.06	**0.55**	0.56	0.06	0.62
GHS_indef/c-70	0.60	0.08	0.68	0.74	0.08	0.82	0.57	0.07	**0.64**	0.62	0.08	0.70
GHS_indef/c-71	0.81	0.15	0.96	1.26	0.14	1.40	0.72	0.14	**0.86**	0.81	0.14	0.95
GHS_indef/c-72	0.66	0.10	0.76	1.20	0.09	1.29	0.58	0.13	**0.72**	0.65	0.10	0.76
Schenk_IBMNA/c-73	0.39	0.13	0.51	10.62	0.09	10.71	0.41	0.08	**0.49**	0.41	0.08	**0.49**
Schenk_IBMNA/c-big	2.68	0.26	**2.93**	7.00	0.25	7.26	2.73	0.25	2.97	2.75	0.25	3.00

on 8 threads and are again for a single right-hand side. We observe that for these test problems the (serial) incomplete factorization can be significantly faster than the direct solver, particularly if $lsize$ and $rsize$ are relatively small (but the direct solver obtains a much more accurate solution, with a scaled residual of 10^{-16}).

Finally, we provide a comparison with the signed incomplete Cholesky factorization approach of [65] that is implemented within `HSL_MI30` and with SYM-ILDL. The results in Table 5.7 use $lsize = rsize = 30$. `HSL_MI30` uses initial shifts $\alpha_{in}(1:2) = 0.01$, Sloan ordering, and equilibration scaling (see [65] for details of these settings). For SYM-ILDL, we use equilibration scaling, AMD ordering and the parameter settings $fill = 12.0$ and $tol = 0.003$ (again, see [65]). With these choices, the level of fill is similar to that for our incomplete LDL^T factorization and the signed IC factorization. We see that our incomplete LDL^T factorization generally results in the best preconditioner (in terms of fill, iteration count and efficiency).

5.3. Results for power system optimization matrices. Test Set 2 are from the TSOPF test set of the UFL Collection and are transient stability-constrained optimal power flow problems. They are of the saddle point structure (5.3) but, in this case, H is not positive definite and $C = 0$, the $m \times m$

Table 5.5
GMRES(1000) convergence results for Test Set 1 using different choices of lsize and rsize.

Identifier	$lsize = rsize = 10$			$lsize = 10$, $rsize = 0$			$lsize = rsize = 30$		
	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters
Schenk_IBMNA/c-54	1.91	7.9×10^{7}	195	1.27	1.3×10^{8}	488	2.65	4.8×10^{7}	86
GHS_indef/c-55	2.19	1.4×10^{7}	30	1.56	1.3×10^{7}	38	3.67	8.0×10^{6}	10
Schenk_IBMNA/c-56	2.30	1.3×10^{7}	28	1.41	5.7×10^{7}	196	2.70	5.1×10^{6}	9
GHS_indef/c-58	1.85	3.2×10^{7}	58	1.41	2.8×10^{7}	67	2.79	7.4×10^{6}	9
GHS_indef/c-59	2.19	1.3×10^{7}	23	1.53	3.6×10^{7}	90	3.68	9.6×10^{6}	10
Schenk_IBMNA/c-61	2.58	3.6×10^{6}	8	1.64	5.5×10^{6}	19	2.79	1.5×10^{6}	3
Schenk_IBMNA/c-62	2.11	1.8×10^{7}	29	1.58	4.2×10^{7}	88	3.41	1.3×10^{7}	13
GHS_indef/c-63	2.41	1.1×10^{7}	19	1.57	2.8×10^{7}	74	3.90	6.5×10^{6}	7
GHS_indef/c-68	2.15	4.1×10^{6}	6	1.44	3.6×10^{6}	8	3.31	3.1×10^{6}	3
GHS_indef/c-69	2.35	1.1×10^{7}	13	1.54	1.5×10^{7}	29	3.44	4.8×10^{6}	4
GHS_indef/c-70	2.35	1.4×10^{7}	16	1.57	3.9×10^{7}	68	3.60	1.0×10^{7}	8
GHS_indef/c-71	2.24	2.3×10^{7}	22	1.59	3.0×10^{7}	41	3.64	1.9×10^{7}	11
GHS_indef/c-72	2.37	1.6×10^{7}	17	1.57	8.1×10^{7}	130	3.08	4.9×10^{6}	4
Schenk_IBMNA/c-73	1.25	9.0×10^{6}	10	1.06	4.3×10^{7}	56	1.25	8.1×10^{6}	9
Schenk_IBMNA/c-big	2.65	3.2×10^{7}	9	1.68	4.1×10^{7}	18	3.05	2.1×10^{7}	5

Table 5.6
Timings (in seconds) for Test Set 1 using different choices of lsize and rsize and for the direct solver `HSL_MA97`. *T_f and T_g are the times to compute the factorization and run GMRES(1000), respectively. The total time T is the sum of T_f and T_g. The fastest total time is in bold.*

Identifier	$lsize = rsize = 10$			$lsize = 10$, $rsize = 0$			$lsize = rsize = 30$			`HSL_MA97`
	T_f	T_g	T	T_f	T_g	T	T_f	T_g	T	
Schenk_IBMNA/c-54	0.45	0.99	1.44	0.26	4.71	4.98	0.49	0.31	0.80	**0.42**
GHS_indef/c-55	0.35	0.08	0.43	0.10	0.09	**0.20**	1.30	0.03	1.33	0.54
Schenk_IBMNA/c-56	0.24	0.07	0.31	0.11	1.04	1.14	0.26	0.03	**0.28**	**0.28**
GHS_indef/c-58	0.38	0.21	0.59	0.13	0.25	0.39	1.20	0.03	1.23	**0.48**
GHS_indef/c-59	0.37	0.08	**0.44**	0.12	0.37	0.49	1.68	0.04	1.72	0.71
Schenk_IBMNA/c-61	0.21	0.02	0.23	0.07	0.05	**0.12**	0.20	0.01	0.21	0.26
Schenk_IBMNA/c-62	0.46	0.12	**0.57**	0.18	0.38	**0.57**	1.70	0.06	1.76	0.98
GHS_indef/c-63	0.36	0.06	0.43	0.10	0.28	**0.38**	0.87	0.03	0.90	0.39
GHS_indef/c-68	0.79	0.03	0.81	0.14	0.03	**0.17**	2.03	0.02	2.05	1.51
GHS_indef/c-69	0.53	0.06	0.59	0.15	0.14	**0.29**	0.86	0.03	0.88	0.53
GHS_indef/c-70	0.62	0.08	0.70	0.16	0.40	**0.56**	1.52	0.05	1.57	0.64
GHS_indef/c-71	0.82	0.15	0.97	0.24	0.26	**0.50**	4.08	0.09	4.17	2.74
GHS_indef/c-72	0.65	0.11	0.76	0.17	1.25	1.42	0.81	0.03	0.84	**0.66**
Schenk_IBMNA/c-73	0.41	0.08	0.49	0.33	0.61	0.94	**0.43**	0.07	0.50	2.87
Schenk_IBMNA/c-big	2.75	0.25	3.00	0.76	0.44	**1.20**	2.93	0.16	3.08	11.8

Table 5.7
GMRES(1000) convergence results for Incomplete LDL^T and signed incomplete Cholesky applied to Test Set 1 (lsize = rsize = 30). Results are also given for SYM-ILDL. – denotes failure to converge within 2000 iterations.

Identifier	Incomplete LDL^T			`HSL_MI30`			SYM-ILDL		
	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters
Schenk_IBMNA/c-54	2.65	4.8×10^{7}	86	3.75	3.3×10^{7}	41	6.87	–	–
GHS_indef/c-55	3.67	8.0×10^{6}	10	3.37	3.0×10^{7}	41	4.35	7.4×10^{7}	78
Schenk_IBMNA/c-56	2.70	5.1×10^{6}	9	4.08	1.0×10^{8}	123	6.22	–	–
GHS_indef/c-58	2.79	7.4×10^{6}	9	2.78	5.6×10^{7}	68	2.60	1.4×10^{8}	183
GHS_indef/c-59	3.68	9.6×10^{6}	10	3.67	3.9×10^{7}	41	4.33	1.1×10^{8}	97
Schenk_IBMNA/c-61	2.79	1.5×10^{6}	3	3.94	2.5×10^{7}	36	3.38	9.6×10^{6}	16
Schenk_IBMNA/c-62	3.41	1.3×10^{7}	13	3.40	6.2×10^{7}	61	3.23	3.3×10^{7}	34
GHS_indef/c-63	3.90	6.5×10^{6}	7	3.79	4.3×10^{7}	47	4.46	6.5×10^{7}	61
GHS_indef/c-68	3.31	3.1×10^{6}	3	4.06	1.8×10^{7}	14	4.31	8.3×10^{7}	61
GHS_indef/c-69	3.44	4.8×10^{6}	4	4.00	4.1×10^{7}	30	3.81	5.5×10^{7}	42
GHS_indef/c-70	3.60	1.0×10^{7}	8	3.88	6.3×10^{7}	45	3.61	2.9×10^{7}	22
GHS_indef/c-71	3.64	1.9×10^{7}	11	3.51	8.7×10^{7}	53	4.03	7.0×10^{7}	37
GHS_indef/c-72	3.08	4.9×10^{6}	4	3.91	5.7×10^{7}	37	4.02	9.7×10^{7}	61
Schenk_IBMNA/c-73	1.25	8.1×10^{6}	9	4.88	8.6×10^{8}	244	3.84	3.5×10^{9}	1267
Schenk_IBMNA/c-big	3.05	2.1×10^{7}	5	4.44	3.3×10^{8}	56	4.47	–	–

null matrix. Note that, as H is not positive definite, a signed incomplete Cholesky factorization is not recommended. The problems are listed in Table 5.8. HSL_MA97 is run with nested dissection ordering and MC64 scaling. Note that, although of structural full rank, these problems are not all of full numerical rank.

Table 5.8
TSOPF test problems (Test Set 2).

Identifier	n	m	$nz(A)$	HSL_MA97	
				$fill_L$	Time
TSOPF_FS_b9_c6	7230	7224	75801	6.01	0.06
TSOPF_FS_b39_c7	14118	14098	368599	5.48	0.24
TSOPF_FS_b39_c19	38118	38098	998359	5.57	0.61
TSOPF_FS_b39_c30	60118	60098	1575639	5.76	1.20
TSOPF_FS_b162_c1	5424	5374	305732	5.60	0.16
TSOPF_FS_b162_c3	15424	15374	904612	5.90	0.55
TSOPF_FS_b162_c4	20424	20374	1204322	5.99	0.71

In Table 5.9 results are given for $lsize = rsize = 30$ using AMD combined with a matching-based ordering, (symmetric) MC64 scaling, and Bunch tridiagonal pivoting. The first two sets of three columns are results for using L as the preconditioner while the last two sets of three columns are for using $L + R$ as the preconditioner. We present results for initial shifts $\alpha = 0$ and 0.01. With $\alpha = 0$, for a number of the problems (TSOPF_FS_b39_c7, TSOPF_FS_b39_c19 and TSOPF_FS_b162_c1) the factorization suffers breakdown. For this test set, we found that using a shift that was smaller than 0.01 generally led to a poorer quality preconditioner. We see that using a non zero initial shift and/or using $L + R$ yields a reduction in the number of iterations although, because the fill is greater for $L + R$, *efficiency* may not be improved. We also ran using $L + Rs$ as the preconditioner, where Rs includes only selected entries of R, as discussed in Section 4. We found that the results (in terms of the fill and the number of iterations) lie, as expected, between those for L and those for $L + R$. For instance, for problem TSOPF_FS_b39_c30, using $L + Rs$ with $\alpha = 0.01$, we obtain $fill_{IL} = 3.14$ and an iteration count of 673.

Timings are given in Table 5.10 for $\alpha = 0.01$. Comparing columns 4 and 7 , we see that adding R into L incurs a small overhead in the factorization time but this is more than offset by the resulting reduction in the GMRES time. However, the incomplete factorization times are, for these examples, not competitive with the those of the (parallel) direct solver (Table 5.8).

Table 5.9
Results for $lsize = rsize = 30$ with and without an initial non-zero shift, using L and $L+R$ as the preconditioner. BD denotes factorization breakdown. – indicates condest greater than 10^{16}.

Identifier	L, $\alpha = 0.0$			L, $\alpha = 0.01$			$L+R$, $\alpha = 0.0$			$L+R$, $\alpha = 0.01$		
	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters
TSOPF_FS_b9_c6	2.78	6.3×10^6	30	2.74	8.3×10^6	40	3.05	1.8×10^6	8	3.01	5.3×10^6	23
TSOPF_FS_b39_c7	BD	BD	BD	2.31	2.5×10^8	296	BD	BD	BD	3.51	2.5×10^8	194
TSOPF_FS_b39_c19	BD	BD	BD	2.32	1.1×10^9	463	BD	BD	BD	3.50	9.2×10^8	263
TSOPF_FS_b39_c30	2.30	–	–	2.46	2.8×10^9	732	4.05	2.6×10^8	40	4.04	2.5×10^9	386
TSOPF_FS_b162_c1	BD	BD	BD	1.77	9.8×10^7	181	BD	BD	BD	2.83	1.1×10^8	125
TSOPF_FS_b162_c3	1.68	1.2×10^9	790	1.67	2.1×10^8	141	2.70	5.8×10^8	237	2.69	2.1×10^8	87
TSOPF_FS_b162_c4	1.67	8.0×10^8	397	1.67	3.0×10^8	149	2.69	6.6×10^8	205	2.69	3.0×10^8	93

So far, in all the reported results, we have used a fixed global multiplier $\rho = 1.0$. In Table 5.11, results are given for $\rho = 1.1$ with shift $\alpha = 0.0$ and 0.01 (here L is used as the preconditioner). As our analysis predicted, using a multiplier $\rho > 1.0$ reduces *condest* and this can lead to a substantial reduction in the iterations needed for convergence (problems TSOPF_FS_b162_c3 and TSOPF_FS_b162_c4). *condest* is reduced further by using a positive shift. Note, however, that using $\rho > 1.0$ can lead to an increase in the iteration count since the computed L is now for a perturbed system (this is illustrated by problem TSOPF_FS_b9_c6).

TABLE 5.10

Timings (in seconds) for Test Set 2 using L and $L+R$ as the preconditioner ($\alpha = 0.01$). T_f and T_g are the times to compute the factorization and run GMRES(1000), respectively. The total time T is the sum of T_f and T_g.

Identifier	L			$L+R$		
	T_f	T_g	T	T_f	T_g	T
TSOPF_FS_b9_c6	0.03	0.03	0.06	0.03	0.02	0.05
TSOPF_FS_b39_c7	0.25	2.14	2.39	0.34	1.32	1.66
TSOPF_FS_b39_c19	0.72	12.2	12.9	0.85	5.55	6.41
TSOPF_FS_b39_c30	1.56	43.4	45.0	1.88	16.8	18.7
TSOPF_FS_b162_c1	0.23	0.38	0.61	0.25	0.33	0.58
TSOPF_FS_b162_c3	0.82	1.10	1.91	0.86	0.71	1.56
TSOPF_FS_b162_c4	1.05	1.42	2.47	1.19	1.03	2.22

TABLE 5.11

Results for lsize = rsize = 30 with and without an initial non-zero shift α and non-unit multiplier ρ, using L as the preconditioner. BD denotes factorization breakdown. – indicates convergence not achieved.

Identifier	$\rho = 1.0,\ \alpha = 0.0$				$\rho = 1.1,\ \alpha = 0.0$				$\rho = 1.1,\ \alpha = 0.01$			
	$fill_{IL}$	*efficiency*	iters	*condest*	$fill_{IL}$	*efficiency*	iters	*condest*	$fill_{IL}$	*efficiency*	iters	*condest*
TSOPF_FS_b9_c6	2.78	6.3×10^6	30	5.77×10^9	2.65	2.8×10^7	137	1.55×10^5	2.66	3.0×10^7	147	9.45×10^4
TSOPF_FS_b39_c7	BD	BD	BD	BD	2.58	9.2×10^8	965	5.06×10^8	2.27	4.6×10^8	553	3.51×10^5
TSOPF_FS_b39_c19	BD	BD	BD	BD	2.59	–	–	3.12×10^8	2.28	2.1×10^9	906	3.57×10^5
TSOPF_FS_b39_c30	2.30	–	–	2.33×10^{20}	2.20	–	–	4.15×10^9	2.28	7.1×10^9	1991	5.85×10^5
TSOPF_FS_b162_c1	BD	BD	BD	BD	BD	BD	BD	BD	1.75	1.4×10^8	267	7.69×10^5
TSOPF_FS_b162_c3	1.68	1.2×10^9	790	2.91×10^{13}	1.66	3.0×10^8	197	2.03×10^7	1.65	2.9×10^8	191	1.60×10^7
TSOPF_FS_b162_c4	1.67	8.0×10^8	397	6.34×10^{11}	1.65	4.1×10^8	206	1.92×10^7	1.65	3.9×10^8	198	1.51×10^7

5.4. Results for density functional theory matrices. So far, we have reported results for indefinite systems that have a saddle point structure for which using a matching-based ordering is advantageous. We now consider symmetric indefinite problems that are from symmetric eigenvalue problems in density functional theory calculations; these problems do not have a saddle-point structure. Test Set 3 is summarised in Table 5.12; these problems are from the PARSEC group of the UFL Collection. Using a direct solver is very expensive for some of these problems as the amount of fill is high and there are a number that `HSL_MA97` was unable to solve because of insufficient memory.

TABLE 5.12

PARSEC test problems (Test Set 3). – denotes the factorization failed because insufficient memory (in these cases, $fill_L$ is the predicted fill returned by the analyse phase on the basis of the sparsity pattern).

Identifier	n	$nz(A)$	`HSL_MA97`	
			$fill_L$	Time
CO	221119	3943588	495	–
Ga10As10H30	113081	3114357	216	400
Ga19As19H42	133123	4508981	179	532
Ga3As3H12	61349	3016148	80	62.7
Ga41As41H72	268096	9378286	275	–
GaAsH6	61349	1721579	136	48.8
Ge87H76	112985	4002590	160	327
Ge99H100	112985	4282190	153	527
H2O	67024	1141880	198	48.6
Si10H16	17077	446500	70	5.52
Si34H36	97569	2626974	185	231
Si41Ge41H72	185639	7598452	186	–
Si5H12	19896	379247	119	8.99
Si87H76	240369	5451000	376	–
SiO	33401	675528	131	15.8
SiO2	155331	5719417	181	–

Results using initial shifts $\alpha = 0$ and 0.01 are given in Tables 5.13 and 5.14. In these experiments, we use Bunch Kaufman pivoting, Sloan ordering and `MC64` scaling. We see that, as expected, if *condest* is large, there is no convergence but that using $\alpha = 0.01$ can substantially reduce *condest* and improve

Table 5.13

Results for lsize = rsize = 10 with and without an initial non-zero shift, using L and $L+R$ as the preconditioner. – indicates convergence not achieved within 2000 iterations.

Identifier	L, $\alpha = 0.0$			L, $\alpha = 0.01$			$L+R$, $\alpha = 0.0$			$L+R$, $\alpha = 0.01$		
	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters	$fill_{IL}$	*efficiency*	iters
CO	1.54	4.1×10^8	68	1.54	4.1×10^8	68	2.11	4.6×10^8	56	2.10	5.0×10^8	60
Ga10As10H30	1.24	2.7×10^9	691	1.21	9.7×10^8	258	1.61	1.3×10^9	255	1.57	9.7×10^8	199
Ga19As19H42	1.16	–	–	1.11	5.4×10^9	1083	1.45	4.9×10^9	746	1.40	2.5×10^9	390
Ga3As3H12	0.85	2.1×10^8	81	0.83	2.4×10^8	94	1.05	2.4×10^8	75	1.04	2.8×10^8	90
Ga41As41H72	1.18	–	–	1.11	7.3×10^9	704	1.47	–	–	1.39	7.3×10^9	557
GaAsH6	1.14	1.1×10^8	56	1.14	1.7×10^8	89	1.50	1.3×10^8	52	1.49	2.0×10^8	77
Ge87H76	1.12	–	–	1.08	–	–	1.40	–	–	1.36	5.0×10^9	923
Ge99H100	1.11	–	–	1.07	–	–	1.37	–	–	1.34	–	-1
H2O	1.56	5.3×10^7	30	1.56	5.9×10^7	33	2.15	6.6×10^7	27	2.14	7.3×10^7	30
Si10H16	1.23	9.1×10^7	166	1.21	2.5×10^8	460	1.62	8.3×10^7	115	1.59	1.1×10^8	156
Si34H36	1.22	–	–	1.17	9.4×10^8	307	1.59	1.8×10^9	439	1.54	1.0×10^9	246
Si41Ge41H72	1.01	–	–	0.91	4.0×10^9	574	1.25	–	–	1.15	4.2×10^9	483
Si5H12	1.46	2.2×10^7	40	1.45	2.4×10^7	43	1.98	2.5×10^7	33	1.97	2.8×10^7	37
Si87H76	1.39	–	–	1.37	–	–	1.83	–	–	1.82	–	–
SiO	1.37	4.0×10^7	43	1.36	3.7×10^7	40	1.87	4.4×10^7	35	1.86	4.4×10^7	35
SiO2	0.77	1.9×10^8	43	0.76	2.3×10^8	53	1.04	2.3×10^8	39	1.03	2.9×10^8	49

Table 5.14

condest for the incomplete factorization preconditioner computed using lsize = rsize = 10 with and without an initial non-zero shift; L and $L+R$ are used. The timings in the final column are total times (in seconds) for $L+R$ with $\alpha = 0.01$. – denotes convergence not achieved within 2000 iterations.

Identifier	*condest*(L)		*condest*$(L+R)$		
	$\alpha = 0.0$	$\alpha = 0.01$	$\alpha = 0.0$	$\alpha = 0.01$	Time
CO	2.43×10^2	8.59	2.08×10^1	9.25	5.52
Ga10As10H30	5.22×10^6	6.16×10^4	2.49×10^2	3.09×10^2	8.36
Ga19As19H42	1.43×10^{16}	1.39×10^8	2.96×10^4	2.35×10^4	23.4
Ga3As3H12	2.19×10^1	1.42×10^2	2.30×10^1	1.65×10^2	3.08
Ga41As41H72	1.12×10^{48}	4.35×10^3	6.46×10^6	1.13×10^2	80.0
GaAsH6	6.67×10^1	4.24×10^3	8.25×10^1	2.14×10^2	2.01
Ge87H76	8.76×10^{27}	2.80×10^{16}	4.21×10^5	5.58×10^3	74.1
Ge99H100	3.74×10^{36}	2.28×10^{21}	2.27×10^6	1.19×10^4	–
H2O	4.94	1.28×10^1	8.59	1.44×10^1	1.15
Si10H16	5.21×10^5	7.03×10^7	4.27×10^2	1.26×10^2	0.90
Si34H36	9.64×10^{11}	1.67×10^2	2.84×10^4	6.09×10^1	8.71
Si41Ge41H72	5.35×10^{35}	3.50×10^3	8.84×10^5	1.34×10^2	45.0
Si5H12	3.24×10^1	1.11×10^1	3.28×10^1	1.09×10^1	0.33
Si87H76	1.30×10^{38}	9.78×10^{30}	8.48×10^8	1.93×10^8	–
SiO	7.39×10^1	6.24×10^1	6.91×10^1	2.16×10^1	0.57
SiO2	4.17×10^1	9.38	4.83×10^1	9.96	4.35

the quality of the preconditioner. Moreover, if *condest*(L) is large, then in these tests *condest*$(L+R)$ is significantly smaller and using $L+R$ again improves preconditioner quality. As in the previous section, if $L+Rs$ is used as the preconditioner, then the fill and the number of iterations lie between those for L and those for $L+R$. Note that a modest value of *condest* does not guarantee convergence. Comparing the timings reported in the last column of Table 5.14 with those for the direct solver given in Table 5.12, we see that, not only is the iterative method successful in solving more problems, but the time taken is substantially less than is required by `HSL_MA97` (although when it is successful, `HSL_MA97` again computes a solution of higher accuracy). For the PARSEC test problems, we thus have a potentially attractive alternative to a sparse direct solver that requires substantially less memory and computational time.

5.5. Growth instability monitoring and enriching L by entries from R. The following experiments demonstrate the relationship between preconditioning using L, $L+R$ and also $L+Rs$. We first consider problem Ga3As3H12 from the PARSEC test set; we use the same settings as in Section 5.4 with initial shift $\alpha = 0.01$ and multiplier $\rho = 1.0$. As already observed, the incomplete factorization of this matrix provides a high quality preconditioner, even for small values of *lsize* and *rsize*. To explore what happens as the ratio of $rsize$ to $lsize$ decreases, we fix $rsize = 5$ and let $lsize$ vary from 1 to 45; iteration counts, *efficiency*, and *condest* are reported in Figure 5.1. We see that, as expected, as $lsize$ increases, R

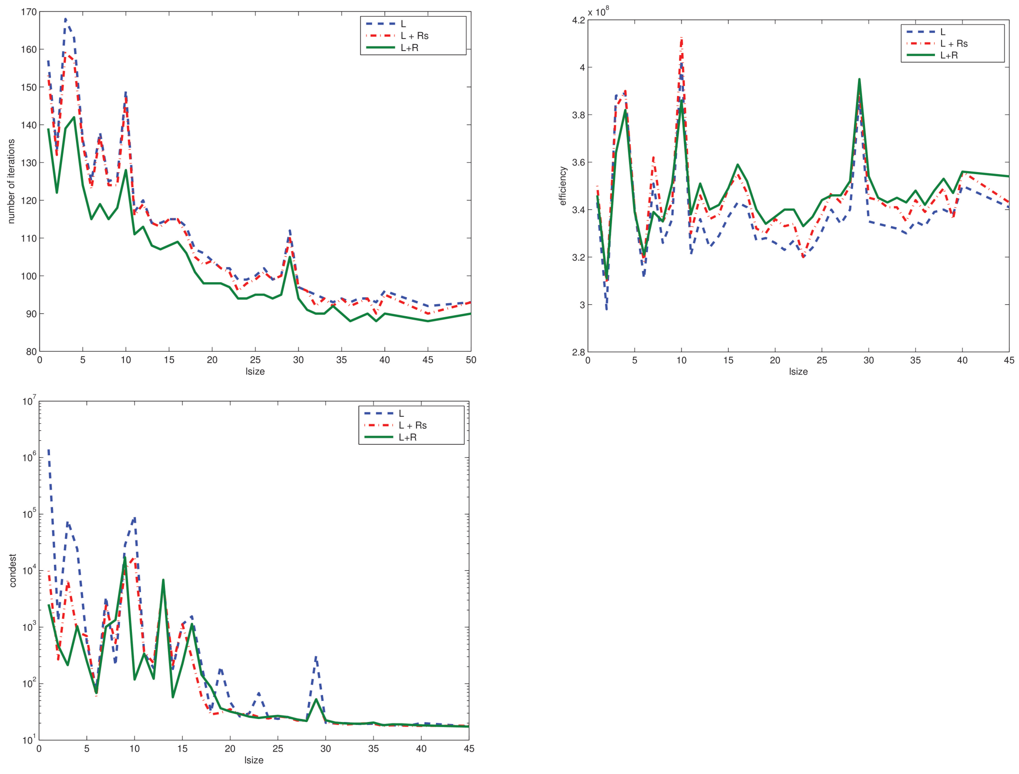

Fig. 5.1. *Iteration counts (top left), efficiency (top right) and condest (bottom left) for problem Ga3As3H12 with* $rsize = 5$ *and* $lsize$ *varying.*

becomes less significant and the differences in the statistics for L, $L + Rs$ and $L + R$ reduce. Moreover, the prediction of the instability growth that is used to construct Rs works well, with $condest(L + Rs)$ generally lying between $condest(L)$ and $condest(L + R)$. In terms of *efficiency* (recalling that the smaller the value of *efficiency* the better), we see using L is generally best as the modest reduction in the iteration counts for $L + Rs$ and $L + R$ are unable to offset the increase in factor size.

We next consider problem Si10H16 from the PARSEC set. In Table 5.15, we report the iteration count and *condest* for a range of values of $rsize$ with $lsize = 10$ and $\alpha = 0.0$, $\rho = 1.0$. As $lsize$ is fixed, $nz(L)$ is essentially the same for all choices of $rsize$ while $nz(L + R) \simeq nz(L) + rsize * n$. This example illustrates that using intermediate memory R in the construction of L does not guarantee to improve the quality of L as a preconditioner but that stability in this case is recovered using $L + R$. If we set $rsize = 0$ and

Table 5.15
Iteration counts and condest for problem Si10H16 using $lsize = 10$ *and* $rsize$ *varying.*

$rsize$	L		$L + R$	
	iters	$condest$	iters	$condest$
0	123	7.14×10^2		
10	166	5.21×10^5	115	4.27×10^2
20	391	3.66×10^7	118	1.41×10^3
30	524	2.35×10^8	114	3.08×10^2
40	532	2.14×10^8	99	1.62×10^3
50	429	2.72×10^7	86	2.73×10^2

increase $lsize$, for a given value $lsize = l_0$, the quality of the resulting L_0 as a preconditioner is, as we would expect, similar to that of $L + R$ computed using $lsize = 10$ and $rsize = l_0 - 10$. The advantage of the latter is that if the prediction of the instability growth indicates there is no instability, the sparser L can be used without including entries of R.

Figure 5.2 reports iteration counts and *efficiency* for problem Schenk_IBMNA/c-56 from Test Set 1. For this example, as $lsize = rsize$ is increased, $condest$ slowly varies from about 10^6 to 10^4 and this is reflected in the value of the instability growth estimate that is computed inside our code. A modest value of the instability growth allows us to choose between using L or $L+R$ as the preconditioner, depending on whether we need the preconditioner to be as sparse as possible and/or whether it is important to minimise the iteration count. For this problem, using $L + Rs$ does not offer advantages over using L.

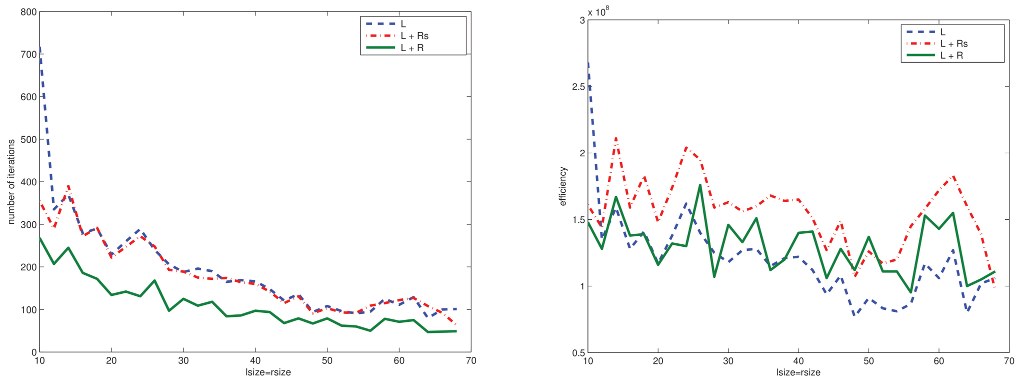

FIG. 5.2. *Iteration counts (left) and efficiency (right) for problem Schenk_IBMNA/c-56 with lsize = rsize varying.*

Our final experiment considers problem CO from the PARSEC group. In order to emphasize further the importance of monitoring instability growth and the potential advantage of employing $L + R$ as the preconditioner, we use Liu threshold partial pivoting with a threshold of 0.1. As illustrated in Table 5.4, this leads to faster computation of the incomplete factorization compared to using Bunch Kaufman pivoting but it can result in instabilities, especially if L or $L+Rs$ is used as the preconditioner. Iteration counts and *efficiency* are reported in Figure 5.3 for $lsize = rsize$ in the range 10 to 68. In this case, we see that using

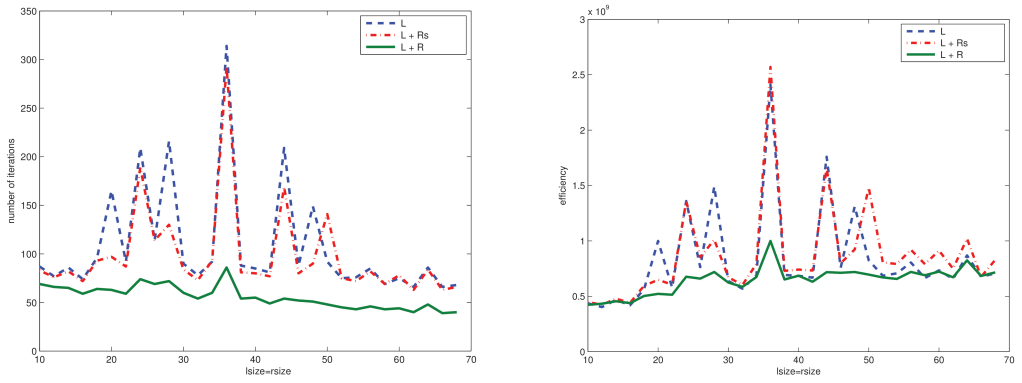

FIG. 5.3. *Iteration counts (left) and efficiency (right) for problem PARSEC/CO with lsize = rsize varying.*

$L + R$ as the preconditioner stabilizes the relaxed Liu pivoting. The specific choices of $lsize = rsize$ that give the peaks in Figure 5.3 correspond to significantly higher values of $condest(L)$ than of $condest(L+R)$. We recommend that if $condest(L)$ is much larger than $condest(L+R)$, $L+R$ should be the preconditioner of choice.

6. Concluding remarks. In this paper, we have focused on the development of incomplete factorization preconditioners for symmetric indefinite sparse linear systems. We have employed a limited memory approach that has proved robust for positive definite systems. We have incorporated numerical pivoting to prevent the entries of the factors from becoming large and have proposed new ideas to prevent instability growth and to monitor stability as the factorization proceeds.

In our experience, the problems that prove difficult to solve with our incomplete factorization approach are generally those for which the triangular solves are unstable, as indicated by a large value of *condest*. It is possible to improve the stability of the triangular solves using pivot modifications and we have discussed how to do this for both 1×1 pivots and for the different types of 2×2 pivots. Our numerical experiments have shown that pivot modifications can be very effective in reducing *condest* and improving preconditioner quality. However, if the shift α and/or multiplier ρ needs to be large to prevent instability then the computed incomplete factorization is inaccurate and convergence of the iterative solver may not be achieved. Moreover, increasing the amount of fill is not always sufficient to obtain an accurate and stable factorization.

To successfully solve a wide range of problems, our software incorporates a number of different pivoting options as well as different scalings. In addition, the user can control the choice of shift and multiplier as well as the amount of memory available for L and R. Our tests have shown that using intermediate memory ($R \neq 0$) can be beneficial but this is not guaranteed. Furthermore, using $L+R$ (or, in some cases, the sparser $L + Rs$) can provide a better preconditioner than L that is much less sensitive to the choice of $lsize$ and $rsize$. The difficulty for a given problem is determining which options should be selected and choosing appropriate values for the parameters. For saddle-point problems, we recommend using a matching-based ordering and scaling combined with using intermediate memory; with these choices L has been seen to provide a high-quality preconditioner. For some classes of problems, the incomplete factorization preconditioner combined with GMRES can compete with a state-of-the-art parallel direct solver and can solve problems for which the direct solver fails because of its memory requirements.

REFERENCES

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30:381–388, 2004.

[2] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. on Matrix Analysis and Applications*, 23:15–41, 2001.

[3] C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM J. on Matrix Analysis and Applications*, 20(2):513–561, 1999.

[4] M. Benzi. Preconditioning techniques for large linear systems: a survey. *J. of Computational Physics*, 182(2):418–477, 2002.

[5] M. Benzi, G.H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

[6] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. on Scientific Computing*, 22(4):1333–1353, 2000.

[7] M. Bollhöfer and Y. Saad. A factored approximate inverse preconditioner with pivoting. *SIAM J. on Matrix Analysis and Applications*, 23(3):692–705, 2001/02.

[8] M. Bollhöfer and Y. Saad. On the relations between *ILU*s and factored approximate inverses. *SIAM J. on Matrix Analysis and Applications*, 24(1):219–237, 2002.

[9] J. R. Bunch. Analysis of the diagonal pivoting method. *SIAM J. on Numerical Analysis*, 8:656–680, 1971.

[10] J. R. Bunch. Equilibration of symmetric matrices in the max-norm. *Journal of the ACM*, 18:566–572, 1971.

[11] J. R. Bunch. Partial pivoting strategies for symmetric matrices. *SIAM Journal on Numerical Analysis*, 11:521–528, 1974.

[12] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31:162–179, 1977.

[13] J. R. Bunch and B. Parlett. Direct methods for solving symmetric indefinite systems of linear systems. *SIAM J. on Numerical Analysis*, 8:639–655, 1971.

[14] E. Chow and Y. Saad. Experimental study of *ILU* preconditioners for indefinite matrices. *J. of Computational and Applied Mathematics*, 86(2):387–414, 1997.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press, Cambridge, MA, third edition, 2009.

[16] E. H. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings* 24^{th} *National Conference of the ACM*, pages 157–172. ACM Press, 1969.

[17] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1), 2011.

[18] A. Druinsky and S. Toledo. The growth-factor bound for the Bunch-Kaufman factorization is tight. *SIAM J. Matrix Anal. Appl.*, 32(3):928–937, 2011.

[19] I. S. Duff. MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144, 2004.

[20] I. S. Duff and J. R. Gilbert. Maximum-weighted matching and block pivoting for symmetric indefinite systems. In *Abstract book of Householder Symposium XV*, pages 73–75. Peebles, Scotland, 2002.

[21] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. Factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis*, 11:181–204, 1991.

[22] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. on Matrix Analysis and Applications*, 22:973–996, 2001.

[23] I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. on Matrix Analysis and Applications*, 27:313–340, 2005.

[24] I.S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. on Matrix Analysis and Applications*, 27:313 – 340, 2005.

[25] P. E. Gill, W. Murray, D. B. Ponceleón, and M. A. Saunders. Preconditioners for indefinite systems arising in optimization. *SIAM J. on Matrix Analysis and Applications*, 13(1):292–311, 1992.

[26] P. E. Gill, M. A. Saunders, and J. R. Shinnerl. On the stability of Cholesky factorization for symmetric quasidefinite systems. *SIAM J. on Matrix Analysis and Applications*, 17(1):35–46, 1996.

[27] G. H. Golub and C. F. Van Loan. Unsymmetric positive definite linear systems. *Linear Algebra and its Applications*, 28:85–97, 1979.

[28] N. I. M. Gould, Y. Hu, and J. A. Scott. A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations. *ACM Transactions on Mathematical Software*, 33(2), 2007.

[29] N. I. M. Gould, D. Orban, and T. Rees. Projected krylov methods for saddle-point systems. Technical Report RAL-P-2013-006, Rutherford Appleton Laboratory, 2013.

[30] C. Greif. Preconditioners for linear systems arising from interior-point methods. Presentation at the International Conference On Preconditioning Techniques for Scientific and Industrial Applications, The University of Oxford, 2013.

[31] 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.

[32] A. Gupta, M. Joshi, and V. Kumar. Wssmp: A high-performance serial and parallel sparse linear solver. Technical Report RC 22038 (98932), IBM T.J. Watson Reserach Center, 2001.

[33] M. Hagemann and O. Schenk. Weighted matchings for preconditioning symmetric indefinite linear systems. *SIAM J. on Scientific Computing*, 28(2):403–420, 2006.

[34] J. D. Hogg, E. Ovtchinnikov, and J. A. Scott. A sparse symmetric indefinite direct solver for GPU architectures. Technical Report RAL-P-2014-006, Rutherford Appleton Laboratory, 2014.

[35] 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, 2008.

[36] 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, 2011.

[37] J. D. Hogg and J. A. Scott. New parallel sparse direct solvers for multicore architectures. *Algorithms*, 6:702–725, 2013. Special issue: Algorithms for Multi Core Parallel Computation.

[38] J. D. Hogg and J. A. Scott. Optimal weighted matchings for rank-deficient sparse matrices. *SIAM J. on Matrix Analysis and Applications*, 34:1431–1447, 2013. DOI 10.1137/120884262.

[39] J. D. Hogg and J. A. Scott. Pivoting strategies for tough sparse indefinite systems. *ACM Transactions on Mathematical Software*, 40, 2013. Article 4, 19 pages.

[40] HSL. A collection of Fortran codes for large-scale scientific computation, 2013. `http://www.hsl.rl.ac.uk`.

[41] I. E. Kaporin. High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$ decomposition. *Numerical Linear Algebra with Applications*, 5:483–509, 1998.

[42] C. Keller, N. I. M. Gould, and A. J. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM J. on Matrix Analysis and Applications*, 21(4):1300–1317, 2000.

[43] D. S. Kershaw. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *J. of Computational Physics*, 26:43–65, 1978.

[44] N. Li and Y. Saad. Crout versions of ILU factorization with pivoting for sparse symmetric matrices. *Electronic Transactions on Numerical Analysis*, 20:75–85, 2005.

[45] C.-J. Lin and J. J. Moré. Incomplete Cholesky factorizations with limited memory. *SIAM J. on Scientific Computing*, 21(1):24–45, 1999.

[46] J.-W. H. Liu. A partial pivoting strategy for sparse symmetric matrix decomposition. *ACM Trans. Math. Software*, 13(2):173–182, 1987.

[47] D. G. Luenberger. The conjugate residual method for constrained minimization problems. *SIAM J. on Numerical Analysis*, 7:390–398, 1970.

[48] L. Lukšan and J. Vlček. Indefinitely preconditioned inexact Newton method for large sparse equality constrained non-linear programming problems. *Numerical Linear Algebra with Applications*, 5(3):219–247, 1998.

[49] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34:473–497, 1980.

[50] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra and its Applications*, 240:131–151, 1996.

[51] D. Orban. Limited-memory LDLT factorization of symmetric quasi-definite matrices. GERAD Technical Report G-2013-87, 2013.

[52] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. on Numerical Analysis*, 12(4):617–629, 1975.

[53] Beresford N. Parlett. *The symmetric eigenvalue problem.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1980. Prentice-Hall Series in Computational Mathematics.

[54] J. K. Reid and J. A. Scott. Ordering symmetric sparse matrices for small profile and wavefront. *International J. of Numerical Methods in Engineering*, 45:1737–1755, 1999.

[55] J. K. Reid and J. A. Scott. An out-of-core sparse Cholesky solver. *ACM Transactions on Mathematical Software*, 36(2), 2009. Article 9, 33 pages.

[56] J. K. Reid and J. A. Scott. Partial factorization of a dense symmetric indefinite matrix. *ACM Transactions on Mathematical Software*, 38, 2011. article 10, 19 pages.

[57] Y. Saad and M. H. Schulz. Topological properties of hypercubes. Research Report YALE/DCS/RR-389, Department of Computer Science, Yale University, 1985.

[58] Y. Saad and M. H. Schulz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[59] M. A. Saunders and J. A. Tomlin. Solving regularized linear porograms using barrier methods and KKT systems. Technical Report SOL-96-4, SOL, Department of Operations Research, Stanford University, 1996.

[60] O. Schenk, K. Gärtner, and W. Fichtner. Efficient sparse *LU* factorization with left-right looking strategy on shared memory multiprocessors. *BIT*, 40(1):158–176, 2000.

[61] O. Schenk, S. Röllin, and A. Gupta. The Effects of Unsymmetric Matrix Permutations and Scalings in Semiconductor Device and Circuit Simulation. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, 23(3):400 – 411, 2004.

[62] J. A. Scott and M. Tůma. The importance of structure in incomplete factorization preconditioners. *BIT Numerical Mathematics*, 51:385–404, 2011.

[63] J. A. Scott and M. Tůma. HSL_MI28: an efficient and robust limited-memory incomplete Cholesky factorization code. *ACM Trans. Math. Software*, 40(4):Art. 24, 19, 2014.

[64] J. A. Scott and M. Tůma. On positive semidefinite modification schemes for incomplete Cholesky factorization. *SIAM J. on Scientific Computing*, 36(2):A609–A633, 2014.

[65] J. A. Scott and M. Tůma. On signed incomplete Cholesky factorization preconditioners for saddle-point systems. *SIAM J. on Scientific Computing*, 36(6):A2984–A3010, 2014.

[66] S. W. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. *International J. of Numerical Methods in Engineering*, 23:239–251, 1986.

[67] S. W. Sloan. A Fortran program for profile and wavefront reduction. *International J. of Numerical Methods in Engineering*, 28:2651–2679, 1989.

[68] E. Stiefel. Relaxationsmethoden besterr Strategie zur Lösung linearer Gleichungssysteme. *Commentarii Mathematici Helvetici*, 29:157–179, 1955.

[69] M. Tismenetsky. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra and its Applications*, 154–156:331–353, 1991.

[70] R. J. Vanderbei. Symmetric quasidefinite matrices. *SIAM J. on Optimization*, 5(1):100–113, 1995.

[71] E. Vecharynski and A. V. Knyazev. Absolute value preconditioning for symmetric indefinite linear systems. *SIAM J. on Scientific Computing*, 35(2):A696–A718, 2013.

[72] X. Wu, G. H. Golub, J. A. Cuminato, and J. Y. Yuan. Symmetric-triangular decomposition and its applications Part II: Preconditioners for indefinite systems. *BIT*, 48(1):139–162, 2008.